Classical and Quantum Computation

A. Yu. Kitaev
A. H. Shen
M. N. Vyalyi

Graduate Studies in Mathematics
Volume 47

American Mathematical Society

Classical and Quantum Computation

A. Yu. Kitaev
A. H. Shen
M. N. Vyalyi

Graduate Studies
in Mathematics
Volume 47

American Mathematical Society
Providence, Rhode Island

EDITORIAL COMMITTEE

Steven G. Krantz
David Saltman (Chair)
David Sattinger
Ronald Stern

А. Китаев, А. Шень, М. Вялый

КЛАССИЧЕСКИЕ И КВАНТОВЫЕ ВЫЧИСЛЕНИЯ

МЦНМО–ЧеРо, Москва, 1999

Translated from the Russian by Lester J. Senechal

2000 *Mathematics Subject Classification*. Primary 81–02, 68–02;
Secondary 68Qxx, 81P68.

Abstract. The book is an introduction to a new rapidly developing topic: theory of quantum computing. The authors begin with a brief description of complexity theory for classical computations. Then they give a detailed presentation of the basics of quantum computing, including all known efficient quantum algorithms.

The book can be used by graduate and advanced undergraduate students and by researchers working in mathematics, quantum physics, and communication.

For additional information and updates on this book, visit
www.ams.org/bookpages/gsm-47

Library of Congress Cataloging-in-Publication Data

Kitaev, A. Yu. (Alexei Yu.), 1963–
Classical and quantum computation / A. Yu. Kitaev, A. H. Shen, M. N. Vyalyi ; [translated from the Russian by Lester J. Senechal].
p. cm. — (Graduate studies in mathematics, ISSN 1065-7339 ; v. 47)
Includes bibliographical references and index.
ISBN 0-8218-2161-X (acid-free paper) ISBN 0-8218-3229-8 (softcover)
1. Machine theory. 2. Computational complexity. 3. Quantum computers. I. Shen, A. (Alexander), 1958– II. Vyalyi, M. N. (Mikhail N.), 1961– III. Title. IV. Series.
QA267.K57 2002
530.12—dc21 2002016686

Copying and reprinting. Individual readers of this publication, and nonprofit libraries acting for them, are permitted to make fair use of the material, such as to copy a chapter for use in teaching or research. Permission is granted to quote brief passages from this publication in reviews, provided the customary acknowledgment of the source is given.

Republication, systematic copying, or multiple reproduction of any material in this publication is permitted only under license from the American Mathematical Society. Requests for such permission should be addressed to the Acquisitions Department, American Mathematical Society, 201 Charles Street, Providence, Rhode Island 02904-2294 USA. Requests can also be made by e-mail to `reprint-permission@ams.org`.

© 2002 by the American Mathematical Society. All rights reserved.
The American Mathematical Society retains all rights
except those granted to the United States Government.
Printed in the United States of America.

♾ The paper used in this book is acid-free and falls within the guidelines
established to ensure permanence and durability.
Visit the AMS home page at `http://www.ams.org/`

10 9 8 7 6 5 4 3 2 18 17 16 15 14 13

Contents

Foreword

In recent years interest in what is called "quantum computers" has grown extraordinarily. The idea of using the possibilities of quantum mechanics in organizing computation looks all the more attractive now that experimental work has begun in this area.

However, the prospects for physical realization of quantum computers are presently entirely unclear. Most likely this will be a matter of several decades. The fundamental achievements in this area bear at present a purely mathematical character.

This book is intended for a first acquaintance with the mathematical theory of quantum computation. For the convenience of the reader, we give at the outset a brief introduction to the classical theory of computational complexity. The second part includes the descriptions of basic effective quantum algorithms and an introduction to quantum codes.

The book is based on material from the course "Classical and quantum computations", given by A. Shen (classical computations) and A. Kitaev (quantum computations) at the Independent Moscow University in Spring of 1998. In preparing the book we also used materials from the course Physics 229 — Advanced Mathematical Methods of Physics (Quantum Computation) given by John Preskill and A. Kitaev at the California Institute of Technology in 1998–99 (solutions to some problems included in the course were proposed by Andrew Landahl). The original version of this book was published in Russian [**37**], but the present edition extends it in many ways.

The prerequisites for reading this book are modest. In essence, it is enough to know the basics of linear algebra (as studied in a standard university course), elementary probability, basic notions of group theory, and a

few concepts from the theory of algorithms (some computer programming experience may do as well as the formal knowledge). Some topics require an acquaintance with Lie groups and homology of manifolds — but only at the level of definitions.

To reduce the amount of information the reader needs to digest at the first reading, part of the material is given in the form of *problems* and *solutions*. Each problem is assigned a grade according to its difficulty: 1 for an exercise in use of definitions, 2 for a problem that requires some work, 3 for a difficult problem which requires a nontrivial idea. (Of course, the difficulty of a problem is a subjective thing. Also, if several problems are based on the same idea, only the first of them is marked as difficult). The grade appears in square brackets before the problem number. Some problems are marked with an exclamation sign, which indicates that they are almost as important as the main text. Thus, [1!] means an easy but important exercise, whereas [3] is a difficult problem which is safe to skip.

Further reading

In this book we focus on algorithm complexity (in particular, for quantum algorithms), while many related things are not covered. As a general reference on quantum information theory we recommend the book by Michael Nielsen and Isaac Chuang [**51**], which includes such topics as the von Neumann entropy, quantum communication channels, quantum cryptography, fault-tolerant computation, and various proposed schemes for the realization of a quantum computer. Another book on quantum computation and information was written by Josef Gruska [**30**]. Most original papers on the subject can be found in the electronic archive at `http://arXiv.org`, section "Quantum Physics" (`quant-ph`).

Acknowledgements

A. K. thanks Michael Freedman and John Preskill for many inspiring discussions on the topics included in this book. We are grateful to Andrew Landahl for providing the solution to Problem 3.6 and pointing to some inconsistencies in the original manuscript. Among other people who have helped us to improve the book are David DiVincenzo and Barbara Terhal.

Thanks to the people at AMS, and especially to our patient editor Sergei Gelfand and the copy-editor Natalya Pluzhnikov, for their help in bringing this book into reality.

The book was written while A. K. was a member of Microsoft Research and Caltech, and while A. S. and M. V. were members of Independent Moscow University. The preparation of the original Russian version was started

while all three of us were working at IMU, and A. K. was a member of L. D. Landau Institute for Theoretical Physics.

A. K. gratefully acknowledges the support from the National Science Foundation through Caltech's Institute for Quantum Informaiton. M. V. acknowledges the support from the Russian Foundation for Basic Research under grant 02–01–00547.

Notation

$\vee$	disjunction (logical OR)
$\wedge$	conjunction (logical AND)
$\neg$	negation
$\oplus$	addition modulo 2 (and also the direct sum of linear subspaces)
$\textcircled{\oplus}$	controlled NOT gate (p. 62)
$\sqcup$	blank symbol in the alphabet of a Turing machine
$\delta(\cdot,\cdot)$	transition function of a Turing machine
δ_{jk}	Kronecker symbol
$\chi_S(\cdot)$	characteristic function of the set S
$f_\oplus$	invertible function corresponding to the Boolean function f (p. 61)
$\overline{x_{n-1}\cdots x_0}$	number represented by binary digits $x_{n-1},\ldots,x_0$
$\gcd(x,y)$	greatest common divisor of x and y
$a \bmod q$	residue of a modulo q
$\boxed{\begin{array}{c}a\\b\end{array}}$	representation of the rational number a/b in the form of an irreducible fraction
$a \mid b$	a divides b
$a \equiv b \pmod q$	a is congruent to b modulo q
$A \Rightarrow B$	A implies B
$A \Leftrightarrow B$	A is logically equivalent to B
$L_1 \propto L_2$	Karp reduction of predicates (L_1 can be reduced to L_2 (p. 30))
$\lfloor x \rfloor$	the greatest integer not exceeding x
$\lceil x \rceil$	the least integer not greater than x

A^*	set of all finite words in the alphabet A
E^*	group of characters on the Abelian group E, i.e., $\mathrm{Hom}(E, \mathbf{U}(1))$
z^*	complex conjugate of z
$\mathcal{M}^*$	space of linear functionals on the space $\mathcal{M}$
$\langle\xi\|$	bra-vector (p. 56)
$\|\xi\rangle$	ket-vector (p. 56)
$\langle\xi\|\eta\rangle$	inner product
$A^\dagger$	Hermitian adjoint operator
$\widehat{G}$	unitary operator corresponding to the permutation G (p. 61)
$I_{\mathcal{L}}$	identity operator on the space $\mathcal{L}$
$\Pi_{\mathcal{M}}$	projection (the operator of projecting onto the subspace $\mathcal{M}$)
$\mathrm{Tr}_{\mathcal{F}} A$	partial trace of the operator A over the space (tensor factor) $\mathcal{F}$ (p. 96)
$A \cdot B$	superoperator $\rho \mapsto A\rho B$ (p. 108)
$\mathcal{M}^{\otimes n}$	n-th tensor degree of $\mathcal{M}$
$\mathbb{C}(a, b, \dots)$	space generated by the vectors $a, b, \dots$
$\Lambda(U)$	operator U with quantum control (p. 65)
$U[A]$	application of the operator U to a quantum register (set of qubits) A (p. 58)
$\mathcal{E}[A]$, $\mathcal{E}(n, k)$	error spaces (p. 156)
$\sigma(\alpha_1, \beta_1, \dots, \alpha_n, \beta_n)$	basis operators on the space $\mathcal{B}^{\otimes n}$ (p. 162)
$\mathrm{SympCode}(F, \mu)$	symplectic code (p. 168)
$\|\cdot\|$	cardinality of a set or modulus of a number
$\|\cdot\|$	norm of a vector (p. 71)
	or operator norm (p. 71)
$\|\cdot\|_{\mathrm{tr}}$	trace norm (p. 98)
$\|\cdot\|_{\diamondsuit}$	superoperator norm (p. 110)
$\mathbf{Pr}[A]$	probability of the event A
$\mathbf{P}(\cdot\,|\,\cdot)$	conditional probability (in various contexts)
$\mathbf{P}(\rho, \mathcal{M})$	quantum probability (p. 95)
$f(n) = O(g(n))$	there exist numbers C and n_0 such that $f(n) \le Cg(n)$ for all $n \ge n_0$
$f(n) = \Omega(g(n))$	there exist numbers C and n_0 such that $f(n) \ge Cg(n)$ for all $n \ge n_0$
$f(n) = \Theta(g(n))$	$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ at the same time
$f(n) = \mathrm{poly}(n)$	means the same as $f(n) = n^{O(1)}$
$\mathrm{poly}(n, m)$	abbreviation for $\mathrm{poly}(n + m)$

$\mathbb{N}$	set of natural numbers, i.e., $\{0, 1, 2, \dots\}$
$\mathbb{Z}$	set of integers
$\mathbb{R}$	set of real numbers
$\mathbb{C}$	set of complex numbers
$\mathbb{B}$	classical bit (set $\{0, 1\}$)
$\mathcal{B}$	quantum bit (qubit, space $\mathbb{C}^2$ — p. 53)
$\mathbb{F}_q$	finite field of q elements
$\mathbb{Z}/n\mathbb{Z}$	ring of residues modulo n
$\mathbb{Z}_n$	additive group of the ring $\mathbb{Z}/n\mathbb{Z}$
$(\mathbb{Z}/n\mathbb{Z})^*$	multiplicative group of invertible elements of $\mathbb{Z}/n\mathbb{Z}$
$\mathrm{Sp}_2(n)$	symplectic group of order n over the field $\mathbb{F}_2$ (p. 165)
$\mathrm{ESp}_2(n)$	extended symplectic group of order n over the field $\mathbb{F}_2$ (p. 164)
$\mathbf{L}(\mathcal{N})$	space of linear operators on $\mathcal{M}$
$\mathbf{L}(\mathcal{N}, \mathcal{M})$	space of linear operators from $\mathcal{N}$ to $\mathcal{M}$
$\mathbf{U}(\mathcal{M})$	group of unitary operators in the space $\mathcal{M}$
$\mathbf{SU}(\mathcal{M})$	special unitary group in the space $\mathcal{M}$
$\mathbf{SO}(\mathcal{M})$	special orthogonal group in the Euclidean space $\mathcal{M}$

Notation for matrices:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad K = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix},$$

Pauli matrices: $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$, $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

Notation for complexity classes:

NC	(p. 23)	NP	(p. 28)	BQP	(p. 91)
P	(p. 14)	MA	(p. 138)	BQNP	(p. 139)
BPP	(p. 36)	Π_k	(p. 46)	PSPACE	(p. 15)
PP	(p. 91)	Σ_k	(p. 46)	EXPTIME	(p. 22)
P/poly	(p. 20)				

Introduction

All computers, beginning with Babbage's unconstructed "analytical machine"[1] and ending with the Cray, are based on the very same principles. From the logical point of view a computer consists of bits (variables, taking the values 0 and 1), and a program — that is, a sequence of operations, each using some bits. Of course, the newest computers work faster than old ones, but progress in this direction is limited. It is hard to imagine that the size of a transistor or another element will ever be smaller than 10^{-8} cm (the diameter of the hydrogen atom) or that the clock frequency will be greater than 10^{15} Hz (the frequency of atomic transitions), so that even the supercomputers of the future will not be able to solve computational problems having *exponential* complexity. Let us consider, for example, the problem of factoring an integer number x into primes. The obvious method is to attempt to divide x by all numbers from 2 to $\sqrt{x}$. If x has n digits (as written in the binary form), we need to go through $\sim \sqrt{x} \sim 2^{n/2}$ trials. There exists an ingenious algorithm that solves the same problem in approximately $\exp(cn^{1/3})$ steps (c is a constant). But even so, to factor a number of a million digits, a time equal to the age of the Universe would not suffice. (There may exist more effective algorithms, but it seems impossible to dispense with the exponential.)

There is, however, another way of speeding up the calculation process for several special classes of problems. The situation is such that ordinary computers do not employ all the possibilities that are offered to us by nature.

[1] Charles Babbage began his work on the "analytical machine" project in 1833. In contrast to calculating devices already existed at the time, his was supposed to be a *universal* computer. Babbage devoted his whole life to its development, but was not successful in realizing his dream. (A simpler, nonuniversal machine was *partially* constructed. In fact, this smaller project could have been completed — in 1991 the machine was produced in accordance with Babbage's design.)

This assertion may seem extremely obvious: in nature there exists a multitude of processes that are unlike operations with zeros and ones. We might attempt to use those processes for the creation of an analog computer. For example, interference of light can be used to compute the Fourier transform. However, in most cases the gain in speed is not major, i.e., it depends weakly on the size of the device. The reason lies in the fact that the equations of *classical* physics (for example, Maxwell's equations) can be effectively solved on an ordinary digital computer. What does "effectively" mean? The calculation of an interference pattern may require more time than the real experiment by a factor of a million, because the speed of light is great and the wave length is small. However, as the size of the modelled physical system gets bigger, the required number of computational operations grows at a moderate rate — as the size raised to some power or, as is customarily said in complexity theory, *polynomially*. (As a rule, the number of operations is proportional to the quantity Vt, where V is the volume and t is the time.) Thus we see that classical physics is too "simple" from the computational point of view.

Quantum mechanics is more interesting from this perspective. Let us consider, for example, a system of n spins. Each spin has two so-called *basis states* (0 = "spin up" and 1 = "spin down"), and the whole system has 2^n basis states $|x_1, \ldots, x_n\rangle$ (each of the variables $x_1, \ldots, x_n$ takes values 0 or 1). By a general principle of quantum mechanics, $\sum_{x_1,\ldots,x_n} c_{x_1,\ldots,x_n}|x_1, \ldots, x_n\rangle$ is also a possible state of the system; here $c_{x_1,\ldots,x_n}$ are complex numbers called *amplitudes*. The summation sign must be understood as a pure formality. In fact, the "sum" (also called a *superposition*) represents a new mathematical object — a vector in a 2^n-dimensional complex vector space. Physically, $|c_{x_1,\ldots,x_n}|^2$ is the probability to find the system in the basis state $|x_1, \ldots, x_n\rangle$ by a measurement of the values of the variables x_j. (We note that such a measurement destroys the superposition.) For this to make sense, the formula $\sum_{x_1,\ldots,x_n} |c_{x_1,\ldots,x_n}|^2 = 1$ must hold. Therefore, the general state of the system (i.e., a superposition) is a unit vector in the 2^n-dimensional complex space. A state change over a specified time interval is described by a unitary matrix of size $2^n \times 2^n$. If the time interval is very small ($\ll \hbar/J$, where J is the energy of spin-spin interaction and $\hbar$ is Planck's constant), then this matrix is rather easily constructed; each of its elements is easily calculated knowing the interaction between the spins. If, however, we want to compute the change of the state over a large time interval, then it is necessary to multiply such matrices. For this purpose an exponentially large number of operations is needed. Despite much effort, no method has been found to simplify this computation (except for some special cases). Most plausibly, simulation of quantum mechanics is indeed an exponentially hard computational problem. One may think this is unfortunate, but let us take

a different point of view: quantum mechanics being *hard* means it is *powerful*. Indeed, a quantum system effectively "solves" a complex computational problem — it models its very self.

Can we use quantum systems for solving other computational problems? What would be a mathematical model of a quantum computer that is just as independent of physical realization as are models of classical computation?[2] It seems that these questions were first posed in 1980 in the book by Yu. I. Manin [**49**]. They were also discussed in the works of R. Feynman [**23, 24**] and other authors. In 1985 D. Deutsch [**20**] proposed a concrete mathematical model — the quantum Turing machine, and in 1989 an equivalent but more convenient model — the quantum circuit [**21**] (the latter was largely based on Feynman's ideas).

What exactly is a quantum circuit? Suppose that we have N spins, each located in a separate numbered compartment and completely isolated from the surrounding world. At each moment of time (as a computer clock ticks) we choose, at our discretion, any two spins and act on them with an arbitrary 4×4 unitary matrix. A sequence of such operations is called a *quantum circuit*. Each operation is determined by a pair of integers, indexing the spins, and sixteen complex numbers (the matrix entries). So a quantum circuit is a kind of computer program, which can be represented as text and written on paper. The word "quantum" refers to the way this program is executed.

Let us try to use a quantum circuit for calculating a function $F : \mathbb{B}^n \to \mathbb{B}^m$, where $\mathbb{B} = \{0, 1\}$ is the set of values of a classical bit.[3] It is necessary to be able to enter the initial data, perform the computations, and read out the result. Input into a quantum computer is a sequence $(x_1, \ldots, x_n)$ of zeros and ones — meaning that we prepare an initial state $|x_1, \ldots, x_n, 0, \ldots, 0\rangle$. (The amount of initial data, n, is usually smaller than the overall number of "memory cells," i.e., of spins, N. The remaining cells are filled with zeros.) The initial data are fed into a quantum circuit, which depends on the *problem being solved*, but not on the specific initial data. The circuit turns the initial state into a new quantum state,

$$|\psi(x_1, \ldots, x_n)\rangle = \sum_{y_1,\ldots,y_N} c_{y_1,\ldots,y_N}(x_1, \ldots, x_n)\, |y_1, \ldots, y_N\rangle,$$

[2] The standard mathematical model of an ordinary computer is the Turing machine. Most other models in use are polynomially equivalent to this one and to each other, i.e., a problem, that is solvable in L steps in one model, will be solvable in cL^k steps in another model, where c and k are constants.

[3] Any computational problem can be posed in this way. For example, if we wish to solve the problem of factoring an integer into primes, then $(x_1, \ldots, x_n) = x$ (in binary notation) and $F(x)$ is a list of prime factors (in some binary code).

which depends on $(x_1, \ldots, x_n)$. It is now necessary to read out the result. If the circuit were classical (and correctly designed to compute F), we would expect to find the answer in the first m bits of the sequence $(y_1, \ldots, y_N)$, i.e., we seek $(y_1, \ldots, y_m) = F(x_1, \ldots, x_n)$. To determine the actual result in the quantum case, the values of all spins should be *measured*. The measurement may produce *any* sequence of zeros and ones $(y_1, \ldots, y_N)$, the probability of obtaining such a sequence being equal to $\left|c_{y_1,\ldots,y_N}(x_1, \ldots, x_n)\right|^2$. A quantum circuit is "correct" for a given function F if the correct answer $(y_1, \ldots, y_m) = F(x_1, \ldots, x_n)$ is obtained with *probability* that is sufficiently close to 1. By repeating the computation several times and choosing the answer that is encountered most frequently, we can reduce the probability of an error to be as small as we want.

We have just formulated (omitting some details) a mathematical model of quantum computation. Now, two questions arise naturally.

1. For which problems does quantum computation have an advantage in comparison with classical?
2. What system can be used for the physical realization of a quantum computer? (This does not necessarily have to be a system of spins.)

With regard to the first question we now know the following. First, on a quantum computer it is possible to model an arbitrary quantum system in *polynomially* many steps. This will allow us (when quantum computers become available) to predict the properties of molecules and crystals and to design microscopic electronic devices, say, 100 atoms in size. Presently such devices lie at the edge of technological possibility, but in the future they will likely be common elements of ordinary computers. So, a quantum computer will not be a thing to have in every home or office, but it will be used to make such things.

A second example is factoring integers into primes and analogous number-theoretic problems. In 1994 P. Shor [**62**] found a quantum algorithm[4] which factors an n-digit integer in about n^3 steps. This beautiful result could have an outcome that is more harmful than useful: factoring allows one to break the most commonly used cryptosystem (RSA), to forge electronic signatures, etc. (But anyway, building a quantum computer is such a difficult task that cryptography users may have good sleep — at least, for the next 10 years.) The method at the core of Shor's algorithms deals with Abelian groups. Some non-Abelian generalizations have been found, but it remains to be seen if they can be applied to any practical problem.

[4]Without going into detail, a quantum algorithm is much the same thing as a quantum circuit. The difference lies in the fact that a circuit is defined for problems of fixed size ($n = \text{const}$), whereas an algorithm applies to any n.

A third example is a search for a needed entry in an unsorted database. Here the gain is not so significant: to locate one entry in N we need about $\sqrt{N}$ steps on a quantum computer, compared to N steps on a classical one. As of this writing, these are all known examples — not because quantum computers are useless for other problems, but because their theory has not been worked out yet. We can hope that there will soon appear new mathematical ideas that will lead to new quantum algorithms.

The physical realization of a quantum computer is an exceedingly interesting, but difficult problem. Only a few years ago doubts were expressed about its solvability in principle. The trouble is that an arbitrary unitary transformation can be realized only with certain accuracy. Apart from that, a system of spins or a similar quantum system cannot be fully protected from the disturbances of the surrounding environment. All this leads to errors that accumulate in the computational process. In $L \sim \delta^{-1}$ steps (where δ is the precision of each unitary transformation) the probability of an error will be of the order of 1, which renders the computation useless. In part this difficulty can be overcome using *quantum error-correcting codes*. In 1996 P. Shor [**65**] proposed a scheme of error correction in the quantum computing process (fault-tolerant quantum computation). The original method was not optimal but it was soon improved by a number of authors. The end result amounts to the following. There exists some threshold value δ_0 such that for any precision $\delta < \delta_0$ arbitrarily long quantum computation is possible. However, for $\delta > \delta_0$ errors accumulate faster than we can succeed in correcting them. By various estimates, δ_0 lies in the interval from 10^{-6} to 10^{-2} (the exact value depends on the character of the disturbances and the circuit that is used for error correction).

So, there are no obstacles in principle for the realization of a quantum computer. However, the problem is so difficult that it can be compared to the problem of controlled thermonuclear synthesis. In fact, it is essential to simultaneously satisfy several almost contradictory demands:

1. The elements of a quantum computer — quantum bits (spins or something similar) — must be isolated from one another and from the environment.
2. It is essential to have the possibility to act selectively on each pair of quantum bits (at least, on each neighboring pair). Generally, one needs to implement several types of elementary operations (called *quantum gates*) described by different unitary operators.
3. Each of the gates must be realized with precision $\delta < \delta_0$ (see above).
4. The quantum gates must be sufficiently nontrivial, so that any other operator is, in a certain sense, expressible in terms of them.

At the present time there exist several approaches to the problem of realizing a quantum computer.

1. Individual atoms or ions. This first-proposed and best-developed idea exists in several variants. For representing a quantum bit one can employ both the usual electron levels and the levels of fine and superfine structures. There is an experimental technique for keeping an individual ion or atom in the trap of a steady magnetic or alternating electric field for a reasonably long time (of the order of 1 hour). The ion can be "cooled down" (i.e., its vibrational motion eliminated) with the aid of a laser beam. Selecting the duration and frequency of the laser pulses, it is possible to prepare an arbitrary superposition of the ground and excited states. In this way it is rather easy to control individual ions. Within the trap, one can also place two or more ions at distances of several microns one from another, and control each of them individually. However, it is rather difficult to choreograph the interactions between the ions. To this end it has been proposed that collective vibrational modes (ordinary mechanical vibrations with a frequency of several MHz) be used. Dipole-dipole interactions could also be used, with the advantage of being a lot faster. A second method (for neutral atoms) is as follows: place atoms into separate electromagnetic resonators that are coupled to one another (at the moment it is unclear how to achieve this technically). Finally, a third method: using several laser beams, one can create a periodic potential ("optical lattice") which traps unexcited atoms. However, an atom in an excited state can move freely. Thus, by exciting one of the atoms for a certain time, one lets it move around and interact with its neighbors. This field of experimental physics is now developing rapidly and seems to be very promising.

2. Nuclear magnetic resonance. In a molecule with several different nuclear spins, an arbitrary unitary transformation can be realized by a succession of magnetic field pulses. This has been tested experimentally at room temperature. However, for the preparation of a suitable initial state, a temperature $< 10^{-3}$ K is required. Apart from difficulties with the cooling, undesirable interactions between the molecules increase dramatically as the liquid freezes. In addition, it is nearly impossible to address a given spin selectively if the molecule has several spins of the same kind.

3. Superconducting granules and "quantum dots". Under supercool temperatures, the unique degree of freedom of a small (submicron size) superconducting granule is its charge. It can change in magnitude by a multiple of two electron charges (since electrons in a superconductor are bound in pairs). Changing the external electric potential, one can achieve a situation where two charge states have almost the same energy. These two states can be used as basis states of a quantum bit. The granules interact with each other by means of Josephson junctions and mutual electric

capacitance. This interaction can be controlled. A quantum dot is a microstructure which can contain few electrons or even a single electron. The spin of this electron can be used as a qubit. The difficulty is that one needs to control each granule or quantum dot individually with high precision. This seems harder than in the case of free atoms, because all atoms of the same type are identical while parameters of fabricated structures fluctuate. This approach may eventually succeed, but a new technology is required for its realization.

4. Anyons. Anyons are quasi-particles (excitations) in certain two-dimensional quantum systems, e.g. in a two-dimensional electron liquid in magnetic field. What makes them special is their *topological properties*, which are stable to moderate variation of system parameters. One of the authors (A. K.) considers this approach especially interesting (in view of it being his own invention, cf. [**35**]), so that we will describe it in more detail. (At a more abstract level, the connection between quantum computation and topology was discussed by M. Freedman [**26**].)

The fundamental difficulty in constructing a quantum computer is the necessity for realizing unitary transformations with precision $\delta < \delta_0$, where δ_0 is between 10^{-2} and 10^{-6}. To achieve this it is necessary, as a rule, to control the parameters of the system with still greater precision. However, we can imagine a situation where high precision is achieved automatically, i.e., where error correction occurs on the physical level. An example is given by two-dimensional systems with anyonic excitations.

All particles in three-dimensional space are either bosons or fermions. The wave function of bosons does not change if the particles are permuted. The wave function of fermions is multiplied by -1 under a transposition of two particles. In any case, the system is unchanged when each of the particles is returned to its prior position. In two-dimensional systems, more complex behavior is possible. Note, however, that the discussion is not about fundamental particles, such as an electron, but about excitations ("defects") in a two-dimensional electron liquid. Such excitations can move, transform to each other, etc., just like "genuine" particles.[5] However, excitations in the two-dimensional electron liquid display some unusual properties. An excitation can have a fractional charge (for example, 1/3 of the charge of an electron). If one excitation makes a full turn *around another*, the state of the surrounding electron liquid changes in a precisely defined manner that depends on the types of the excitations and on the *topology* of the path, but not on the specific trajectory. In the simplest case, the wave function gets multiplied by a number (which is equal to $e^{2\pi i/3}$ for anyons in

[5]Fundamental particles can also be considered as excitations in the vacuum which is, actually, a nontrivial quantum system. The difference is that the vacuum is unique, whereas the electron liquid and other "quantum media" can be designed to meet our needs.

the two-dimensional electron liquid in a magnetic field at the filling factor 1/3). Excitations with such properties are called *Abelian anyons*. Another example of Abelian anyons is described (in a mathematical language) in Section 15.11.

More interesting are *non-Abelian anyons*, which have not yet been observed experimentally. (Theory predicts their existence in a two-dimensional electron liquid in a magnetic field at the filling factor 5/2.) In the presence of non-Abelian anyons, the state of the surrounding electron liquid is degenerate, the multiplicity of the degeneracy depending on the number of anyons. In other words, there exist not one, but many states, which can display arbitrary quantum superpositions. It is utterly impossible to act on such a superposition without moving the anyons, so the system is ideally protected from perturbations. If one anyon is moved around another, the superposition undergoes a certain unitary transformation. This transformation is *absolutely precise*. (An error can occur only if the anyon "gets out of hand" as a result of quantum tunneling.)

At first glance, the design using anyons seems least realistic. Firstly, Abelian anyons will not do for quantum computation, and non-Abelian ones are still awaiting experimental discovery. But in order to realize a quantum computer, it is necessary to control (i.e., detect and drag by a specified path) *each* excitation in the system, which will probably be a fraction of a micron apart from each other. This is an exceedingly complex technical problem. However, taking into account the high demands for precision, it may not be at all easier to realize any of the other approaches we have mentioned. Beyond that, the idea of *topological quantum computation*, lying at the foundation of the anyonic approach, might be expedited by other means. For example, the quantum degree of freedom protected from perturbation, might shoot up at the end of a "quantum wire" (a one-dimensional conductor with an odd number of propagating electronic modes, placed in contact with a three-dimensional superconductor).

Thus, the idea of a quantum computer looks so very attractive, and so very unreal. It is likely that the design of an ordinary computer was perceived in just that way at the time of Charles Babbage, whose invention was realized only a hundred years later. We may hope that in our time the science and the industry will develop faster, so that we will not have to wait that long. Perhaps a couple of fresh ideas plus a few years for working out a new technology will do.

Part 1

Classical Computation

1. Turing machines

Note. In this section we address the abstract notion of computability, of which we only need a few basic properties. Therefore our exposition here is very brief. For the most part, the omitted details are simply exercises in programming a Turing machine, which is but a primitive programming language. A little of programming experience (in any language) suffices to see that these tasks are doable but tedious.

Informally, an *algorithm* is a set of instructions; using it, "we need only to carry out what is prescribed as if we were robots: neither understanding, nor cleverness, nor imagination is required of us" [**39**]. Applying an algorithm to its *input* (initial data) we get some *output* (result). (It is quite possible that computation never terminates for some inputs; in this case we get no result.)

Usually inputs and outputs are strings. A *string* is a finite sequence of symbols (characters, letters) taken from some finite *alphabet*. Therefore, before asking for an algorithm that, say, factors polynomials with integer coefficients, we should specify the encoding, i.e., specify some alphabet A and the representation of polynomials by strings over A. For example, each polynomial may be represented by a string formed by digits, letter x, signs $+$, $-$ and $\times$. In the answer, two factors can be separated by a special delimiter, etc.

One should be careful here because sometimes the encoding becomes really important. For example, if we represent large integers as bit strings (in binary), it is rather easy to compare them (to find which of two given integers is larger), but multiplication is more difficult. On the other hand, if an

integer is represented by its remainders modulo different primes $p_1, p_2, \ldots, p_n$ (using the Chinese remainder theorem; see Theorem A.5 in Appendix A), it is easy to multiply them, but comparison is more difficult. So we will specify the encoding in case of doubt.

We now give a formal definition of an algorithm.

1.1. Definition of a Turing machine.

Definition 1.1. A Turing machine (TM) consists of the following components:

- a finite set S called the *alphabet*;
- an element $\sqcup \in S$ (blank symbol);
- a subset $A \subset S$ called the *external alphabet*; we assume that the blank symbol does not belong to A;
- a finite set Q whose elements are called *states* of the TM;
- an *initial state* $q_0 \in Q$;
- a *transition function*, specifically, a partial function

$$\delta : Q \times S \to Q \times S \times \{-1, 0, 1\}. \tag{1.1}$$

(The term "*partial function*" means that the domain of δ is actually a subset of $Q \times S$. A function that is defined everywhere is called *total*.)

Note that there are infinitely many Turing machines, each representing a particular algorithm. Thus the above components are more like a computer program. We now describe the "hardware" such programs run on.

A Turing machine has a *tape* that is divided into *cells*. Each cell carries one symbol from the machine alphabet S. We assume that the tape is infinite to the right. Therefore, the content of the tape is an infinite sequence $\sigma = s_0, s_1, \ldots$ (where $s_i \in S$).

A Turing machine also has a read-write *head* that moves along the tape and changes symbols: if we denote its position by $p = 0, 1, 2, \ldots$, the head can read the symbol s_p and write another symbol in its place.

Position of head				▽	
Cells	s_0	s_1	$\ldots$	s_p	$\ldots$
Cell numbers	0	1		p	

The behavior of a Turing machine is determined by a control device, which is a finite-state automaton. At each step of the computation this device is in some state $q \in Q$. The state q and the symbol s_p under the head determine the action performed by the TM: the value of the transition function, $\delta(q, s_p) = (q', s', \Delta p)$, contains the new state q', the new symbol

s', and the shift Δp (for example, $\Delta p = -1$ means that the head moves to the left).

More formally, the *configuration* of a TM is a triple $\langle \sigma; p; q\rangle$, where σ is an infinite sequence $s_0, \dots, s_n, \dots$ of elements of S, p is a nonnegative integer, and $q \in Q$. At each step the TM changes its configuration $\langle \sigma; p; q\rangle$ as follows:

(a) it reads the symbol s_p;

(b) it computes the value of the transition function: $\delta(q, s_p) = (q', s', \Delta p)$ (if $\delta(q, s_p)$ is undefined, the TM stops);

(c) it writes the symbol s in cell p of the tape, moves the head by Δp, and passes to state q'. In other words, the new configuration of the TM is the triple $\langle s_0, \dots, s_{p-1}, s', s_{p+1}, \dots; p+\Delta p; q'\rangle$. (If $p + \Delta p < 0$, the TM stops.)

Perhaps everyone would agree that these actions require neither cleverness, nor imagination.

It remains to define how the input is given to the TM and how the result is obtained. Inputs and outputs are strings over A. An input string α is written on the tape and is padded by blanks. Initially the head is at the left end of the tape; the initial state is q_0. Thus the initial configuration is $\langle \alpha\text{␣␣}\dots; 0; q_0\rangle$. Subsequently, the configuration is transformed step by step using the rules described above, and we get the sequence of configurations

$$\langle \alpha\text{␣␣}\dots; 0; q_0\rangle,\ \langle \sigma_1; p_1; q_1\rangle,\ \langle \sigma_2; p_2; q_2\rangle, \dots\ .$$

As we have said, this process terminates if δ is undefined or the head bumps into the (left) boundary of the tape ($p + \Delta p < 0$). After that, we read the tape from left to right (starting from the left end) until we reach some symbol that does not belong to A. The string before that symbol will be the output of the TM.

1.2. Computable functions and decidable predicates. Every Turing machine M *computes* a partial function $\varphi_M \colon A^* \to A^*$, where A^* is the set of all strings over A. By definition, $\varphi_M(\alpha)$ is the output string for input α. The value $\varphi_M(\alpha)$ is undefined if the computation never terminates.

Definition 1.2. A partial function f from A^* to A^* is *computable* if there exists a Turing machine M such that $\varphi_M = f$. In this case we say that f is *computed by* M.

Not all functions are computable because the set of all functions of type $A^* \to A^*$ is uncountable, while the set of all Turing machines is countable. For concrete examples of noncomputable functions see Problems 1.3–1.5.

By a *predicate* we mean a function with Boolean values: 1 ("true") or 0 ("false"). Informally, a predicate is a property that can be true or false. Normally we consider predicates whose domain is the set A^* of all strings over some alphabet A. Such predicates can be identified with subsets of A^*: a predicate P corresponds to the set $\{x : P(x)\}$, i.e., the set of strings x for which $P(x)$ is true. Subsets of A^* are also called *languages* over A.

As has been said, a predicate P is a function $A^* \to \{0, 1\}$. A predicate is called *decidable* if this function is computable. In other words, a predicate P is decidable if there exists a Turing machine that answers question "is $P(\alpha)$ true?" for any $\alpha \in A^*$, giving either 1 ("yes") or 0 ("no"). (Note that this machine must terminate for any $\alpha \in A^*$.)

The notions of a computable function and a decidable predicate can be extended to functions and predicates in several variables in a natural way. For example, we can fix some separator symbol # that does not belong to A and consider a Turing machine M with external alphabet $A \cup \{\#\}$. Then a partial function $\varphi_{M,n} \colon (A^*)^n \to A^*$ is defined as follows:

$$\varphi_{M,n}(\alpha_1, \ldots, \alpha_n) = \text{ output of } M \text{ for the input } \alpha_1 \# \alpha_2 \# \cdots \# \alpha_n.$$

The value $\varphi_{M,n}(\alpha_1, \ldots, \alpha_n)$ is undefined if the computation never terminates or the output string does not belong A^*.

Definition 1.3. A partial function f from $(A^*)^n$ to A^* is *computable* if there is a Turing machine M such that $\varphi_{M,n} = f$.

The definition of a decidable predicate can be given in the same way.

We say that a Turing machine works in time $T(n)$ if it performs at most $T(n)$ steps for any input of size n. Analogously, a Turing machine M works in space $s(n)$ if it visits at most $s(n)$ cells for any computation on inputs of size n.

1.3. Turing's thesis and universal machines. Obviously a TM is an algorithm in the informal sense. The converse assertion is called the *Turing thesis*:

"*Any algorithm can be realized by a Turing machine.*"

It is called also the Church thesis because Church gave an alternative definition of computable functions that is formally equivalent to Turing's definition. Note that the Church-Turing thesis is not a mathematical theorem, but rather a statement about our informal notion of algorithm, or the physical reality this notion is based upon. Thus the Church-Turing thesis cannot be proved, but it is supported by empirical evidence. At the early age of mathematical computation theory (1930's), different definitions of

algorithm were proposed (Turing machine, Post machine, Church's lambda-calculus, Gödel's theory of recursive functions), but they all turned out to be equivalent to each other. The reader can find a detailed exposition of the theory of algorithms in [**5, 39, 40, 48, 54, 60, 61**].

We make some informal remarks about the capabilities of Turing machines. A Turing machine behaves like a person with a restricted memory, pencil, eraser, and a notebook with an infinite number of pages. Pages are of fixed size; therefore there are finitely many possible variants of filling a page, and these variants can be considered as letters of the alphabet of a TM. The person can work with one page at a time but can then move to the previous or to the next page. When turning a page, the person has a finite amount of information (corresponding to the state of the TM) in her head.

The input string is written on several first pages of the notebook (one letter per page); the output should be written in a similar way. The computation terminates when the notebook is closed (the head crosses the left boundary) or when the person does not know what to do next (δ is undefined).

Think about yourself in such a situation. It is easy to realize that by memorizing a few letters near the head you can perform any action in a fixed-size neighborhood of the head. You can also put extra information (in addition to letters from the external alphabet) on pages. This means that you extend the tape alphabet by taking the Cartesian product with some other finite set that represents possible notes. You can leaf through the notebook until you find a note that is needed. You can create a free cell by moving all information along the tape. You can memorize symbols and then copy them onto free pages of the notebook. Extra space on pages may also be used to store auxiliary strings of arbitrary length (like the initial word, they are written one symbol per page). These auxiliary strings can be processed by "subroutines". In particular, auxiliary strings can be used to implement counters (integers that can be incremented and decremented). Using counters, we can address a memory cell by its number, etc.

Note that in the definition of computability for functions of type $A^* \to A^*$ we restrict neither the number of auxiliary tape symbols (the set of symbols S could be much bigger than A) nor the number of states. It is easy to see, however, that one auxiliary symbol (the blank) is enough. Indeed, we can represent each letter from $S \setminus A$ by a combination of blanks and nonblanks. (The details are left to the reader as an exercise.)

Since a Turing machine is a finite object (according to Definition 1.1), it can be encoded by a string. (Note that Turing machines with arbitrary numbers of states and alphabets of any size can be encoded by strings over a fixed alphabet.) Then for any fixed alphabet A we can consider a *universal*

Turing machine U. Its input is a pair $([M], x)$, where $[M]$ is the encoding of a machine M with external alphabet A, and x is a string over A. The output of U is $\varphi_M(x)$. Thus U computes the function u defined as follows:

$$u\big([M], x\big) = \varphi_M(x).$$

This function is *universal* for the class of computable functions of type $A^* \to A^*$ in the following sense: for any computable function $f : A^* \to A^*$, there exists some M such that $u([M], x) = f(x)$ for all $x \in A^*$. (The equality actually means that either both $u(M, x)$ and $f(x)$ are undefined, or they are defined and equal. Sometimes the notation $u([M], x) \simeq f(x)$ is used to stress that both expressions can be undefined.)

The existence of a universal machine U is a consequence of the Church-Turing thesis since our description of Turing machines was algorithmic. But, unlike the Church-Turing thesis, this is also a mathematical theorem: the machine U can be constructed explicitly and proved to compute the function u. The construction is straightforward but boring. It can be explained as follows: the notebook begins with pages where instructions (i.e., $[M]$) are written; the input string x follows the instructions. The universal machine interprets the instructions in the following way: it marks the current page, goes back to the beginning of the tape, finds the instruction that matches the current state and the current symbol, then returns to the current page and performs the action required. Actually, the situation is a bit more complex: both the current state and the current symbol of M have to be represented in U by several symbols on the tape (because the number of states of the universal machine U is fixed whereas the alphabet and the number of states of the simulated machine M are arbitrary). Therefore, we need subroutines to move strings along the tape, compare them with instructions, etc.

1.4. Complexity classes. The computability of a function does not guarantee that we can compute it in practice: an algorithm computing it can require too much time or space. So from the practical viewpoint we are interested in *effective* algorithms.

The idea of an effective algorithm can be formalized in different ways, leading to different complexity classes. Probably the most important is the class of *polynomial* algorithms.

We say that a function $f(n)$ is of *polynomial growth* if $f(n) \le cn^d$ for some constants c, d and for all sufficiently large n. (Notation: $f(n) = \mathrm{poly}(n)$.)

Let $\mathbb{B}$ be the set $\{0, 1\}$. In the sequel we usually consider functions and predicates defined on $\mathbb{B}^*$, i.e., on binary strings.

Definition 1.4. A function F on $\mathbb{B}^*$ is *computable in polynomial time* if there exists a Turing machine that computes it in time $T(n) = \mathrm{poly}(n)$,

where n is the length of the input. If F is a predicate, we say that it is *decidable in polynomial time*.

The class of all functions computable in polynomial time, or all predicates decidable in polynomial time (sometimes we call them "polynomial predicates" for brevity), is denoted by P. (Some other complexity classes considered below are only defined for predicates.) Note that if F is computable in polynomial time, then $|F(x)| = \mathrm{poly}(|x|)$, since the output length cannot exceed the maximum of the input length and the number of computation steps. (Here $|t|$ stands for the length of the string t.)

The computability in polynomial time is still a theoretic notion: if the degree of the polynomial is large (or the constant c is large), an algorithm running in polynomial time may be quite impractical.

One may use other computational models instead of Turing machines to define the class P. For example, we may use a usual programming language dealing with integer variables, if we require that all integers used in the program have at most $\mathrm{poly}(n)$ bits.

In speaking about polynomial time computation, one should be careful about encoding. For example, it is easy to see that the predicate that is true for all *unary* representations of prime numbers (i.e., strings $1\ldots1$ whose length N is a prime number) is polynomial. Indeed, the obvious algorithm that tries to divide N by all numbers $\le \sqrt{N}$ runs in polynomial time, namely, $\mathrm{poly}(N)$. On the other hand, we do not know whether the predicate $P(x) =$ "x is a binary representation of a prime number" is polynomial or not. For this to be true, there should exist an algorithm with running time $\mathrm{poly}(n)$, where $n = \lfloor \log_2 N \rfloor$ is the length of the binary string x. (A probabilistic polynomial algorithm for this problem is known; see below.)

Definition 1.5. A function (predicate) F on $\mathbb{B}^*$ is computable (decidable) in polynomial space if there exists a Turing machine that computes F and runs in space $s(n) = \mathrm{poly}(n)$, where n is the length of the input.

The class of all functions (predicates) computable (decidable) in polynomial space is called PSPACE.

Note that any machine that runs in polynomial time also runs in polynomial space, therefore $\mathrm{P} \subseteq \mathrm{PSPACE}$. Most experts believe that this inclusion is strict, i.e., $\mathrm{P} \ne \mathrm{PSPACE}$, although nobody has succeeded in proving it so far. This is a famous open problem.

Problems

[1] **1.1.** Construct a Turing machine that reverses its input (e.g., produces "0010111" from "1110100").

[1] **1.2.** Construct a Turing machine that adds two numbers written in binary. (Assume that the numbers are separated by a special symbol "+" that belongs to the external alphabet of the TM.)

[3!] **1.3** ("The halting problem is undecidable"). Prove that there is no algorithm that determines, for given Turing machine and input string, whether the machine terminates at that input or not.

[2] **1.4.** Prove that there is no algorithm that enumerates all Turing machines that do not halt when started with the empty tape.

(Informally, enumeration is a process which produces one element of a set after another so that every element is included in this list. Exact definition: a set $X \subseteq A^*$ is called *enumerable* if it is the set of all possible outputs of some Turing machine E.)

[3] **1.5.** Let $T(n)$ be the maximum number of steps performed by a Turing machine with $\leq n$ states and $\leq n$ symbols before it terminates starting with the empty tape. Prove that the function $T(n)$ grows faster than any computable total function $b(n)$, i.e., $\lim_{n\to\infty} T(n)/b(n) = \infty$.

The mode of operation of Turing machines is rather limited and can be extended in different ways. For example, one can consider *multitape* Turing machines that have a finite number of tapes. Each tape has its own head that can read and write symbols on the tape. There are two special tapes: an *input* read-only tape, and an *output* write-only tape (after writing a symbol the head moves to the right). A k-tape machine has an input tape, an output tape, and k work tapes.

At each step the machine reads symbols on all of the tapes (except for the output tape), and its action depends upon these symbols and the current state. This action is determined by a transition function that says what the next state is, what symbols should be written on each work tape, what movement is prescribed for each head (except for the output one), and what symbol should be written on the output tape (it is possible also that no symbol is written; in this case the output head does not move).

Initially all work tapes are empty, and the input string is written on the input tape. The output is the content of the output tape after the TM halts (this happens when the transition function is undefined or when one of the heads moves past the left end of its tape).

More tapes allow Turing machine to work faster; however, the difference is not so great, as the following problems show.

[2] **1.6.** Prove that a 2-tape Turing machine working in time $T(n)$ for inputs of length n can be simulated by an ordinary Turing machine working in time $O(T^2(n))$.

[3] **1.7.** Prove that a 3-tape Turing machine working in time $T(n)$ for inputs of length n can be simulated by a 2-tape machine working in time $O\big(T(n)\log T(n)\big)$.

[3] **1.8.** Let M be a (single-tape) Turing machine that duplicates the input string (e.g., produces "blabla" from "bla"). Let $T(n)$ be its maximum running time when processing input strings of length n. Prove that $T(n) \geq \varepsilon n^2$ for some ε and for all n. What can be said about $T'(n)$, the minimum running time for inputs of length n?

[2] **1.9.** Consider a programming language that includes 100 integer variables, the constant 0, increment and decrement statements, and conditions of type "variable = 0". One may use `if-then-else` and `while` constructs, but recursion is not allowed. Prove that any computable function of type $\mathbb{Z} \to \mathbb{Z}$ has a program in this language.

2. Boolean circuits

2.1. Definitions. Complete bases. A *Boolean circuit* is a representation of a given Boolean function as a composition of other Boolean functions.

By a *Boolean function* of n variables we mean a function of type $\mathbb{B}^n \to \mathbb{B}$. (For $n = 0$ we get two constants 0 and 1.) Assume that some set $\mathcal{A}$ of Boolean functions (*basis*) is fixed. It may contain functions with different arity (number of arguments).

A circuit C over $\mathcal{A}$ is a sequence of assignments. These assignments involve n *input* variables $x_1, \ldots, x_n$ and several *auxiliary* variables $y_1, \ldots, y_m$; the j-th assignment has the form $y_j := f_j(u_1, \ldots, u_r)$. Here f_j is some function from $\mathcal{A}$, and each of the variables $u_1, \ldots, u_r$ is either an input variable or an auxiliary variable that precedes y_j. The latter requirement guarantees that the right-hand side of the assignment is defined when we perform it (we assume that the values of the input variables are defined at the beginning; then we start defining y_1, y_2, etc.).

The value of the last auxiliary variable is the *result* of computation. A circuit with n input variables $x_1, \ldots, x_n$ computes a Boolean function $F : \mathbb{B}^n \to \mathbb{B}$ if the result of computation is equal to $F(x_1, \ldots, x_n)$ for any values of $x_1, \ldots, x_n$.

If we select m auxiliary variables (instead of one) to be the output, we get the definition of the computation of a function $F : \mathbb{B}^n \to \mathbb{B}^m$ by a circuit.

A circuit can also be represented by an acyclic directed graph (as in Figure 2.1), in which vertices of in-degree 0 (*inputs* — we put them on top of the figure) are labeled by input variables; all other vertices (*gates*) are labeled with functions from $\mathcal{A}$ in such a way that the in-degree of a vertex matches the arity of the function placed at that vertex, and each incoming

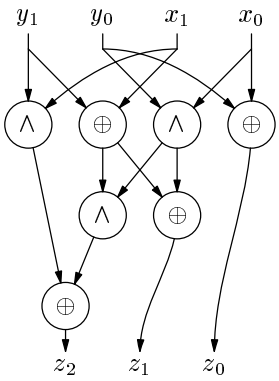


Fig. 2.1. Circuit over the basis $\{\wedge, \oplus\}$ for the addition of two 2-digit numbers: $\overline{z_2 z_1 z_0} = \overline{x_1 x_0} + \overline{y_1 y_0}$.

edge is linked to one of the function arguments. Some vertices are called *output* vertices. Since the graph is acyclic, any value assignment to the input variables can be extended (uniquely) to a consistent set of values for all vertices. Therefore, the set of values at the output vertices is a function of input values. This function is computed by a circuit. It is easy to see that this representation of a circuit can be transformed into a sequence of assignments, and vice versa. (We will not use this representation much, but it explains the name "circuit".)

A circuit for a Boolean function is called a *formula* if each auxiliary variable, except the last one, is used (i.e., appears on the right-hand side of an assignment) exactly once. The graph of a formula is a tree whose leaves are labeled by input variables; each label may appear any number of times. (In a general circuit an auxiliary variable may be used more than once, in which case the out-degree of the corresponding vertex is more than 1.)

Why the name "formula"? If each auxiliary variable is used only once, we can replace it by its definition. Performing all these "inline substitutions", we get an expression for f that contains only input variables, functions from the basis, and parentheses. The size of this expression approximately equals the total length of all assignments. (It is important that each auxiliary variable is used only once; otherwise we would need to replace all occurrences of each auxiliary variable by their definitions, and the size might increase exponentially.)

A basis $\mathcal{A}$ is called *complete* if, for any Boolean function f, there is a circuit over $\mathcal{A}$ that computes f. (It is easy to see that in this case any function of type $\mathbb{B}^n \to \mathbb{B}^m$ can be computed by an appropriate circuit.)

The most common basis contains the following three functions:

$$NOT(x) = \neg x,\ \ OR(x_1, x_2) = x_1 \vee x_2,\ \ AND(x_1, x_2) = x_1 \wedge x_2$$

(*negation*, *disjunction*, *conjunction*). Here are the value tables for these functions:

x	$\neg x$
0	1
1	0

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Theorem 2.1. *The basis* $\{NOT, OR, AND\} = \{\neg, \vee, \wedge\}$ *is complete.*

Proof. Any Boolean function of n arguments is determined by its value table, which contains 2^n rows. Each row contains the values of the arguments and the corresponding value of the function.

If the function takes value 1 only once, it can be computed by a conjunction of *literals*; each literal is either a variable or the negation of a variable. For example, if $f(x_1, x_2, x_3)$ is true (equals 1) only for $x_1 = 1, x_2 = 0, x_3 = 1$, then

$$f(x_1, x_2, x_3) = x_1 \wedge \neg x_2 \wedge x_3$$

(the conjunction is associative, so we omit parentheses; the order of literals is also unimportant).

In the general case, a function f can be represented in the form

$$f(x) = \bigvee_{\{u:\, f(u)=1\}} \chi_u(x), \tag{2.1}$$

where $u = (u_1, \ldots, u_n)$, and χ_u is the function such that $\chi_u(x) = 1$ if $x = u$, and $\chi_u(x) = 0$ otherwise. □

A representation of type (2.1) is called a *disjunctive normal form* (DNF). By definition, a DNF is a disjunction of conjunctions of literals. Later we will also need the conjunctive normal form (CNF) — a conjunction of disjunctions of literals. Any Boolean function can be represented by a CNF. This fact is dual to Theorem 2.1 and can be proved in a dual way (we start with functions that have only one zero in the table). Or we can represent $\neg f$ by a DNF and then get a CNF for f by negation using De Morgan's identities

$$x \wedge y = \neg(\neg x \vee \neg y), \qquad x \vee y = \neg(\neg x \wedge \neg y).$$

These identities show that the basis $\{\neg, \vee, \wedge\}$ is redundant: the subsets $\{\neg, \vee\}$ and $\{\neg, \wedge\}$ also constitute complete bases. Another useful example of a complete basis is $\{\wedge, \oplus\}$.

The number of assignments in a circuit is called its *size*. The minimal size of a circuit over $\mathcal{A}$ that computes a given function f is called the *circuit*

complexity of f (with respect to the basis $\mathcal{A}$) and is denoted by $c_{\mathcal{A}}(f)$. The value of $c_{\mathcal{A}}(f)$ depends on $\mathcal{A}$, but the transition from one finite complete basis to another changes the circuit complexity by at most a constant factor: if $\mathcal{A}_1$ and $\mathcal{A}_2$ are two finite complete bases, then $c_{\mathcal{A}_1}(f) = O(c_{\mathcal{A}_2}(f))$ and vice versa. Indeed, each $\mathcal{A}_2$-assignment can be replaced by $O(1)$ $\mathcal{A}_1$-assignments since $\mathcal{A}_1$ is a complete basis.

We are interested in asymptotic estimates for circuit complexity (up to an $O(1)$-factor); therefore the particular choice of a complete basis is not important. We use the notation $c(f)$ for the circuit complexity of f with respect to some finite complete basis.

[2] **Problem 2.1.** Construct an algorithm that determines whether a given set of Boolean functions $\mathcal{A}$ constitutes a complete basis. (Functions are represented by tables.)

[2!] **Problem 2.2.** Let c_n be the maximum complexity $c(f)$ for Boolean functions f in n variables. Prove that $1.99^n < c_n < 2.01^n$ for sufficiently large n.

2.2. Circuits versus Turing machines. Any predicate F on $\mathbb{B}^*$ can be restricted to strings of fixed length n, giving rise to the Boolean function

$$F_n(x_1, \ldots, x_n) = F(x_1 x_2 \cdots x_n).$$

Thus F may be regarded as the sequence of Boolean functions $F_0, F_1, F_2, \ldots$.

Similarly, in most cases of practical importance a (partial) function of type $F : \mathbb{B}^* \to \mathbb{B}^*$ can be represented by a sequence of (partial) functions $F_n : \mathbb{B}^n \to \mathbb{B}^{p(n)}$, where $p(n)$ is a polynomial with integer coefficients. We will focus on predicates for simplicity, though.

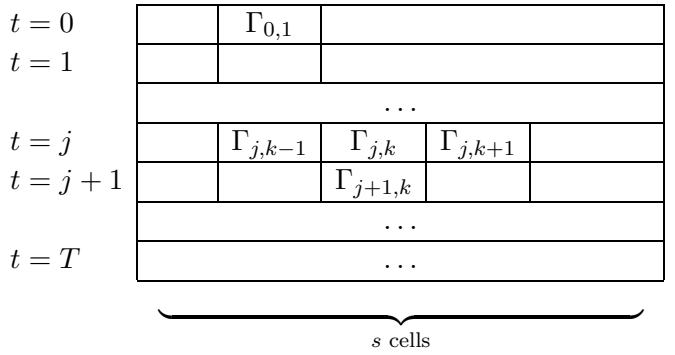
Definition 2.1. A predicate F belongs to the class P/poly ("nonuniform P") if

$$c(F_n) = \mathrm{poly}(n).$$

(The term "nonuniform" indicates that a separate procedure, i.e., a Boolean circuit, is used to perform computation with input strings of each individual length.)

Theorem 2.2. $\mathrm{P} \subset \mathrm{P/poly}$.

Proof. Let F be a predicate decidable in polynomial time. We have to prove that $F \in \mathrm{P/poly}$. Let M be a TM that computes F and runs in polynomial time (and therefore in polynomial space). The computation by M on some input x of length n can be represented as a space-time diagram Γ that is a rectangular table of size $T \times s$, where $T = \mathrm{poly}(n)$ and $s = \mathrm{poly}(n)$.



In this table the j-th row represents the configuration of M after j steps: $\Gamma_{j,k}$ corresponds to cell k at time j and consists of two parts: the symbol on the tape and the state of the TM if its head is in k-th cell (or a special symbol Λ if it is not). In other words, all $\Gamma_{j,k}$ belong to $S \times \big(\{\Lambda\} \cup Q\big)$. (Only one entry in a row can have the second component different from Λ.) For simplicity we assume that after the computation stops all subsequent rows in the table repeat the last row of the computation.

There are local rules that determine the contents of a cell $\Gamma_{j+1,k}$ if we know the contents of three neighboring cells in row j, i.e., $\Gamma_{j,k-1}$, $\Gamma_{j,k}$ and $\Gamma_{j,k+1}$. Indeed, the head speed is at most one cell per step, so no other cell can influence $\Gamma_{j+1,k}$. Rules for boundary cells are somewhat special; they take into account that the head cannot be located outside the table.

Now we construct a circuit that computes $F(x)$ for inputs x of length n. The contents of each table cell can be encoded by a constant (i.e., independent of n) number of Boolean variables. These variables (for all cells) will be the auxiliary variables of the circuit.

Each variable encoding the cell $\Gamma_{j+1,k}$ depends only on the variables that encode $\Gamma_{j,k-1}$, $\Gamma_{j,k}$, and $\Gamma_{j,k+1}$. This dependence is a Boolean function with a constant number of arguments. These functions can be computed by circuits of size $O(1)$. Combining these circuits, we obtain a circuit that computes all of the variables which encode the state of every cell. The size of this circuit is $O(sT)O(1) = \mathrm{poly}(n)$.

It remains to note that the variables in row 0 are determined by the input string, and this dependence leads to additional $\mathrm{poly}(n)$ assignments. Similarly, to find out the result of M it is enough to look at the symbol written in the 0-th cell of the tape at the end of the computation. So the output is a function of $\Gamma_{T,0}$ and can be computed by an additional $O(1)$-size circuit. Finally we get a $\mathrm{poly}(n)$-size circuit that simulates the behavior of M for inputs of length n and therefore computes the Boolean function F_n. □

Remark 2.1. The class P/poly is bigger than P. Indeed, let $\varphi : \mathbb{N} \to \mathbb{B}$ be an arbitrary function (maybe even a noncomputable one). Consider the predicate F_φ such that $F_\varphi(x) = \varphi(|x|)$, where $|x|$ stands for the length of string x. The restriction of F_φ to strings of length n is a constant function (0 or 1), so the circuit complexity of $(F_\varphi)_n$ is $O(1)$. Therefore F_φ for any φ belongs to P/poly, although for a noncomputable φ the predicate F_φ is not computable and thus does not belong to P.

Remark 2.2. That said, P/poly seems to be a good approximation of P for many purposes. Indeed, the class P/poly is relatively small: out of 2^{2^n} Boolean functions in n variables only $2^{\mathrm{poly}(n)}$ functions have polynomial circuit complexity (see solution to Problem 2.2). The difference between uniform and nonuniform computation is more important for bigger classes. For example, EXPTIME, the class of predicates decidable in time $2^{\mathrm{poly}(n)}$, is a nontrivial computational class. However, the nonuniform analog of this class includes all predicates!

The arguments used to prove Theorem 2.2 can also be used to prove the following criterion:

Theorem 2.3. *F belongs to* P *if and only if these conditions hold:*

(1) $F \in$ P/poly;

(2) *the functions F_n are computed by polynomial-size circuits C_n with the following property: there exists a TM that for each positive integer n runs in time* poly(n) *and constructs the circuit C_n.*

A sequence of circuits C_n with this property is called *polynomial-time uniform.*

Note that the TM mentioned in (2) is *not* running in polynomial time since its running time is polynomial in n but not in $\log n$ (the number of bits in the binary representation of n). Note also that we implicitly use some natural encoding for circuits when saying "TM constructs a circuit".

Proof. $\Rightarrow$ The circuit for computing F_n constructed in Theorem 1.2 has regular structure, and it is clear that the corresponding sequence of assignments can be produced in polynomial time when n is known.

$\Leftarrow$ This is also simple. We compute the size of the input string x, then apply the TM to construct a circuit $C_{|x|}$ that computes $F_{|x|}$. Then we perform the assignments indicated in $C_{|x|}$, using x as the input, and get $F(x)$. All these computations can be performed in polynomial (in $|x|$) time. □

[1] **Problem 2.3.** Prove that there exists a decidable predicate that belongs to P/poly but not to P.

2.3. Basic algorithms. Depth, space and width. We challenge the reader to study these topics by working on problems. (Solutions are also provided, of course.) In Problems 2.9–2.16 we introduce some basic algorithms which are used universally throughout this book. The algorithms are described in terms of uniform (i.e., effectively constructed) sequences of circuits. In this book we will be satisfied with *polynomial-time uniformity*; cf. Theorem 2.3. [This property is intuitive and usually easy to check. Another useful notion is *logarithmic-space uniformity*: the circuits should be constructed by a Turing machine with work space $O(\log n)$ (machines with limited work space are defined below; see 2.3.3). Most of the circuits we build satisfy this stronger condition, although the proof might not be so easy.]

2.3.1. *Depth.* In practice (e.g., when implementing circuits in hardware), size is not the only circuit parameter that counts. Another important parameter is *depth*. Roughly, it is the time that is needed to carry out all assignments in the circuit, if we can do more than one *in parallel*. Interestingly enough, it is also related to the space needed to perform the computation (see Problems 2.17 and 2.18). In general, there is a trade-off between size and depth. In our solutions we will be willing to increase the size of a circuit a little to gain a considerable reduction of the depth (see e.g., Problem 2.14). As a result, we come up with circuits of polynomial size and poly-logarithmic depth. (With a certain notion of uniformity, the functions computed by such circuits form the so-called class NC, an interesting subclass of P.)

More formally, the *depth* of a Boolean circuit is the maximum number of gates on any path from the input to the output. The depth is $\leq d$ if and only if one can arrange the gates into d layers, so that the input bits of any gate at layer j come from the layers $1, \ldots, j-1$. For example, the circuit in Figure 2.1 has depth 3.

Unless stated explicitly otherwise, we assume that all gates have bounded *fan-in*, i.e., the number of input bits. (This is always the case when circuits are built over a finite basis. Unbounded fan-in can occur, for example, if one uses *OR* gates with an arbitrary number of inputs.) We also assume that the *fan-out* (the number of times an input or an auxiliary variable is used) is bounded.[1] If it is necessary to use the variable more times, one may insert additional "trivial gates" (identity transformations) into the circuit, at the cost of some increase in size and depth. Note that a formula is a circuit in which all auxiliary variables have fan-out 1, whereas the fan-out of the input variables is unbounded.

[1] This restriction is needed to allow conversion of Boolean circuits into quantum circuits without extra cost (see Section 7). However, it is in no way standard: in most studies in computational complexity, unbounded fan-out is assumed.

[1] **2.4.** Let C be an $O(\log n)$-depth circuit which computes some function $f : \mathbb{B}^n \to \mathbb{B}^m$. Prove that after eliminating all extra variables and gates in C (those which are not connected to the output), we get a circuit of size $\mathrm{poly}(n+m)$.

[1!] **2.5.** Let C be a circuit (over some basis B) which computes a function $f : \mathbb{B}^n \to \mathbb{B}$. Prove that C can be converted into an equivalent (i.e., computing the same function) formula C' of the same depth over the same basis. (It follows from the solution that the size of C' does not exceed c^d, where d is the depth and c is the maximal fan-in.)

[3] **2.6** ("Balanced formula"). Let C be a formula of size L over the basis $\{NOT, OR, AND\}$ (with fan-in ≤ 2). Prove that it can be converted into an equivalent formula of depth $O(\log L)$ over the same basis.

(Therefore, it does not matter whether we define the formula complexity of a function in terms of size or in terms of depth. This is not true for the circuit complexity.)

[1] **2.7.** Show that any function can be computed by a circuit of depth ≤ 3 with gates of type *NOT*, *AND*, and *OR*, if we allow *AND*- and *OR*-gates with arbitrary fan-in and fan-out.

[2] **2.8.** By definition, the function PARITY is the sum of n bits modulo 2. Suppose it is computed by a circuit of depth 3 containing *NOT*-gates, *AND*-gates and *OR*-gates with arbitrary fan-in. Show that the size of the circuit is exponential (at least c^n for some $c > 1$ and for all n).

2.3.2. *Basic algorithms.*

[3!] **2.9.** Comparison. Construct a circuit of size $O(n)$ and depth $O(\log n)$ that tells whether two n-bit integers are equal and if they are not, which one is greater.

[2] **2.10.** Let $n = 2^l$. Construct circuits of size $O(n)$ and depth $O(\log n)$ for the solution of the following problems.

a) Access by index. Given an n-bit string $x = x_0 \cdots x_{n-1}$ (a table) and an l-bit number j (an index), find x_j.

b) Search. Evaluate the disjunction $y = x_0 \vee \cdots \vee x_{n-1}$, and if it equals 1, find the smallest j such that $x_j = 1$.

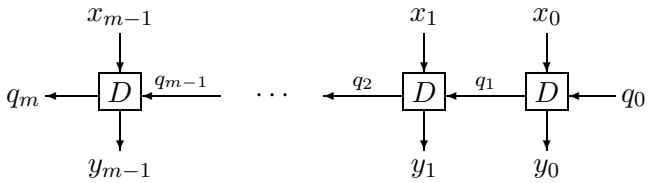
We now describe one rather general method of parallelizing computation. A *finite-state automaton* is a device with an input alphabet A', an output alphabet A'', a set of states Q and a transition function $D : Q \times A' \to Q \times A''$ (recall that such a device is a part of a Turing machine). It is initially set to some state $q_0 \in Q$. Then it receives input symbols $x_0, \ldots, x_{m-1} \in A'$ and

changes its state from q_0 to q_1 to q_2, etc., according to the rule[2]

$$(q_{j+1}, y_j) = D(q_j, x_j).$$

The iterated application of D defines a function $D^m : (q_0, x_0, \dots, x_{m-1}) \mapsto (q_m, y_0, \dots, y_{m-1})$. We may assume without loss of generality that $Q = \mathbb{B}^r$, $A' = \mathbb{B}^{r'}$, $A'' = \mathbb{B}^{r''}$; then $D : \mathbb{B}^{r+r'} \to \mathbb{B}^{r+r''}$ whereas $D^m : \mathbb{B}^{r+r'm} \to \mathbb{B}^{r+r''m}$.

The work of the automaton can be represented by this diagram:



[3!] **2.11** ("Parallelization of iteration"). Let the integers r, r', r'' and m be fixed; set $k = r + r' + r''$. Construct a circuit of size $\exp(O(k))\, m$ and depth $O(k \log m)$ that receives a transition function $D : \mathbb{B}^{r+r'} \to \mathbb{B}^{r+r''}$ (as a value table), an initial state q_0 and input symbols $x_0, \dots, x_{m-1}$, and produces the output $(q_m, y_0, \dots, y_{m-1}) = D^m(q_0, x_0, \dots, x_{m-1})$.

[2!] **2.12.** Addition. Construct a circuit of size $O(n)$ and depth $O(\log n)$ that adds two n-bit integers.

[3!] **2.13.** The following two problems are closely related.

a) Iterated addition. Construct a circuit of size $O(nm)$ and depth $O(\log n + \log m)$ for the addition of m n-digit numbers.

b) Multiplication. Construct a circuit with the same parameters for the multiplication of an n-digit number by an m-digit number.

[3!] **2.14.** Division. This problem also comes in two variants.

a) Compute the inverse of a real number $x = \overline{1.x_1x_2\cdots} = 1 + \sum_{j=1}^{\infty} 2^{-j}x_j$ with precision 2^{-n}. By definition, this requires to find a number z such that $|z - x^{-1}| \le 2^{-n}$. For this to be possible, x must be known with precision 2^{-n} or better; let us assume that x is represented by an $n+1$-digit number x' such that $|x' - x| \le 2^{-(n+1)}$. Construct an $O(n^2 \log n)$-size, $O((\log n)^2)$-depth circuit for the solution of this problem.

[2]Our definition of an automaton is not standard. We require that the automaton reads *and* outputs one symbol at each step. Traditionally, an automaton is allowed to either read a symbol, or output a symbol, or both, depending on the current state. The operation of such a general automaton can be represented as the application of an automaton in our sense (with a suitable output alphabet B) followed by substitution of a word for each output symbol.

b) Divide two integers with remainder: $(a, b) \mapsto \big(\lfloor a/b \rfloor, (a \bmod b)\big)$, where $0 \le a < 2^k b$ and $0 < b < 2^n$. In this case, construct a circuit of size $O(nk + k^2 \log k)$ and depth $O(\log n + (\log k)^2)$.

[2!] **2.15.** MAJORITY. The majority function $\mathrm{MAJ} : \mathbb{B}^n \to \mathbb{B}$ equals 1 for strings in which the number of 1s is greater than the number of 0s, and equals 0 elsewhere. Construct a circuit of size $O(n)$ and depth $O(\log n)$ that computes the majority function.

[3!] **2.16.** CONNECTING PATH. Construct a circuit of size $O(n^3 \log n)$ and depth $O((\log n)^2)$ that checks whether two fixed vertices of an undirected graph are connected by a path. The graph has n vertices, labeled $1, \ldots, n$; there are $m = n(n-1)/2$ input variables x_{ij} (where $i < j$) indicating whether there is an edge between i and j.

2.3.3. ***Space and width.*** In the solution to the above problems we strove to provide parallel algorithms, which were described by circuits of poly-logarithmic depth. We now show that, if the circuit size is not taken into account, then uniform computation with poly-logarithmic depth[3] is equivalent to computation with poly-logarithmic space.

We are going to study computation with very limited space — so small that the input string would not fit into it. So, let us assume that the input string $x = x_0 \cdots x_{n-1}$ is stored in a supplementary read-only memory, and the Turing machine can access bits of x by their numbers. We may think of the input string as a function $X : j \mapsto x_j$ computed by some external agent, called "oracle". The length of an "oracle query" j is included into the machine work space, but the length of x is not. This way we can access all input bits with space $O(\log n)$ (but no smaller).

Definition 2.2. Let $X : A^* \to A^*$ be a partial function. A *Turing machine with oracle* X is an ordinary TM with a supplementary tape, in which it can write a string z and have the value of $X(z)$ available for inspection at the next computation step.

In our case $X(z) = x_j$, where z is the binary representation of j $(0 \le j \le n-1)$ by $\lceil \log_2 n \rceil$ digits; otherwise $X(z)$ is undefined.

The computation of a function $f : \mathbb{B}^n \to \mathbb{B}^m$ by such a machine is defined as follows. Let x $(|x| = n)$ be the input. We write n and another number k $(0 \le k < m)$ on the machine tape and run the machine. We allow it to query bits of x. When the computation is complete, the first cell of the tape must contain the k-th bit of $f(x)$. Note that if the work space is limited by s, then $n \le 2^s$ and $m \le 2^s$. The computation time is also bounded:

[3] As is usual, we consider circuits over a finite complete basis; the fan-in and fan-out are bounded.

the machine either stops within $\exp(O(s))$ steps, or enters a cycle and runs forever.

[2!] **2.17** ("Small depth circuits can be simulated in small space"). Prove that there exists a Turing machine that evaluates the output variables of a circuit of depth d over a fixed basis, using work space $O(d + \log m)$ (where m is the number of the output variables). The input to the machine consists of a description of the circuit and the values of its input variables.

[3!] **2.18** ("Computation with small space is parallelizable"). Let M be a Turing machine. For each choice of n, m, s, let $f_{n,m,s} : \mathbb{B}^n \to \mathbb{B}^m$ be the function computed by the machine M with space s (it may be a partial function). Prove that there exists a family of $\exp(O(s))$-size, $O(s^2)$-depth circuits $C_{n,m,s}$ which compute the functions $f_{n,m,s}$.

(These circuits can be constructed by a TM with space $O(s)$, but we will not prove that.)

The reader might wonder why we discuss the space restriction in terms of Turing machines while the circuit language is apparently more convenient. So, let us ask this question: what is the circuit analog of the computation space? The obvious answer is that it is the circuit width.

Let C be a circuit whose gates are arranged into d layers. The *width* of C is the maximum amount of information carried between the layers, *not including the input variables*. More formally, for each $l = 1, \ldots, d$ we define w_l to be the number of auxiliary variables from layers $1, \ldots, l$ that are output variables or connected to some variables from layers $l+1, \ldots, d$ (i.e., used in the right-hand side of the corresponding assignments). Then the width of C is $w = \max\{w_1, \ldots, w_d\}$.

But here comes a little surprise: any Boolean function can be computed by a circuit of bounded width (see Problem 2.19 below). Therefore the width is rather meaningless parameter, unless we put some other restrictions on the circuit. To characterize computation with limited space (e.g., the class PSPACE), one has to use either Turing machines, or some class of circuits with regular structure.

[3] **2.19** (Barrington [**8**]). Let C be a formula of depth d that computes a Boolean function $f(x_1, \ldots, x_n)$. Construct a circuit of size $\exp(O(d))$ and width $O(1)$ that computes the same function.

3. The class NP: Reducibility and completeness

3.1. Nondeterministic Turing machines. NP is the class of predicates recognizable in polynomial time by "nondeterministic Turing machines." (The word "nondeterministic" is not appropriate but widely used.)

The class NP is defined only for predicates. One says, for example, that "the property that a graph has a Hamiltonian cycle belongs to NP". (A *Hamiltonian cycle* is a cycle that traverses all vertices exactly once.)

We give several definitions of this class. The first uses *nondeterministic Turing machines*. A nondeterministic Turing machine (NTM) resembles an ordinary (deterministic) machine, but can nondeterministically choose one of several actions possible in a given configuration. More formally, a transition function of an NTM is multivalued: for each pair (state, symbol) there is a set of possible actions. Each action has a form (new state, new symbol, shift). If the set of possible actions has cardinality at most 1 for each state-symbol combination, we get an ordinary Turing machine.

A *computational path* of an NTM is determined by a choice of one of the possible transitions at each step; different paths are possible for the same input.

Definition 3.1. A predicate L belongs to the class NP if there exists an NTM M and a polynomial $p(n)$ such that

$L(x) = 1 \quad \Rightarrow$ there exists a computational path that gives answer "yes" in time not exceeding $p(|x|)$;

$L(x) = 0 \quad \Rightarrow$ (version 1) there is no path with this property;
(version 2) ... and, moreover, there is no path (of any length) that gives answer "yes".

Remark 3.1. Versions 1 and 2 are equivalent. Indeed, let an NTM M_1 satisfy version 1 of the definition. To exclude "yes" answers for long computational paths, it suffices to simulate M_1 while counting its steps, and to abort the computation after $p(|x|)$ steps.

Remark 3.2. The argument in Remark 3.1 has a subtle error. If the coefficients of the polynomial p are noncomputable, difficulties may arise when we have to compare the number of steps with the value of the polynomial. In order to avoid this complication we will add to Definition 3.1 an additional requirement: $p(n)$ has integer coefficients.

Remark 3.3. By definition $\mathrm{P} \subseteq \mathrm{NP}$. Is this inclusion strict? Rather intense although unsuccessful attempts have been made over the past 30 years to prove the strictness. Recently S. Smale included the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ problem in the list of most important mathematical problems for the new century (the other problems are the Riemann hypothesis and the Poincaré conjecture). More practical people can dream of \$1,000,000 that Clay Institute offers for the solution of this problem.

Now we give another definition of the class NP, which looks more natural. It uses the notion of a polynomially decidable predicate of two variables:

a predicate $R(x, y)$ (where x and y are strings) is *polynomially decidable* (decidable in polynomial time) if there is a (deterministic) TM that computes it in time $\mathrm{poly}(|x|, |y|)$ (which means $\mathrm{poly}(|x|+|y|)$ or $\mathrm{poly}\big(\max\{|x|, |y|\}\big)$ — these two expressions are equivalent).

Definition 3.2. A predicate L belongs to the class NP if it can be represented as

$$L(x) \;=\; \exists y \,\Big(\big(|y| < q(|x|)\big) \wedge R(x, y)\Big),$$

where q is a polynomial (with integer coefficients), and R is a predicate of two variables decidable in polynomial time.

Remark 3.4. Let $R(x, y) =$ "y is a Hamiltonian cycle in the graph x". More precisely, we should say: "x is a binary encoding of some graph, and y is an encoding of a Hamiltonian cycle in that graph". Take $q(n) = n$. Then $L(x)$ means that graph x has a Hamiltonian cycle. (We assume that the encoding of any cycle in a graph is shorter than the encoding of the graph itself.)

Theorem 3.1. *Definitions* 3.1 *and* 3.2 *are equivalent.*

Proof. *Definition* 3.1 $\Rightarrow$ *Definition* 3.2. Let M be an NTM and let $p(n)$ be the polynomial of the first definition. Consider the predicate $R(x, y) =$ "y is a description of a computational path that starts with input x, ends with answer 'yes', and takes at most $p(|x|)$ steps". Such a description has length proportional to the computation time if an appropriate encoding is used (and even if we use a table as in the proof of Theorem 2.2, the description length is at most quadratic). Therefore for $q(n)$ in the second definition we can take $O(p(n))$ (or $O(p^2(n))$ if we use less efficient encoding).

It remains to prove that predicate R belongs to P. This is almost obvious. We must check that we are presented with a valid description of some computational path (this is a polynomial task), and that this path starts with x, takes at most $p(|x|)$ steps, and ends with "yes" (that is also easy).

Definition 3.2 $\Rightarrow$ *Definition* 3.1. Let R, q be as in Definition 3.2. We construct an NTM M for Definition 3.1. M works in two stages.

First, M nondeterministically guesses y. More precisely, this means that M goes to the end of the input string, moves one cell to the right, writes #, moves once more to the right, and then writes some string y (M's nondeterministic rules allow it to write any symbol and then move to the right, or finish writing). After that, the tape has the form $x\#y$ for some y, and M goes to the second stage.

At the second stage M checks that $|y| < q(|x|)$ (note that M can write a very long y for any given x) and computes $R(x, y)$ (using the polynomial algorithm that exists according to Definition 3.2). If $x \in L$, then there

is some y such that $|y| < q(|x|)$ and $R(x, y)$. Therefore M has, for x, a computational path of polynomial length ending with "yes". If $x \notin L$, no computational path ends with "yes". □

Now we proceed to yet another description of NP that is just a reformulation of Definition 3.2, but which has a form that can be used to define other complexity classes.

Definition 3.3. Imagine two persons: King **A**rthur, whose mental abilities are polynomially bounded, and a wizard **M**erlin, who is intellectually omnipotent and knows everything. **A** is interested in some property $L(x)$ (he wants, for example, to be sure that some graph x has a Hamiltonian cycle). **M** wants to convince **A** that $L(x)$ is true. But **A** does not trust **M** ("he is too clever to be loyal") and wants to make sure $L(x)$ is true, not just believe **M**.

So Arthur arranges that, after both he and Merlin see input string x, **M** writes a note to **A** where he "proves" that $L(x)$ is true. Then **A** verifies this proof by some polynomial proof-checking procedure.

The proof-checking procedure is a polynomial predicate

$$R(x, y) = \text{"}y \text{ is a proof of } L(x)\text{"}.$$

It should satisfy two properties:

$L(x) = 1 \quad \Rightarrow$ **M** can convince **A** that $L(x)$ is true by presenting some proof y such that $R(x, y)$;

$L(x) = 0 \quad \Rightarrow$ whatever **M** says, **A** is not convinced: $R(x, y)$ is false for any y.

Moreover, the proof y should have polynomial (in $|x|$) length, otherwise **A** cannot check $R(x, y)$ in polynomial (in $|x|$) time.

In this way, we arrive precisely at Definition 3.2.

3.2. Reducibility and NP-completeness. The notion of reducibility allows us to verify that a predicate is at least as difficult as some other predicate.

Definition 3.4 (*Karp reducibility*). A predicate L_1 is *reducible* to a predicate L_2 if there exists a function $f \in \mathrm{P}$ such that $L_1(x) = L_2(f(x))$ for any input string x.

We say that f *reduces* L_1 to L_2. Notation: $L_1 \propto L_2$.

Karp reducibility is also called "polynomial reducibility" (or just "reducibility").

Lemma 3.2. *Let $L_1 \propto L_2$. Then*

(a) $L_2 \in \mathrm{P} \Rightarrow L_1 \in \mathrm{P}$;

(b) $L_1 \notin \mathrm{P} \Rightarrow L_2 \notin \mathrm{P}$;

(c) $L_2 \in \mathrm{NP} \Rightarrow L_1 \in \mathrm{NP}$.

Proof. To prove (a) let us note that $|f(x)| = \mathrm{poly}(|x|)$ for any $f \in \mathrm{P}$. Therefore, we can decide $L_1(x)$ in polynomial time as follows: we compute $f(x)$ and then compute $L_2(f(x))$.

Part (b) follows from (a).

Part (c) is also simple. It can be explained in various ways. Using the Arthur–Merlin metaphor, we say that Merlin communicates to Arthur a proof that $L_2(f(x))$ is true (if it is true). Then Arthur computes $f(x)$ by himself and checks whether $L_2(f(x))$ is true, using Merlin's proof.

Using Definition 3.2, we can explain the same thing as follows:

$$L_1(x) \Leftrightarrow L_2(f(x)) \Leftrightarrow \exists\, y \Big(\big(|y| < q(|f(x)|)\big) \wedge R(f(x), y) \Big)$$
$$\Leftrightarrow \exists\, y \Big(\big(|y| < r(|x|)\big) \wedge R'(x, y) \Big).$$

Here $R'(x, y)$ stands for $\big(|y| < q(|f(x)|)\big) \wedge R(f(x), y)$, and $r(n) = q(h(n))$, where $h(n)$ is a polynomial bound for the time needed to compute $f(x)$ for any string x of length n (and, therefore, $|f(x)| \le h(|x|)$ for any x). □

Definition 3.5. A predicate $L \in \mathrm{NP}$ is NP-*complete* if any predicate in NP is reducible to it.

If some NP-complete predicate can be computed in time $T(n)$, then any NP-predicate can be computed in time $\mathrm{poly}(n) + T(\mathrm{poly}(n))$. Therefore, if some NP-complete predicate belongs to P, then P = NP. Put is this way: if P ≠ NP (which is probably true), then no NP-complete predicate belongs to P.

If we measure running time "up to a polynomial", then we can say that NP-complete predicates are the "most difficult" ones in NP.

The key result in computational complexity says that NP-complete predicates do exist. Here is one of them, called *satisfiability*: $SAT(x)$ means that x is a propositional formula (containing Boolean variables and operations $\neg$, $\wedge$, and $\vee$) that is satisfiable, i.e., true for some values of the variables.

Theorem 3.3 (Cook, Levin).

(1) $SAT \in \mathrm{NP}$;

(2) SAT *is* NP-*complete.*

Corollary. *If* $SAT \in \mathrm{P}$, *then* P = NP.

Proof of Theorem 3.3. (1) To convince Arthur that a formula is satisfiable, Merlin needs only to show him the values of the variables that make it true. Then Arthur can compute the value of the whole formula by himself.

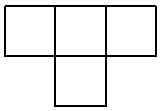
(2) Let $L(x)$ be an NP-predicate and

$$L(x) \;=\; \exists y \left(\big(|y| < q(|x|) \big) \wedge R(x,y) \right)$$

for some polynomial q and some predicate R decidable in polynomial time.

We need to prove that L is reducible to SAT. Let M be a Turing machine that computes R in polynomial time. Consider the computation table (see the proof of Theorem 2.2) for M working on some input $x\#y$. We will use the same variables as in the proof of Theorem 2.2. These variables encode the contents of cells in the computation table.

Now we write a formula that says that values of variables form an encoding of a successful computation (with answer "yes"), starting with the input $x\#y$. To form a valid computation table, values should obey some local rules for each four cells configured as follows:



These local rules can be written as formulas in $4t$ variables (if t variables are needed to encode one cell). We write the conjunction of these formulas and add formulas saying that the first line contains the input string x followed by # and some binary string y, and that the last line contains the answer "yes".

The satisfying assignment for our formula will be an encoding of a successful computation of M on input $x\#y$ (for some binary string y). On the other hand, any successful computation that uses at most S tape cells and requires at most T steps (where $T \times S$ is the size of the computation table that is encoded) can be transformed into a satisfying assignment.

Therefore, if we consider a computational table that is large enough to contain the computation of $R(x,y)$ for any y such that $|y| < q(|x|)$, and write the corresponding formula as explained above, we get a polynomial-size formula that is satisfiable if and only if $L(x)$ is true. Therefore L is reducible to SAT. □

Other examples of NP-complete problems (predicates) can be obtained using the following lemma.

Lemma 3.4. *If $SAT \propto L$ and $L \in$ NP, then L is* NP*-complete. More generally, if L_1 is* NP*-complete, $L_1 \propto L_2$, and $L_2 \in$ NP, then L_2 is* NP*-complete.*

Proof. The reducibility relation is transitive: if $L_l \propto L_2$ and $L_2 \propto L_3$, then $L_1 \propto L_3$. (Indeed, the composition of two functions from P belongs to P).

According to the hypothesis, any NP-problem is reducible to L_1, and L_1 is reducible to L_2. Therefore any NP-problem is reducible to L_2. □

Now let us consider the satisfiability problem restricted to 3-CNF. Recall that a CNF (conjunctive normal form) is a conjunction of *clauses*; each clause is a disjunction of literals; each literal is either a variable or a negation of a variable. If each clause contains at most three literals, we get a 3-CNF.

By 3-SAT we denote the following predicate:

$$\text{3-}SAT(x) = x \text{ is a satisfiable 3-CNF.}$$

Evidently, 3-SAT is reducible to SAT (because any 3-CNF is a formula). The next theorem shows that the reverse is also true: SAT is reducible to 3-SAT. Therefore 3-SAT is NP-complete (by Lemma 3.4).

Theorem 3.5. *$SAT \propto$ 3-SAT.*

Proof. For any propositional formula (and even for any circuit over the standard basis $\{AND, OR, NOT\}$), we construct a 3-CNF that is satisfiable if and only if the given formula is satisfiable (the given circuit produces output 1 for some input).

Let $x_1, \dots, x_n$ be input variables of the circuit, and let $y_1, \dots, y_s$ be auxiliary variables (see the definition of a circuit). Each assignment involves at most three variables (1 on the left-hand side, and 1 or 2 on the right-hand side).

Now we construct a 3-CNF that has variables $x_1, \dots, x_n, y_1, \dots, y_s$ and is true if and only if the values of all y_j are correctly computed (i.e., they coincide with the right-hand sides of the assignments) and the last variable is true. To this end, we replace each assignment by an equivalence (of Boolean expressions) and represent this equivalence as a 3-CNF:

$$\begin{aligned}
\big(y \Leftrightarrow (x_1 \vee x_2)\big) &= (x_1 \vee x_2 \vee \neg y) \wedge (\neg x_1 \vee x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee y) \\
&\quad \wedge (\neg x_1 \vee \neg x_2 \vee y), \\
\big(y \Leftrightarrow (x_1 \wedge x_2)\big) &= (x_1 \vee x_2 \vee \neg y) \wedge (\neg x_1 \vee x_2 \vee \neg y) \wedge (x_1 \vee \neg x_2 \vee \neg y) \\
&\quad \wedge (\neg x_1 \vee \neg x_2 \vee y), \\
\big(y \Leftrightarrow \neg x\big) &= (x \vee y) \wedge (\neg x \vee \neg y).
\end{aligned}$$

Finally, we take the conjunction of all these 3-CNF's and the variable y_s (the latter represents the condition that the output of the circuit is 1).

Let us assume that the resulting 3-CNF is satisfied by some $x_1, \dots, x_n$, $y_1, \dots, y_s$. If we plug the same values of $x_1, \dots, x_n$ into the circuit, then the auxiliary variables will be equal to $y_1, \dots, y_s$, so the circuit output will be $y_s = 1$. Conversely, if the circuit produces 1 for some inputs, then the

3-CNF is satisfied by the same values of $x_1, \ldots, x_n$ and appropriate values of the auxiliary variables.

So our transformation (of a circuit into a 3-CNF) is indeed a reduction of SAT to 3-SAT. □

Here is another simple example of reduction.

ILP (integer linear programming). Given a system of linear inequalities with integer coefficients, is there an integer solution? (In other words, is the system consistent?)

In this problem the input is the coefficient matrix and the vector of the right-hand sides of the inequalities. It is not obvious that $ILP \in$ NP. Indeed, the solution might exist, but Merlin might not be able to communicate it to Arthur because it is not immediately clear that the number of bits needed to represent the solution is polynomial.

However, it is in fact true that, if a system of inequalities with integer coefficients has an integer solution, then it has an integer solution whose binary representation has size bounded by a polynomial in the bit size of the system; see [**55**, vol. 2, §17.1]. Therefore, *ILP* is in NP.

Now we reduce 3-SAT to *ILP*. Assume that a 3-CNF is given. We construct a system of inequalities that has integer solutions if and only if the 3-CNF is satisfiable. For each Boolean variable x_i we consider an integer variable p_i. The negation $\neg x_i$ corresponds to the expression $1 - p_i$. Each clause $X_j \vee X_k \vee X_m$ (where X_* are literals) corresponds to the inequality $P_j + P_k + P_m \geq 1$, where P_j, P_k, P_m are the expressions corresponding to X_j, X_k, X_m. It remains to add the inequalities $0 \leq p_i \leq 1$ for all i, and we get a system whose solutions are satisfying assignments for the given 3-CNF.

Remark 3.5. If we do not require the solution to be integer-valued, we get the standard linear programming problem. Polynomial algorithms for the solution of this problem (due to Khachiyan and Karmarkar) are described, e. g., in [**55**, vol. 1, §§13, 15.1].

An extensive list of NP-complete problems can be found in [**28**]. Usually NP-completeness is proved by some reduction. Here are several examples of NP-complete problems.

3-COLORING. For a given graph G determine whether it admits a 3-coloring. (By a 3-coloring we mean coloring of the vertices with 3 colors such that each edge has endpoints of different colors.)

(It turns out that a similar 2-coloring problem can be solved in polynomial time.)

CLIQUE. For a graph G and an integer k determine whether the graph has a k-clique (a set of k vertices such that every two of its elements are connected by an edge).

Problems

[3] **3.1.** Prove that one can check the satisfiability of a 2-CNF (a conjunction of disjunctions, each containing two literals) in polynomial time.

[2] **3.2.** Prove that the problem of the existence of an Euler cycle in an undirected graph (an Euler cycle is a cycle that traverses each edge exactly once) belongs to P.

[1!] **3.3.** Suppose we have an NP-*oracle* — a magic device that can immediately solve any instance of the *SAT* problem for us. In other words, for any propositional formula the oracle tells whether it is satisfiable or not. Prove that there is a polynomial-time algorithm that finds a satisfying assignment to a given formula by making a polynomial number of queries to the oracle. (A similar statement is true for the Hamiltonian cycle: finding a Hamiltonian cycle in a graph is at most polynomially harder than checking for its existence.)

[3!] **3.4.** There are n boys and n girls. It is known which boys agree to dance with which girls and vice versa. We want to know whether there exists a *perfect matching* (the boys and the girls can dance in pairs so that everybody is satisfied). Prove that this problem belongs not only to NP (which is almost evident), but also to P.

3.5. Construct

[2!] (a) a polynomial reduction of the 3-*SAT* problem to the clique problem;

[3] (b) a polynomial reduction of 3-*SAT* to CLIQUE that conserves the number of solutions (if a 3-CNF F is transformed into a graph H and an integer k, then the number of satisfying assignments for F equals the number of k-cliques in H).

3.6. Construct

[2!] (a) a polynomial reduction of 3-*SAT* to 3-COLORING;

[3] (b) the same as for (a), with the additional requirement that the number of satisfying assignments is one sixth of the number of 3-colorings of the corresponding graph.

[2] **3.7.** A *tile* is a (1×1)-square whose sides are marked with letters. We want to cover an $(n \times n)$-square with n^2 tiles; it is known which letters are allowed at the boundary of the $n \times n$-square and which letters can be adjacent.

The tiling problem requires us to find, for a given set of tile types (containing at most poly(n) types) and for given restrictions, whether or not there exists a tiling of the $(n \times n)$-square.

Prove that the tiling problem is NP-complete.

[1] **3.8.** Prove that the predicate "x is the binary representation of a composite integer" belongs to NP.

[3] **3.9.** Prove that the predicate "x is the binary representation of a prime integer" belongs to NP.

4. Probabilistic algorithms and the class BPP

4.1. Definitions. Amplification of probability. A probabilistic Turing machine (PTM) is somewhat similar to a nondeterministic one; the difference is that choice is produced by coin tossing, not by guessing. More formally, some (state, symbol) combinations have two associated actions, and the choice between them is made probabilistically. Each instance of this choice is controlled by a random bit. We assume that each random bit is 0 or 1 with probability 1/2 and that different random bits are independent.

(In fact we can replace 1/2 by, say, 1/3 and get almost the same definition; the class BPP (see below) remains the same. However, if we replace 1/2 by some noncomputable real p, we get a rather strange notion which is better avoided.)

For a given input string a PTM generates not a unique output string, but a probability distribution on the set of all strings (different values of the random bits lead to different computation outputs, and each possible output has a certain probability).

Definition 4.1. Let ε be a constant such that $0 < \varepsilon < 1/2$. A predicate L belongs to the class BPP if there exist a PTM M and a polynomial $p(n)$ such that the machine M running on input string x always terminates after at most $p(|x|)$ steps, and

$L(x) = 1 \quad \Rightarrow \quad M$ gives the answer "yes" with probability $\geq 1 - \varepsilon$;

$L(x) = 0 \quad \Rightarrow \quad M$ gives the answer "no" with probability $\leq \varepsilon$.

In this definition the admissible error probability ε can be, say, 0.49 or 10^{-10} — the class BPP will remain the same. Why? Assume that the PTM has probability of error at most $\varepsilon < 1/2$. Take k copies of this machine, run them all for the same input (using independent random bits) and let them vote. Formally, what we do is applying the majority function MAJ (see Problem 2.15) to k individual outputs. The "majority opinion" will be

wrong with probability

$$p_{\text{error}} \le \sum_{\substack{S\subseteq\{1,\dots,k\},\\ |S|\le k/2}} (1-\varepsilon)^{|S|}\varepsilon^{k-|S|}$$

$$= \big((1-\varepsilon)\varepsilon\big)^{k/2} \sum_{\substack{S\subseteq\{1,\dots,k\},\\ |S|\le k/2}} \left(\frac{\varepsilon}{1-\varepsilon}\right)^{k/2-|S|} \tag{4.1}$$

$$< \left(\sqrt{(1-\varepsilon)\varepsilon}\right)^k 2^k = \lambda^k, \qquad \text{where } \lambda = 2\sqrt{\varepsilon(1-\varepsilon)} < 1.$$

If k is big enough, the effective error probability will be as small as we wish. This is called *amplification of probability*. To make the quantity p_{error} smaller than a given number ε', we need to set $k = \Theta(\log(1/\varepsilon'))$. (Since ε and ε' are constants, k does not depend on the input. Even if we require that $\varepsilon' = \exp(-p(n))$ for an arbitrary polynomial p, the composite TM still runs in polynomial time.)

Let us we give an equivalent definition of the class BPP using predicates in two variables (this definition is similar to Definition 3.2).

Definition 4.2. A predicate L belongs to BPP if there exist a polynomial p and a predicate R, decidable in polynomial time, such that

$L(x) = 1 \;\Rightarrow$ the fraction of strings r of length $p(|x|)$ satisfying $R(x,r)$ is greater than $1-\varepsilon$;

$L(x) = 0 \;\Rightarrow$ the fraction of strings r of length $p(|x|)$ satisfying $R(x,r)$ is less than ε.

Theorem 4.1. *Definitions* 4.1 *and* 4.2 *are equivalent.*

Proof. *Definition* 4.1 $\Rightarrow$ *Definition* 4.2. Let $R(x,r)$ be the following predicate: "M says 'yes' for the input x using $r_1,\dots,r_{p_n}$ as the random bits" (we assume that a coin is tossed at each step of M). It is easy to see that the requirements of Definition 4.2 are satisfied.

Definition 4.2 $\Rightarrow$ *Definition* 4.1. Assume that p and R are given. Consider a PTM that (for input x) randomly chooses a string r of length $p(|x|)$, making $p(|x|)$ coin tosses, and then computes $R(x,r)$. This machine satisfies Definition 4.1 (with a different polynomial p' instead of p). □

Definition 4.2 is illustrated in Figure 4.1. We represent a pair (x,y) of strings as a point and draw the set $S = \big\{(x,y) : (|y| = p(|x|)) \wedge R(x,y)\big\}$. For each x we consider the x-section of S defined as $S_x = \{y : (x,y) \in S\}$. The set S is rather special in the sense that, for any x, either S_x is large (contains at least $1-\varepsilon$ fraction of all strings of length $p(|x|)$) or is small

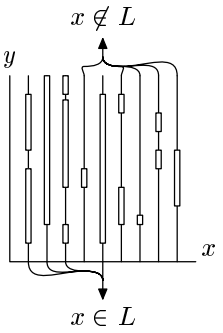


Fig. 4.1. The characteristic set of the predicate $R(x, y)$. Vertical lines represent the sets S_x.

(contains at most ε fraction of them). Therefore all values of x are divided into two categories: for one of them $L(x)$ is true and for the other $L(x)$ is false.

Remark 4.1. Probabilistic Turing machines (unlike nondeterministic ones, which depend on almighty Merlin for guessing the computational path) can be considered as real computing devices. Physical processes like the Nyquist noise or radioactive decay are believed to provide sources of random bits; in the latter case it is guaranteed by the very nature of quantum mechanics.

4.2. Primality testing. A classic example of a BPP problem is checking whether a given integer q (represented by $n = \lceil \log_2 q \rceil$ bits) is prime or not. We call this problem PRIMALITY. We will describe a probabilistic primality test, called *Miller–Rabin test*. For reader's convenience all necessary facts from number theory are given in Appendix A.

4.2.1. *Main idea.* We begin with a much simpler "Fermat test", though its results are generally inconclusive. It is based on Fermat's little theorem (Theorem A.9), which says that

$$\text{if } q \text{ is prime, then} \quad a^{q-1} \equiv 1 \pmod{q} \quad \text{for } x \in \{1, \dots, q-1\}.$$

We may regard a as a (mod q)-residue and simply write $a^{q-1} = 1$, assuming that arithmetic operations are performed with residues rather than integers.

So, the test is this: we pick a random a and check whether $a^{q-1} = 1$. If this is true, then q *may* be a prime; but if this is false, then q is *not* a prime. Such a can be called a *witness* saying that a is composite. This kind of evidence is indirect (it does not give us any factor of q) but usually easy to find: it often suffices to check $a = 2$. But we will do a better job if we sample a from the uniform distribution over the set $\{1, \dots, q-1\}$ (i.e., each element of this set is taken with probability $1/(q-1)$).

Suppose q is composite. We want to know if the test actually shows that with nonnegligible probability. Consider two cases.

1) $\gcd(a,q) = d \neq 1$. Then $a^{q-1} \equiv 0 \not\equiv 1 \pmod d$, therefore $a^{q-1} \not\equiv 1 \pmod q$. The test detects that q is composite. Unfortunately, the probability to get such an a is usually small.
2) $\gcd(a,q) = 1$, i.e. $a \in (\mathbb{Z}/q\mathbb{Z})^*$ (where $(\mathbb{Z}/q\mathbb{Z})^*$ denotes the group of invertible (mod q)-residues). This is the typical case; let us consider it more closely.

Lemma 4.2. *If $a^{q-1} \neq 1$ for some element $a \in (\mathbb{Z}/q\mathbb{Z})^*$, then the Fermat test detects the compositeness of q with probability $\geq 1/2$.*

Proof. Let $G = (\mathbb{Z}/q\mathbb{Z})^*$. For any integer k define the following set:

$$G_{(m)} = \bigl\{x \in G :\ x^m = 1\bigr\}. \tag{4.2}$$

This is a subgroup in G (due to the identity $a^m b^m = (ab)^m$ for elements of an Abelian group). If $a^{q-1} \neq 1$ for some a, then $a \notin G_{(q-1)}$, therefore $G_{(q-1)} \neq G$. By Lagrange's theorem, the ratio $|G| \,/\, |G_{(m)}|$ is an integer, hence $|G| \,/\, |G_{(m)}| \geq 2$. It follows that $a^{q-1} \neq 1$ for at least half of $a \in (\mathbb{Z}/q\mathbb{Z})^*$. And, as we already know, $a^{q-1} \neq 1$ for all $a \notin (\mathbb{Z}/q\mathbb{Z})^*$. □

Is it actually possible that q is composite but $a^{q-1} = 1$ for all invertible residues a? Such numbers q are rare, but they exist (they are called *Carmichael numbers*). Example: $q = 561 = 3 \cdot 11 \cdot 17$. Note that the numbers $3-1$, $11-1$ and $17-1$ divide $q-1$. Therefore $a^{q-1} = 1$ for any $a \in (\mathbb{Z}/q\mathbb{Z})^* \cong \mathbb{Z}_{3-1} \times \mathbb{Z}_{11-1} \times \mathbb{Z}_{17-1}$.

We see that the Fermat test alone is not sufficient to detect a composite number. The Miller–Rabin test uses yet another type of witnesses for the compositeness: if $b^2 \equiv 1 \pmod q$, and $b \not\equiv \pm 1 \pmod q$ for some b, then q is composite. Indeed, in this case $b^2 - 1 = (b-1)(b+1)$ is a multiple of q but $b-1$ and $b+1$ are not, therefore q has nontrivial factors in common with both $b-1$ and $b+1$.

4.2.2. *Required subroutines and their complexity.* Addition (or subtraction) of n-bit integers is done by an $O(n)$-size circuit; multiplication and division are performed by $O(n^2)$-size circuits. These estimates refer to the standard algorithms learned in school, though they are not the most efficient for large integers. In the solutions to Problems 2.12, 2.13 and 2.14 we described alternative algorithms, which are much better in terms of the circuit depth, but slightly worse in terms of the size. If only the size is important, the standard addition algorithm is optimal, but the ones for the multiplication and division are not. For example, an $O(n \log n \log\log n)$-size circuit for the multiplication exists; see [**5**, Sec. 7.5] or [**43**, vol. 2, Sec. 4.3.3].

Euclid's algorithm for $\gcd(a,b)$ has complexity $O(n^3)$, but there is also a so-called "binary" gcd algorithm (see [**43**, vol. 2, Sec. 4.5.2]) of complexity $O(n^2)$. It does not solve the equation $xa+yb=\gcd(a,b)$, though.

We will also use modular arithmetic. It is clear that the addition of (mod q)-residues is done by a circuit of size $O(n)$, whereas modular multiplication can be reduced to integer multiplication and division; therefore it is performed by a circuit of size $O(n^2)$ (by the standard technique). To invert a residue $a \in (\mathbb{Z}/q\mathbb{Z})^*$, we need to find an integer x such that $xa \equiv 1 \pmod q$, i.e., $xa+yq=1$. This is done by extended Euclid's algorithm, which has complexity $O(n^3)$.

It is also possible to compute $(a^m \bmod q)$ by a polynomial time algorithm. (Note that we speak about an algorithm that is polynomial in the length n of its input (a,m,q); therefore performing m multiplications is out of the question. Note also that the size of the integer a^m is exponential.) But we can compute $(a^{2^k} \bmod q)$ for $k=1,2,\dots,\lfloor \log_2 m \rfloor$ by repeated squaring, applying the (mod q) reduction at each step. Then we multiply some of the results in such a way that the powers, i.e., the numbers 2^k, add to m (using the binary representation of m). This takes $O(\log m)=O(n)$ modular multiplications, which translates to circuit size $O(n^3)$.

4.2.3. ***The algorithm.*** Assume that a number q is given.

Step 1. If q is even (and $q \neq 2$), then q is composite. If q is odd, we proceed to Step 2.

Step 2. Let $q-1=2^k l$, where $k>0$, and l is odd.

Step 3. We choose a random $a \in \{1,\dots,q-1\}$.

Step 4. We compute $a^l, a^{2l}, a^{4l}, \dots, a^{2^k l}=a^{q-1}$ modulo q.

Test 1. If $a^{q-1} \neq 1$ (where modular arithmetic is assumed), then q is composite.

Test 2. If the sequence $a^l, a^{2l}, \dots, a^{2^k l}$ (Step 4) contains a 1 that is preceded by anything except ± 1, then q is composite. In other words, if there exists j such that $a^{2^j l} \neq \pm 1$ but $a^{2^{j+1} l}=1$, then q is composite.

In all other cases the algorithm says that "q is prime" (though in fact it is not guaranteed).

Theorem 4.3. *If q is prime, then the algorithm always (with probability 1) gives the answer "prime".*

If q is composite, then the answer "composite" is given with probability at least $1/2$.

Remark 4.2. To get a probabilistic algorithm in sense of Definition 4.1, we repeat this test twice: the probability of an error (a composite number being undetected twice) is at most $1/4 < 1/2$.

Proof of Theorem 4.3. As we have seen, the algorithm always gives the answer "prime" for prime q.

Assume that q is composite (and odd). If $\gcd(a, q) > 1$ then Test 1 shows that q is composite. So, we may assume that that a is uniformly distributed over the group $G = (\mathbb{Z}/q\mathbb{Z})^*$. We consider two major cases.

Case A: $q = p^\alpha$, where p is an odd prime, and $\alpha > 1$. We show that there is an invertible (mod q)-residue x such that $x^{q-1} \neq 1$, namely $x = (1 + p^{\alpha-1}) \bmod q$. Indeed, $x^{-1} = (1 - p^{\alpha-1}) \bmod q$, and[4]

$$\begin{aligned} x^{q-1} &\equiv (1 + p^{\alpha-1})^{q-1} = 1 + (q-1)p^{\alpha-1} + \text{ higher powers of } p \\ &\equiv 1 - p^{\alpha-1} \not\equiv 1 \pmod{q}. \end{aligned}$$

Therefore Test 1 detects the compositeness of q with probability $\geq 1/2$ (due to Lemma 4.2).

Case B: q has at least two distinct prime factors. Then $q = uv$, where u and v are odd numbers, $u, v > 1$, and $\gcd(u, v) = 1$. The Chinese remainder theorem (Theorem A.5) says that the group $G = (\mathbb{Z}/q\mathbb{Z})^*$ is isomorphic to the direct product $U \times V$, where $U = (\mathbb{Z}/u\mathbb{Z})^*$ and $V = (\mathbb{Z}/v\mathbb{Z})^*$, and that each element $x \in G$ corresponds to the pair $\big((x \bmod u), (x \bmod v)\big)$.

For any m we define the following subgroup (cf. formula (4.2)):

$$G^{(m)} = \big\{x^m : x \in G\big\} = \operatorname{Im} \varphi_m, \qquad \text{where } \varphi_m : x \mapsto x^m.$$

Note that $G^{(m)} = \{1\}$ if and only if $G_{(m)} = G$; this is yet another way to formulate the condition that $a^m = 1$ for all $a \in G$. Also note that if a is uniformly distributed over G, then a^m is uniformly distributed over $G^{(m)}$. Indeed, the map $\varphi_m : x \mapsto x^m$ is a group homomorphism; therefore the number or pre-images is the same for all elements of $G^{(m)}$. It is clear that $G^{(m)} \cong U^{(m)} \times V^{(m)}$. Now we have two subcases.

Case 1. $U^{(q-1)} \neq \{1\}$ *or* $V^{(q-1)} \neq \{1\}$. This condition implies that $G^{(q-1)} \neq \{1\}$, so that Test 1 detects q being composite with probability at least $1/2$.

Case 2. $U^{(q-1)} = \{1\}$ *and* $V^{(q-1)} = \{1\}$. In this case Test 1 is always passed, so we have to study Test 2. Let us define two sequences of subgroups:

$$U^{(l)} \supseteq U^{(2l)} \supseteq \cdots \supseteq U^{(2^k l)} = \{1\}, \qquad V^{(l)} \supseteq V^{(2l)} \supseteq \cdots \supseteq V^{(2^k l)} = \{1\}.$$

[4] A similar argument is used to prove that the group $(\mathbb{Z}/p^\alpha\mathbb{Z})^*$ is cyclic; see Theorem A.11.

Note that $U^{(l)} \neq \{1\}$ and $V^{(l)} \neq \{1\}$. Specifically, both $U^{(l)}$ and $V^{(l)}$ contain the residues that correspond to -1. Indeed, both U and V contain -1, and $(-1)^l = -1$ since l is odd.

Going from right to left, we find the first place where one of the sets $U^{(m)}$, $V^{(m)}$ contains an element different from 1. It other words, we find $t = 2^s l$ such that $0 \leq s < k$, $U^{(2t)} = \{1\}$, $V^{(2t)} = \{1\}$, and either $U^{(t)} \neq \{1\}$ or $V^{(t)} \neq \{1\}$.

We will prove that Test 2 shows (with probability at least 1/2) that q is composite. By our assumption $a^{2t} = 1$, so we need to know the probability of the event $a^t \neq \pm 1$. Let us consider two possibilities.

Case 2a. *One of the sets* $U^{(t)}$, $V^{(t)}$ *equals* $\{1\}$ (for example, let $U^{(t)} = \{1\}$). This means that for any a the pair $\big((a^t \bmod u), (a^t \bmod v)\big)$ has 1 as the first component. Therefore $a^t \neq -1$, since -1 is represented by the pair $(-1, -1)$.

At the same time, $V^{(t)} \neq \{1\}$; therefore the probability that a^t has 1 in the second component is at most 1/2 (by Lagrange's theorem; cf. proof of Lemma 4.2). Thus Test 2 says "composite" with probability at least 1/2.

Case 2b. *Both sets* $U^{(t)}$ *and* $V^{(t)}$ *contain at least two elements:* $|U^{(t)}| = c \geq 2$, $|V^{(t)}| = d \geq 2$. In this case a^t has 1 in the first component with probability $1/c$ (there are c equiprobable possibilities) and has 1 in the second component with probability $1/d$. These events are independent due to the Chinese remainder theorem, so the probability of the event $a^t = 1$ is $1/cd$. For similar reasons the probability of the event $a^t = -1$ is either 0 or $1/cd$. In any case the probability of the event $a^t = \pm 1$ is at most $2/cd \leq 1/2$. □

4.3. BPP and circuit complexity.

Theorem 4.4. $\mathrm{BPP} \subseteq \mathrm{P/poly}$.

Proof. Let $L(x)$ be a BPP-predicate, and M a probabilistic TM that decides $L(x)$ with probability at least $1 - \varepsilon$. By running M repeatedly we can decrease the error probability. Recall that a polynomial number of repetitions leads to the exponential decrease. Therefore we can construct a polynomial probabilistic TM M' that decides $L(x)$ with error probability less that $\varepsilon' < 1/2^n$ for inputs x of length n.

The machine M' uses a random string r (one random bit for each step). For each input x of length n, the fraction of strings r that lead to an incorrect answer is less than $1/2^n$. Therefore the overall fraction of "bad" pairs (x, r) among all such pairs is less than $1/2^n$. [If one represents the set of all pairs (x, r) as a unit square, the "bad" subset has area $< 1/2^n$.] It follows that there exists $r = r_*$ such that the fraction of bad pairs (x, r) is less than

$1/2^n$ among all pairs with $r = r_*$. However, there are only 2^n such pairs (corresponding to 2^n possibilities for x). The only way the fraction of bad pairs can be smaller than $1/2^n$ is that there are no bad pairs at all!

Thus we conclude that there is a particular string r_* that produces correct answers for all x of length n.

The machine M' can be transformed into a polynomial-size circuit with input (x, r). Then we fix the value of r (by setting $r = r_*$) and obtain a polynomial-size circuit with input x that decides $L(x)$ for all n-bit strings x. □

This is a typical nonconstructive existence proof: we know that a "good" string r_* exists (by probability counting) but have no means of finding it, apart from exhaustive search.

Remark 4.3. It might well be the case that BPP = P. Let us explain briefly the motivation of this conjecture and why it is hard to prove.

Speaking about proved results, it is clear that BPP $\subseteq$ PSPACE. Indeed, the algorithm that counts all values of the string r that lead to the answer "yes" runs in polynomial space. Note that the running time of this algorithm is $2^N \operatorname{poly}(n)$, where $N = |r|$ is the number of random bits.

On the other hand, there is empirical evidence that probabilistic algorithms usually work well with pseudo-random bits instead of truly random ones. So attempts have been made to construct a mathematical theory of pseudo-random numbers. The general idea is as follows. A *pseudo-random generator* is a function $g : \mathbb{B}^l \to \mathbb{B}^L$ which transforms short truly random strings (of length l, which can be as small as $O(\log L)$) into much longer pseudo-random strings (of length L). "Pseudo-random" means that any computational device with limited resources (say, any Boolean circuit of a given size N computing a function $F : \mathbb{B}^L \to \mathbb{B}$) is unable to distinguish between truly random and pseudo-random strings of length L. Specifically, we require that

$$\Big|\mathbf{Pr}\big[F(g(x)) = 1\big] - \mathbf{Pr}\big[F(y) = 1\big]\Big| \le \delta, \qquad x \in \mathbb{B}^l,\ \ y \in \mathbb{B}^L$$

for some constant $\delta < 1/2$, where x and y are sampled from the uniform distributions. (In this definition the important parameters are l and N, while L should fit the number of input bits of the circuit. For simplicity let us require that $L = N$: it will not hurt if the pseudo-random generator produces some extra bits.)

It is easy to show that pseudo-random generators $g : \mathbb{B}^{O(\log L)} \to \mathbb{B}^L$ exist: if we choose the function g randomly, it fulfills the above condition with high probability. What we actually need is a sequence of efficiently computable pseudo-random generators $g_L : \mathbb{B}^{l(L)} \to \mathbb{B}^L$, where $l(L) = O(\log L)$.

If such pseudo-random generators exist, we can use their output instead of truly random bits in any probabilistic algorithm. The definition of pseudo-randomness guarantees that this will work, provided the running time of the algorithm is limited by $c\sqrt{L}$ (for a suitable constant c) and the error probability ε is smaller than $1/2 - \delta$. (With pseudo-random bits the error probability will be $\varepsilon + \delta$, which is still less than $1/2$. The estimate $c\sqrt{L}$ comes from the simulation of a Turing machine by Boolean circuits, see Theorem 2.2.) Thus we decrease the number of needed genuine random bits from L to l. Then we can *derandomize* the algorithm by trying all 2^l possibilities. If $l = O(\log L)$, the resulting computation has polynomial complexity.

We do not know whether efficiently computable pseudo-random generators exist. The trouble is that the definition has to be satisfied for "any Boolean circuit of a given size"; this condition is extremely hard to deal with. But we may try to reduce this problem to a more fundamental one — obtaining lower bounds for the circuit complexity of Boolean functions. Even this idea, which sets the most difficult part of the problem aside, took many years to realize. Much work in this area was done in 1980's and early 1990's, but the results were not as strong as needed. Recently there has been dramatic progress leading to more efficient constructions of pseudo-random generators and new derandomization techniques. It has been proved that BPP = P if there is an algorithm with running time $\exp(O(n))$ that computes a sequence of functions with circuit complexity $\exp(\Omega(n))$ [**32**]. We also mention a new work [**69**] in which pseudo-random generators are constructed from arbitrary hard functions in an optimal (up to a polynomial) way.

5. The hierarchy of complexity classes

Recall that we identify languages (sets of strings) and predicates (and $x \in L$ means $L(x) = 1$).

Definition 5.1. Let A be some class of languages. The dual class co-A consists of the complements of all languages in A. Formally,

$$L \in \text{co-}A \Leftrightarrow (\mathbb{B}^* \setminus L) \in A.$$

It follows immediately from the definitions that P = co-P, BPP = co-BPP, PSPACE = co-PSPACE.

5.1. Games machines play. Consider a game with two players called White (W) and Black (B). A string x is shown to both players. After that, players alternately choose binary strings: W starts with some string w_1, B

replies with b_1, then W says w_2, etc. Each string has length polynomial in $|x|$. Each player is allowed to see the strings already chosen by his opponent.

The game is completed after some prescribed number of steps, and the referee, who knows x and all the strings and who acts according to a polynomial-time algorithm, declares the winner. In other words, there is a predicate $W(x, w_1, b_1, w_2, b_2, \dots)$ that is true when W is the winner, and we assume that this predicate belongs to P. If this predicate is false, B is the winner (there are no ties). This predicate (together with polynomial bounds for the length of strings and the number of steps) determines the game.

Let us note that in fact the termination rule can be more complicated, but we always assume that the number of moves is bounded by a polynomial. Therefore we can "pad" the game with dummy moves that are ignored by the referee and consider only games where the number of moves is known in advance and is a polynomial in the input length.

Since this game is finite and has no ties, for each x either B or W has a winning strategy. (One can formally prove this using induction over the number of moves.) Therefore, any game determines two complementary sets,

$$L_w = \{x\colon \text{ W has a winning strategy}\},$$
$$L_b = \{x\colon \text{ B has a winning strategy}\}.$$

Many complexity classes can be defined as classes formed by the sets L_w (or L_b) for some classes of games. Let us give several examples.

P: the sets L_w (or L_b) for games of zero length (the referee declares the winner after he sees the input)

NP: the sets L_w for games that are finished after W's first move. In other words, NP-sets are sets of the form

$$\{x : \exists\, w_1 W(x, w_1)\}.$$

co-NP: the sets L_b for games that are finished after W's first move. In other words, co-NP-sets are sets of the form

$$\{x : \forall\, w_1\, B(x, w_1)\}$$

(here $B = \neg W$ means that B wins the game.)

Σ_2: the sets L_w for games where W and B make one move each and then the referee declares the winner. In other words, Σ_2-sets are sets of the form

$$\{x : \exists\, w_1 \forall\, b_1\ W(x, w_1, b_1)\}.$$

(W can make a winning move w_1 after which any move b_1 of B makes B lose).

Π_2: the sets L_b for the same class of games, i.e., the sets of the form

$$\{x : \forall\, w_1\, \exists\, b_1\ B(x, w_1, b_1)\}.$$

...

Σ_k: the sets L_w for games of length k (the last move is made by W if k is odd or by B is k is even), i.e., the sets

$$\{x : \exists\, w_1\, \forall\, b_1 \ldots\, \mathbf{Q}_k\, y_k\, W(x, w_1, b_1, \ldots)\}$$

(if k is even, then $\mathbf{Q}_k = \forall$, $y_k = b_{k/2}$; if k is odd, then $\mathbf{Q}_k = \exists$, $y_k = w_{(k+1)/2}$).

Π_k: the sets L_b for the same class of games, i.e., the sets

$$\{x :\ \forall\, w_1\ \exists\, b_1 \ldots\, \mathbf{Q}_k\, y_k\, B(x, w_1, b_1, \ldots)\}$$

(if k is even, then $\mathbf{Q}_k = \exists$, $y_k = b_{k/2}$; if k is odd, then $\mathbf{Q}_k = \forall$, $y_k = w_{(k+1)/2}$).

...

Evidently, complements of Σ_k-sets are Π_k-sets and vice versa: $\Sigma_k = \text{co-}\Pi_k$, $\Pi_k = \text{co-}\Sigma_k$.

Theorem 5.1 (Lautemann [46]). BPP $\subseteq \Sigma_2 \cap \Pi_2$.

Proof. Since BPP=co-BPP, it suffices to show that BPP $\subseteq \Sigma_2$.

Let us assume that $L \in$ BPP. Then there exist a predicate R (computable in polynomial time) and a polynomial p such that the fraction $|S_x|/2^N$, where

$$S_x = \big\{y \in \mathbb{B}^N :\ R(x, y)\big\}, \qquad N = p(|x|), \tag{5.1}$$

is either large (greater than $1-\varepsilon$ for $x \in L$) or small (less than ε for $x \notin L$).

To show that $L \in \Sigma_2$, we need to reformulate the property "X is a large subset of G" (where G is the set of all strings y of length N) using existential and universal quantifiers.

This could be done if we impose a group structure on G. Any group structure will work if the group operations are polynomial-time computable. For example, we can consider an additive group formed by bit strings of a given length with bit-wise addition modulo 2.

The property that distinguishes large sets from small ones is the following: "several copies of X shifted by some elements cover G", i.e.,

$$\exists\, g_1, \ldots, g_m\ \Big(\bigcup\nolimits_i (g_i + X) = G\Big), \tag{5.2}$$

where "+" denotes the group operation. To choose an appropriate value for m, let us see when (5.2) is guaranteed to be true (or false).

It is obvious that condition (5.2) is false if

$$m|X| < |G|. \tag{5.3}$$

On the other hand, (5.2) is true if for independent random $g_1, \dots, g_m \in G$ the probability of the event $\bigcup_i (g_i + X) = G$ is positive; in other words, if $\mathbf{Pr}\big[\bigcup_i (g_i + X) \neq G\big] < 1$. Let us estimate this probability.

The probability that a random shift $g + X$ does not contain a fixed element $u \in G$ (for a given X and random g) is $1 - |X|/|G|$. When $g_1, \dots, g_m$ are chosen independently, the corresponding sets $g_1 + X, \dots, g_m + X$ do not cover u with probability $(1 - |X|/|G|)^m$. Summing these probabilities over all $u \in G$, we see that the probability of the event $\bigcup_i (g_i + X) \neq G$ does not exceed $|G|(1 - |X|/|G|)^m$.

Thus condition (5.2) is true if

$$|G|\big(1 - |X|/|G|\big)^m < 1. \tag{5.4}$$

Let us now apply these results to the set $X = S_x$ (see formula (5.1)). We want to satisfy (5.3) and (5.4) when $|S_x|/2^N < \varepsilon$ (i.e., $x \notin L$) and when $|S_x|/2^N > 1 - \varepsilon$ (i.e., $x \in L$), respectively. Thus we get the inequalities $\varepsilon m < 1$ and $2^N \varepsilon^m < 1$, which should be satisfied simultaneously by a suitable choice of m. This is not always possible if N and ε are fixed. Fortunately, we have some flexibility in the choice of these parameters. Using "amplification of probability" by repeating the computation k times, we increase N by factor of k, while decreasing ε exponentially. Let the initial value of ε be a constant, and λ given by (4.1). The amplification changes N and ε to $N' = kN$ and $\varepsilon' = \lambda^k$. Thus we need to solve the system

$$\lambda^k m < 1, \qquad 2^{kN} \lambda^{km} < 1$$

by adjusting m and k. It is obvious that there is a solution with $m = O(N)$ and $k = O(\log N)$.

We have proved that $x \in L$ is equivalent to the following Σ_2-condition:

$$\exists g_1, \dots, g_m \, \forall y \, \Big(\big(|y| = p'(|x|)\big) \Rightarrow \big((y \in g_1 + S'_x) \vee \dots \vee (y \in g_m + S'_x)\big) \Big).$$

Here $p'(n) = kp(|x|)$ (k and m also depend on $|x|$), whereas S'_x is the "amplified" version of S_x.

In other words, we have constructed a game where W names m strings (group elements) $g_1, \dots, g_m$, and B chooses some string y. If y is covered by some $g_i + S'_x$ (which is easy to check: it means that $y - g_i$ belongs to S'_x), then W wins; otherwise B wins. In this game W has a winning strategy if and only if S'_x is big, i.e., if $x \in L$. □

5.2. The class PSPACE. This class contains predicates that can be computed by a TM running in polynomial (in the input length) space. The class PSPACE also has a game-theoretic description:

Theorem 5.2. $L \in$ PSPACE *if and only if there exists a polynomial game such that*

$$L = \{x\colon W \textit{ has a winning strategy for input } x\}.$$

By a polynomial game we mean a game where the number of moves is bounded by a polynomial (in the length of the input), players' moves are strings of polynomial length, and the referee's algorithm runs in polynomial time.

Proof. $\Leftarrow$ We show that a language determined by a game belongs to PSPACE. Let the number of turns be $p(|x|)$. We construct a sequence of machines $M_1, \dots, M_{p(|x|)}$. Each M_k gets a prefix $x, w_1, b_1, \dots$ of the play that includes k moves and determines who has the winning strategy in the remaining game.

The machine $M_{p(|x|)}$ just computes the predicate $W(x, w_1, \dots)$ using referee's algorithm. The machine M_k tries all possibilities for the next move and consults M_{k+1} to determine the final result of the game for each of them. Then M_k gives an answer according to the following rule, which says whether W wins. If it is W's turn, then it suffices to find a single move for which M_{k+1} declares W to be the winner. If it is B's turn, then W needs to win after all possible moves of B.

The machine M_0 says who is the winner before the game starts and therefore decides $L(x)$. Each machine in the sequence $M_1, \dots, M_{p(|x|)}$ uses only a small (polynomially bounded) amount of memory, so that the composite machine runs in polynomial space. (Note that the computation time is exponential since each of the M_k calls M_{k+1} many times.)

$\Rightarrow$ Let M be a machine that decides the predicate L and runs in polynomial space s. We may assume that computation time is bounded by $2^{O(s)}$. Indeed, there are $2^{O(s)}$ different configurations, and after visiting the same configuration twice the computation repeats itself, i.e., the computation becomes cyclic.

[To see why there are at most $2^{O(s)}$ configurations note that configuration is determined by head position (in $\{0, 1, \dots, s\}$), internal state (there are $|Q|$ of them) and the contents of the s cells of the tape ($|A|^s$ possibilities where A is the alphabet of TM); therefore the total number of configurations is $|A|^s \cdot |Q| \cdot s = 2^{O(s)}$.]

Therefore we may assume without loss of generality that the running time of M on input x is bounded by 2^q, where $q = \mathrm{poly}(|x|)$.

In the description of the game given below we assume that TM keeps its configuration unchanged after the computation terminates.

During the game, W claims that M's result for an input string x is "yes", and B wants to disprove this. The rules of the game allow W to win if $M(x)$ is indeed "yes" and allow B to win if $M(x)$ is not "yes".

In his first move W declares the configuration of M after 2^{q-1} steps dividing the computation into two parts. B can choose any of the parts: either the time interval $[0, 2^{q-1}]$ or the interval $[2^{q-1}, 2^q]$. (B tries to catch W by choosing the interval where W is cheating.) Then W declares the configuration of M at the middle of the interval chosen by B and divides this interval into two halves, B selects one of the halves, W declares the configuration of M at the middle, etc.

The game ends when the length of the interval becomes equal to 1. Then the referee checks whether the configurations corresponding to the ends of this interval match (the second is obtained from the first according to M's rules). If they match, then W wins; otherwise B wins.

If M's output on x is really "yes", then W wins if he is honest and declares the actual configuration of M. If M's output is "no", then W is forced to cheat: his claim is incorrect for (at least) one of the halves. If B selects this half at each move, than B can finally catch W "on the spot" and win. □

[2!] **Problem 5.1.** Prove that any predicate $L(x)$ that is recognized by a nondeterministic machine in space $s = \text{poly}(|x|)$ belongs to PSPACE. (A predicate L is recognized by an NTM M in space $s(|x|)$ if for any $x \in L$ there exists a computational path of M that gives the answer "yes" using at most $s(|x|)$ cells and, for each $x \notin L$, no computational path of M ends with "yes".)

Theorem 5.2 shows that all the classes Σ_k, Π_k are subsets of PSPACE. Relations between these classes are represented by the inclusion diagram in Figure 5.1. The smallest class is P (games of length 0); P is contained in both classes NP and co-NP(which correspond to games of length 1); classes NP and co-NP are contained in Σ_2 and Π_2 (games with two moves) and so on. We get the class PSPACE, allowing the number of moves in a game be polynomial in $|x|$.

We do not know whether the inclusions in the diagram are strict. Computer scientists have been working hard for several decades trying to prove at least something about these classes, but the problem remains open. It is possible that P = PSPACE (though this seems very unlikely). It is also possible that PSPACE = EXPTIME, where EXPTIME is the class of languages decidable in time $2^{\text{poly}(n)}$. Note, however, that P $\neq$ EXPTIME — one can

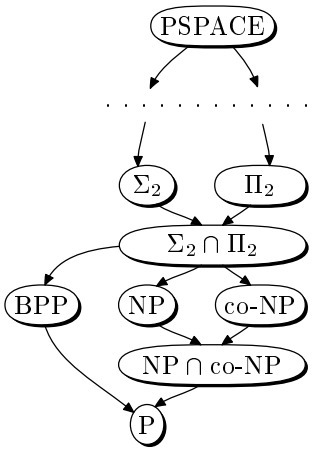


Fig. 5.1. Inclusion diagram for computational classes. An arrow from A to B means that B is a subset of A.

prove this by a rather simple "diagonalization argument" (cf. solution to Problem 1.3).

[3!] **Problem 5.2.** A *Turing machine with oracle for language* L uses a decision procedure for L as an external subroutine (cf. Definition 2.2). The machine has a supplementary *oracle tape*, where it can write strings and then ask the "oracle" whether the string written on the oracle tape belongs to L.

Prove that any language that is decidable in polynomial time by a TM with oracle for some $L \in \Sigma_k$ (or $L \in \Pi_k$) belongs to $\Sigma_{k+1} \cap \Pi_{k+1}$.

The class PSPACE has complete problems (to which any problem from PSPACE is reducible). Here is one of them.

The *TQBF* Problem is given by the predicate

$TQBF(x) \Leftrightarrow$ x is a True Quantified Boolean Formula, i.e., a true statement of type $Q_1 y_1 \ldots Q_n y_n F(y_1, \ldots, y_n)$, where variables y_i range over $\mathbb{B} = \{0, 1\}$, F is some propositional formula (involving $y_1, \ldots, y_n$, $\neg, \wedge, \vee$), and Q_i is either $\forall$ or $\exists$.

By definition, $\forall y\, A(y)$ means $(A(0) \wedge A(1))$ and $\exists y\, A(y)$ means $(A(0) \vee A(1))$.

Theorem 5.3. *TQBF is* PSPACE*-complete.*

Proof. We reduce an arbitrary language $L \in$ PSPACE to *TQBF*. Using Theorem 5.2, we construct a game that corresponds to L. Then we convert a TM that computes the result of the game (a predicate W) into a circuit. Moves of the players are encoded by Boolean variables. Then the existence of the winning strategy for W can be represented by a quantified Boolean

formula

$$\exists\, w_1^1\, \exists\, w_1^2 \ldots \exists\, w_1^{p(|x|)}\ \forall\, b_1^1 \ldots \forall\, b_1^{p(|x|)}\ \exists\, w_2^1 \ldots \exists\, w_2^{p(|x|)} \ldots\ S(x, w_1^1, w_1^2, \ldots),$$

where $S(\cdot)$ denotes the Boolean function computed by the circuit. (Boolean variables $w_1^1, \ldots, w_1^{p(|x|)}$ encode the first move of W, variables $b_1^1, \ldots, b_1^{p(|x|)}$ encode B's answer, $w_2^1, \ldots, w_2^{p(|x|)}$ encode the second move of W, etc.)

In order to convert S into a Boolean formula, recall that a circuit is a sequence of assignments $y_i := R_i$ that determine the values of auxiliary Boolean variables y_i. Then we can replace $S(\cdot)$ by a formula

$$\exists y_1, \ldots, \exists y_s \big((y_1 \Leftrightarrow R_1) \wedge \cdots \wedge (y_s \Leftrightarrow R_s) \wedge y_s\big),$$

where s is the size of the circuit.

After this substitution we obtain a quantified Boolean formula which is true if and only if $x \in L$. □

Remark 5.1. Note the similarity between Theorem 5.3 (which is about polynomial space computation) and Problems 2.17 and 2.18 (which are basically about poly-logarithmic space computation). Also note that a polynomial-size quantified Boolean formula may be regarded as a polynomial depth circuit (though of very special structure): the $\forall$ and $\exists$ quantifiers are similar to the $\wedge$ and $\vee$ gates. It is not surprising that the solutions are based on much the same ideas. In particular, the reduction from NTM to *TQBF* is similar to the parallel simulation of a finite-state automaton (see Problem 2.11). However, in the case of *TQBF* we could afford reasoning at a more abstract level: with greater amount of computational resources we were sure that all bookkeeping constructions could be implemented. This is one of the reasons why "big" computational classes (like PSPACE) are popular among computer scientists, in spite of being apparently impractical. In fact, it is sometimes easier to prove something about big classes, and then scale down the problem parameters while fixing some details.

Part 2

Quantum Computation

As already mentioned in the introduction, ordinary computers do not employ all possibilities offered by Nature. Their internal work is based on operations with 0s and 1s, while in Nature there is possibility of performing unitary transformations, i.e., of operating on an infinite set.[1] This possibility is described by quantum mechanics. Devices (real or imaginary) using this possibility are called *quantum computers*.

It is not clear *a priori* whether the computational power is really increased in passing from Boolean functions to unitary transformations on finite-dimensional spaces. However, there is strong evidence that such an increase is actually achieved. For example, consider the factoring problem — decomposition of an integer into prime factors. No polynomial algorithm is known for solving this problem on ordinary computers. But for quantum computers, such algorithms do exist.

Ordinary computers operate with states built from a finite number of bits. Each bit may exist in one of the two states, 0 or 1. The state of the whole system is given by specifying the values of all the bits. Therefore, the set of states $\mathbb{B}^n=\{0,1\}^n$ is finite and has cardinality 2^n.

A quantum computer works with a finite set of objects called *qubits*. Each qubit has two separate states, also denoted by 0 and 1 (if we think of qubits as spins, then these states are "spin up" and "spin down"). The 2^n assignments of individual states to each qubit do not yield all possible states of the system, but they form a *basis* in a space of states. Arbitrary linear combinations of the basis states, with complex coefficients, are also possible. We will denote the basis states by $|x_1,\ldots,x_n\rangle$, where $x_j \in \mathbb{B}$, or

[1] Of course, actual infinity does not occur in Nature. In the given case the essential fact is that a unitary transformation can be performed only with some precision — for details see Section 8.

by $|x\rangle$, where $x \in \mathbb{B}^n$. An arbitrary state of the system may be represented in the form[2]

$$|\psi\rangle = \sum_{(x_1,\dots,x_n)\in\mathbb{B}^n} c_{x_1,\dots,x_n}|x_1,\dots,x_n\rangle, \text{ where } \sum_{(x_1,\dots,x_n)\in\mathbb{B}^n} |c_{x_1,\dots,x_n}|^2 = 1.$$

The *state space* for such a system is a linear space of dimension 2^n over the field $\mathbb{C}$ of complex numbers.

State

of an ordinary computer	**of a quantum computer**
$\square\ \square\ \cdots\ \square$ bits	$\square\ \square\ \cdots\ \square$ qubits
$x_1\ x_2\ \dots\ x_n\ \ x_j \in \mathbb{B}$	basis: $\lvert\, x_1,\ x_2,\ \dots,\ x_n \,\rangle$, $\quad x_j \in \mathbb{B}$
	arbitrary: $\sum\limits_{x\in\mathbb{B}^n} c_x\lvert x\rangle$, where $\sum\limits_{x\in\mathbb{B}^n} \lvert c_x\rvert^2 = 1$

One detail to add: if we multiply the vector $\sum_x c_x|x\rangle$ by a *phase factor* $e^{i\varphi}$ (φ real), we obtain a physically indistinguishable state. Therefore, a state of a quantum computer is a unit vector defined up to a phase factor.

Computation may be imagined as a sequence of transformations on the set of states of the system. Let us describe which transformations are possible in the classical and in the quantum case:

Classical case:	**Quantum case:**
transformations are functions from $\mathbb{B}^n$ to $\mathbb{B}^n$.	transformations are unitary operators, i.e., operators that preserve the length $\sum\limits_{x\in\mathbb{B}^n} \lvert c_x\rvert^2$ of each vector $\sum\limits_{x\in\mathbb{B}^n} c_x\lvert x\rangle$.

Remark. All that has been said pertains only to isolated systems. A real quantum computer is (will be) a part of a larger system (the Universe), interacting with the remaining world. Quantum states and transformations of open systems will be considered in Sections 10, 11.

Now we must give a formal definition of quantum computation. As in the classical case, one can define both quantum Turing machines and quantum circuits. We choose the second approach, which is more convenient for a number of reasons.

6. Definitions and notation

6.1. The tensor product. A system of n qubits has a state space $\mathbb{C}^{2^n}$, which can be represented as a tensor product, $\mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 = (\mathbb{C}^2)^{\otimes n}$. The factors correspond to a *space of a single qubit.*

[2]The brackets $|\dots\rangle$ in the notation $|\psi\rangle$ do not signify any operation on the object ψ — they merely indicate that ψ represents a vector.

The tensor product of linear spaces $\mathcal{L}$ and $\mathcal{M}$ can be defined as an arbitrary space $\mathcal{N}$ of dimension $(\dim \mathcal{L})(\dim \mathcal{M})$. The idea is that if $\mathcal{L}$ and $\mathcal{M}$ are endowed with some bases, $\{e_1, \dots, e_l\} \subseteq \mathcal{L}$ and $\{f_1, \dots, f_m\} \subseteq \mathcal{M}$, then $\mathcal{N}$ possesses a standard basis whose elements are associated with pairs (e_j, f_k). We denote these elements by $e_j \otimes f_k$, thus the basis is

$$\big\{e_j \otimes f_k : \ j = 1, \dots, l; \ k = 1, \dots, m\big\}.$$

Using this basis, one can define the tensor product of arbitrary two vectors, $u = \sum_j u_j e_j$ and $v = \sum_k v_k f_k$ $(u_j, v_k \in \mathbb{C})$ in such a way that the map $\otimes : (u, v) \mapsto u \otimes v$ is linear in both u and v:

$$u \otimes v = \sum_{j,k} (u_j v_k)\, e_j \otimes f_k. \tag{6.1}$$

We will mostly use this "pedestrian" definition of the tensor product, although it is not invariant, i.e., it depends on the choice of bases in $\mathcal{L}$ and $\mathcal{M}$. An *invariant* definition is abstract and hard to grasp, but it is indispensable if we really want to prove something. The tensor product of two spaces, $\mathcal{L}$ and $\mathcal{M}$, is a space $\mathcal{N} = \mathcal{L} \otimes \mathcal{M}$, together with a bilinear map $H : \mathcal{L} \times \mathcal{M} \to \mathcal{N}$ (also denoted by $\otimes$, i.e., $u \otimes v \stackrel{\text{def}}{=} H(u, v)$) which satisfy the following *universality property*:

> for any space $\mathcal{F}$ and any bilinear function $F : \mathcal{L} \times \mathcal{M} \to \mathcal{F}$, there is a unique linear function $G : \mathcal{L} \otimes \mathcal{M} \to \mathcal{F}$ such that $F(u, v) = G(u \otimes v)$ (for every pair of $u \in \mathcal{L}$, $v \in \mathcal{M}$.)

[1] **Problem 6.1.** Prove that the tensor product defined in the "pedestrian" way (i.e., using bases) satisfies the universality property.

[1] **Problem 6.2.** Consider two linear maps, $A : \mathcal{L} \to \mathcal{L}'$ and $B : \mathcal{M} \to \mathcal{M}'$. Prove that there is a unique linear map $C = A \otimes B : \mathcal{L} \otimes \mathcal{M} \to \mathcal{L}' \otimes \mathcal{M}'$ such that $C(u \otimes v) = A(u) \otimes B(v)$ for any $u \in \mathcal{L}$, $v \in \mathcal{M}$.

[2] **Problem 6.3.** Show that the pair $(\mathcal{N}, H)$ in the abstract definition of the tensor product is unique. (Figure out in what sense).

6.2. Linear algebra in Dirac's notation. In our case, there is a prechosen *classical basis*: $\{|0\rangle, |1\rangle\}$ for $\mathbb{C}^2$, and $\{|x_1, \dots, x_n\rangle : x_j \in \mathbb{B}\}$ for $(\mathbb{C}^2)^{\otimes n}$. The space $\mathbb{C}^2$ furnished with a basis is denoted by $\mathcal{B}$. The basis is considered orthonormal, which yields an inner product on the space of states. The coefficients $c_{x_1 \dots x_n}$ of the decomposition of a vector $|\psi\rangle$ relative to this basis are called *amplitudes*. Their physical meaning is that the square of the absolute value $|c_{x_1 \dots x_n}|^2$ of the amplitude is interpreted as the probability of finding the system in the given state of the basis. As must be the case, the sum of the probabilities is equal to 1 since the length of the vector is assumed to be 1. (Probabilities will be fully discussed later; for some time we will

be occupied with linear algebra, namely with studying unitary operators on the space $\mathcal{B}^{\otimes n}$.)

We will use (and have already used) notation customary in physics, introduced by Dirac, for vectors and the inner product. Vectors are denoted like this: $|\psi\rangle$; the inner product of two vectors is denoted by $\langle\xi|\eta\rangle$. If $|\xi\rangle = \sum_x a_x|x\rangle$ and $|\eta\rangle = \sum_x b_x|x\rangle$, then $\langle\xi|\eta\rangle = \sum_x a_x^* b_x$. (From now on, a^* stands for the complex conjugate.) In the notation for vectors the brackets are needed only "for elegance" — they indicate the type of the object and offer a symmetric designation (see below). In place of $|\xi\rangle$ we could simply write ξ, even though this is not customary. Thus $|\xi_1+\xi_2\rangle = |\xi_1\rangle + |\xi_2\rangle$, and both expressions mean $\xi_1+\xi_2$.

The inner product is *Hermitian*. It is conjugate-linear in the first argument[3] and linear in the second, i.e.,

$$\begin{aligned} &\langle\xi_1+\xi_2|\eta\rangle = \langle\xi_1|\eta\rangle + \langle\xi_2|\eta\rangle, && \langle\xi|\eta_1+\eta_2\rangle = \langle\xi|\eta_1\rangle + \langle\xi|\eta_2\rangle,\\ &\langle c\xi|\eta\rangle = c^*\langle\xi|\eta\rangle, && \langle\xi|c\eta\rangle = c\langle\xi|\eta\rangle. \end{aligned}$$

If we take the left half of the inner product symbol, we get a *bra-vector* $\langle\xi|$, i.e., a linear functional on *ket-vectors* (i.e., the vectors from our space). Bra- and ket-vectors are in a one-to-one correspondence to one another. (Nonetheless, it is necessary to distinguish them in some way — and it is just for this purpose the angle brackets were introduced.) Because of the conjugate linearity of the inner product with respect to the first argument, we have the equation $\langle c\xi| = c^*\langle\xi|$. A bra-vector may be written as a row, and a ket-vector as a column (so as to be able to multiply it on the left by a matrix):

$$\langle\xi| = c_0^*\langle 0| + c_1^*\langle 1| = (c_0^*, c_1^*), \qquad |\xi\rangle = c_0|0\rangle + c_1|1\rangle = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}.$$

The notation $\langle\xi|A|\eta\rangle$, where A is a linear operator, can be interpreted in two ways: either as the product of the bra-vector $\langle\xi|$ and the ket-vector $A|\eta\rangle$, or as the product of $\langle\xi|A$ and $|\eta\rangle$. The first interpretation is completely clear, whereas the second should be viewed as a definition of the linear functional $\langle\psi| = \langle\xi|A$. The corresponding ket-vector $|\psi\rangle$ is related to $|\xi\rangle$ by a linear operator $A^\dagger$ called *Hermitian adjoint* to A. Thus we write $|\psi\rangle = A^\dagger|\xi\rangle$, $\langle\psi| = \langle A^\dagger\xi|$, so the defining property of $A^\dagger$ is

$$\langle A^\dagger\xi|\eta\rangle = \langle\xi|A|\eta\rangle.$$

[3]Note that mathematicians often define the inner product to be conjugate-linear in the *second* argument.

Operators can be specified as matrices relative to the classical basis (or any other orthonormal basis),

$$A = \sum_{j,k} a_{jk}|j\rangle\langle k|, \quad \text{where } a_{jk} = \langle j|A|k\rangle.$$

It is clear that $|j\rangle\langle k|$ is a linear operator: $\big(|j\rangle\langle k|\big)|\xi\rangle = \langle k|\xi\rangle\,|j\rangle$.

The set of linear operators on a space $\mathcal{M}$ is denoted by $\mathbf{L}(\mathcal{M})$. Sometimes we will have to consider linear maps between different spaces, say, from $\mathcal{N}$ to $\mathcal{M}$. The space of such maps is denoted by $\mathbf{L}(\mathcal{N},\mathcal{M})$. It is naturally isomorphic to $\mathcal{M}\otimes\mathcal{N}^*$: the isomorphism takes an operator $\sum a_{jk}|j\rangle\langle k| \in \mathbf{L}(\mathcal{N},\mathcal{M})$ to the vector $\sum a_{jk}\,|j\rangle\otimes\langle k| \in \mathcal{M}\otimes\mathcal{N}^*$.

A *unitary operator* on a space $\mathcal{M}$ is an invertible operator that preserves the inner product. The condition

$$\langle\eta|\xi\rangle = \langle U\eta|U|\xi\rangle = \langle\eta|U^\dagger U|\xi\rangle$$

is equivalent to $U^\dagger U = I$ (where I is the identity operator). Since the space $\mathcal{M}$ has finite dimension, the above condition implies that $|\det U| = 1$, so the existence of U^{-1} follows automatically. Unitary operators are also characterized by the property $U^{-1} = U^\dagger$. The set of unitary operators is denoted by $\mathbf{U}(\mathcal{M})$.

Our definition of the inner product in $\mathcal{B}^{\otimes n}$ is consistent with the tensor product:

$$\big(\langle\xi_1|\otimes\langle\xi_2|\big)\big(|\eta_1\rangle\otimes|\eta_2\rangle\big) = \langle\xi_1|\eta_1\rangle\langle\xi_2|\eta_2\rangle.$$

Later on we will use the *tensor product of operators* (cf. Problem 6.2.) It is an operator acting on the tensor product of the spaces on which the factors act. The action is defined by the rule

$$(A\otimes B)|\xi\rangle\otimes|\eta\rangle = A|\xi\rangle\otimes B|\eta\rangle.$$

If the operators are given in the matrix form relative to some basis, i.e.,

$$A = \sum_{j,k} a_{jk}|j\rangle\langle k|, \quad B = \sum_{j,k} b_{jk}|j\rangle\langle k|,$$

then the matrix elements of the operator $C = A\otimes B$ have the form $c_{(jk)(lm)} = a_{jl}b_{km}$.

[2!] **Problem 6.4.** Let $X : \mathcal{L} \to \mathcal{N}$ be a linear map. Prove that the operators $XX^\dagger$ and $X^\dagger X$ have the same set of nonzero eigenvalues λ_j^2, counted with multiplicities. (The numbers $\lambda_j > 0$ are called the *singular values* of X.) Moreover, the following *singular value decomposition* holds:

$$X = \sum_j \lambda_j\,|\xi_j\rangle\langle\nu_j|, \tag{6.2}$$

where $\{|\xi_j\rangle\}$, $\{|\nu_j\rangle\}$ are orthonormal eigenvector systems for $XX^\dagger$ and $X^\dagger X$, respectively. (There is freedom in the choice of each system, but if one is fixed, the other is determined uniquely.)

6.3. Quantum gates and circuits. Computation consists of transformations, regarded as elementary and performed one at a time.

Elementary transformation **in the classical case:** a map from $\mathbb{B}^n$ to $\mathbb{B}^n$ which alters and depends upon a small number (not depending on n) of bits; the remaining bits are not used.	Elementary transformation **in the quantum case**: the tensor product of an arbitrary unitary operator acting on a small number ($r = O(1)$) of qubits, denoted altogether by $\mathcal{B}^{\otimes r}$, and the identity operator acting on the remaining qubits.

The tensor product of an operator U acting on an ordered set A of qubits and the identity operator acting on the remaining qubits, is denoted by $U[A]$. In this situation, we say that the operator U is *applied to the register* A. This definition is somewhat vague, but the formal construction of the operator $U[A]$ is pretty straightforward.

First, let us define $X[A]$ when A consists of just one qubit, say p. In this case, $X[p] = I_{\mathcal{B}^{\otimes(p-1)}} \otimes X \otimes I_{\mathcal{B}^{\otimes(n-p)}}$. Note that $X[p]$ and $Y[q]$ commute if $p \neq q$. In the general case $A = (p_1, \dots, p_r)$, we can represent U as follows:

$$U = \sum_{j_1,\dots,j_r;\, k_1,\dots,k_r} u_{j_1,\dots,j_r;\, k_1,\dots,k_r} \big(|j_1\rangle\langle k_1|\big) \otimes \cdots \otimes \big(|j_r\rangle\langle k_r|\big).$$

Actually, all we need is a representation of the form

$$U = \sum_m X_{m,1} \otimes \cdots \otimes X_{m,r}, \tag{6.3}$$

where $X_{m,1}, \dots, X_{m,r} \in \mathbf{L}(\mathcal{B})$ are arbitrary one-qubit operators. Then, by definition,

$$U[p_1, \dots, p_r] = \sum_m X_{m,1}[p_1] \cdots X_{m,r}[p_r]. \tag{6.4}$$

The result does not depend on the choice of the representation (6.3) due to the universality property of the tensor product (see p. 55). In the case at hand, we have a multilinear map $F(X_{m,1}, \dots, X_{m,r}) = X_{m,1}[p_1] \cdots X_{m,r}[p_r]$, whereas the corresponding linear map

$$G: \ U \mapsto U[p_1, \dots, p_r]: \ \mathbf{L}(\mathcal{B}^{\otimes r}) \to \mathbf{L}(\mathcal{B}^{\otimes n})$$

is given by (6.4).

Example 6.1. Let $U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$. Then the operators $U[1]$ and $U[2]$, acting on the space $\mathcal{B}^{\otimes 2}$, are represented by these matrices:

$$U[1] = \begin{pmatrix} u_{00} & 0 & u_{01} & 0 \\ 0 & u_{00} & 0 & u_{01} \\ u_{10} & 0 & u_{11} & 0 \\ 0 & u_{10} & 0 & u_{11} \end{pmatrix}, \qquad U[2] = \begin{pmatrix} u_{00} & u_{01} & 0 & 0 \\ u_{10} & u_{11} & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{pmatrix}.$$

The rows and columns are associated with the basis vectors arranged in the lexicographic order: $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$.

[1] **Problem 6.5.**

a) Let $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Write the matrix of the operator $H[2]$ acting on the space $\mathcal{B}^{\otimes 3}$.

b) Let U be an arbitrary two-qubit operator with matrix elements $u_{jk} = \langle j|U|k\rangle$, where $j, k \in \{00, 01, 10, 11\}$. Write the matrix for $U[3, 1]$.

At this point the computational complexity begins, which makes quantum computers so powerful. Let U act on two qubits, i.e., U is a 4×4 matrix. Then $U[1, 2] = U \otimes I$ is a matrix of size $2^n \times 2^n$ that consists of 2^{n-2} copies of U placed along the principal diagonal. This matrix represents one elementary step. When we apply several such operators to various pairs of qubits, the result will appear considerably more complicated. There is no obvious way of determining this result, apart from direct multiplication of the corresponding matrices. Inasmuch as the size of the matrices is exponentially large, exponential time is required for their multiplication.

We remark, however, that the calculation of matrix elements is possible with polynomially bounded memory. Suppose we need to find the matrix element U_{xy} of the operator

$$U = U^{(l)}[j_l, k_l]U^{(l-1)}[j_{l-1}, k_{l-1}] \cdots U^{(2)}[j_2, k_2]U^{(1)}[j_1, k_1].$$

It is obvious that

$$\left(U^{(l)} \cdots U^{(1)}\right)_{x_l x_0} = \sum_{x_{l-1}, \ldots, x_1} U^{(l)}_{x_l x_{l-1}} \cdots U^{(1)}_{x_1 x_0}.$$

(Here $x_0, \ldots, x_l$ are n-bit strings.) To compute this sum, it suffices to allocate $l-1$ registers for keeping the current values of $x_{l-1}, \ldots, x_1$, one register for keeping the partial sum, and some constant number of registers for the calculation of the product $U^{(l)}_{x_l x_{l-1}} \cdots U^{(1)}_{x_1 x_0}$.

Definition 6.1 (*Quantum circuit*). Let $\mathcal{A}$ be a fixed set of unitary operators. (We call $\mathcal{A}$ a *basis*, or a gate set, whereas its elements are called

gates.) A *quantum circuit over the basis* $\mathcal{A}$ is a sequence $U_1[A_1], \ldots, U_L[A_L]$, where $U_j \in \mathcal{A}$, and A_j is an ordered set of qubits.

The *operator realized by the circuit* is $U = U_L[A_L] \cdots U_1[A_1]$ ($U : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes n}$). The number L is called the *size* of the circuit.

We usually assume that $\mathcal{A}$ is closed under inversion: if $X \in \mathcal{A}$, then $X^{-1} \in \mathcal{A}$. In this case U and U^{-1} are realized by circuits of the same size.

Note that several gates, say $U_{j_1}, \ldots, U_{j_s}$, can be applied simultaneously to disjoint sets of qubits (such that $A_{j_a} \cap A_{j_b} = \emptyset$ if $a \neq b$). We say that a circuit has *depth* $\leq d$ if it can be arranged in d layers of simultaneously applied gates. The depth can be also characterized as the maximum length of a path from input to output. (By a path we mean a sequence of gates $U_{k_1}, \ldots, U_{k_d}$ ($k_1 < \cdots < k_d$) such that each pair of adjacent gates, k_l and k_{l+1}, share a qubit they act upon, but no other gate acts on this qubit between the applications of U_{k_l} and $U_{k_{l+1}}$.)

Definition 6.1 is not perfect because it ignores the possibility to use additional qubits (*ancillas*) in the computational process. Therefore we give yet another definition.

(*Operator realized by a quantum circuit using ancillas*). This is an operator $U : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes n}$ such that the product

$$W = U_L[A_L] \cdots U_1[A_1],$$

acting on N qubits ($N \geq n$), satisfies the condition $W\left(|\xi\rangle \otimes |0^{N-n}\rangle\right) = (U|\xi\rangle) \otimes |0^{N-n}\rangle$ for any vector $|\xi\rangle \in \mathcal{B}^{\otimes n}$.

In this manner we "borrow" additional memory, filled with zeros, that we must ultimately return to its prior state. What sense does such a definition make? Why is it necessary to insist that the additional qubits return to the state $|0^{N-n}\rangle$? Actually, this condition is rather technical. However, it is important that at the end of the computation the quantum state is a *product state*, i.e., has the form $|\xi'\rangle \otimes |\eta'\rangle$ (with arbitrary $|\eta'\rangle$). If this is the case, then the first subsystem will be in the specified state $|\xi'\rangle$, so that the second subsystem (the added memory) may be forgotten. In the opposite case, the joint state of the two subsystems will be *entangled*, so that the first subsystem cannot be separated from the second.

7. Correspondence between classical and quantum computation

Quantum computation is supposed to be more general than classical computation. However, quantum circuits do not include Boolean circuits as a special case. Therefore some work is required to specialize the definition

of a quantum circuit and prove that the resulting computational model is equivalent to the Boolean circuit model.

The classical analogue of a unitary operator is an invertible map on a finite set, i.e., a permutation. An arbitrary permutation $G : \mathbb{B}^k \to \mathbb{B}^k$ corresponds naturally to a unitary operator $\widehat{G}$ on the space $\mathcal{B}^{\otimes k}$ acting according to the rule $\widehat{G}|x\rangle \stackrel{\text{def}}{=} |Gx\rangle$.

By analogy with Definition 6.1, we may define reversible classical circuits, which realize permutations.

Definition 7.1 (*Reversible classical circuit*). Let $\mathcal{A}$ be a set of permutations of the form $G : \mathbb{B}^k \to \mathbb{B}^k$. (The set $\mathcal{A}$ is called a basis; its elements are called gates.) A *reversible classical circuit over the basis* $\mathcal{A}$ is a sequence of permutations $G_1[A_1], \ldots, G_l[A_L]$, where A_j is a set of bits and $G_j \in \mathcal{A}$.

(*Permutation realized by a reversible circuit*). This is the product of permutations $G_l[A_l] \cdots G_1[A_1]$.

(*Permutation realized by a reversible circuit using ancillas*). This is a permutation G such that the product of permutations

$$W = G_l[A_l] \cdots G_1[A_1]$$

(acting on N bits, $N \geq n$) satisfies the condition $W(x, 0^{N-n}) = (Gx, 0^{N-n})$ for arbitrary $x \in B^n$.

In what cases a function given by a Boolean circuit can be realized by a reversible circuit? Reversible circuits realize only permutations, i.e., invertible functions. This difficulty can be overcome in this way: instead of computing a general Boolean function $F : \mathbb{B}^n \to \mathbb{B}^m$, we compute the permutation $F_\oplus : \mathbb{B}^{n+m} \to \mathbb{B}^{n+m}$ given by the formula $F_\oplus(x, y) = (x, y \oplus F(x))$ (here $\oplus$ denotes bitwise addition modulo 2). Then $F_\oplus(x, 0) = (x, F(x))$ contains the value of $F(x)$ we need.

Note that two-bit permutation gates do not allow to realize all functions of the form $F_\oplus$. It turns out that any permutation on two-bit states, $g : \mathbb{B}^2 \to \mathbb{B}^2$, is a linear function (under the natural identification of the set $\mathbb{B}$ with the two-element field $\mathbb{F}_2$): $g(x, y) = (ax \oplus by \oplus c, dx \oplus ey \oplus f)$, where $a, b, c, d, e, f \in \mathbb{F}_2$. Therefore all functions realized by reversible circuits over the basis of permutations on two bits, are linear.

However, permutations on three bits already suffice to realize any permutation. In fact, the following two functions form a complete basis for reversible circuits: negation $\neg$ and the *Toffoli gate*, $\bigwedge_\oplus : (x, y, z) \longmapsto (x, y, z \oplus xy)$. Here we mean realization using ancillas, i.e., it is allowed to borrow bits in the state 0 under the condition that they return to the same state after the computation is done.

Lemma 7.1. *Let a function $F : \mathbb{B}^n \to \mathbb{B}^m$ be realized by a Boolean circuit of size L and depth d over some basis $\mathcal{A}$ (the fan-in and fan-out being bounded by a constant). Then we can realize a map of the form $(x, 0) \longmapsto (F(x), G(x))$ by a reversible circuit of size $O(L)$ and depth $O(d)$ over the basis $\mathcal{A}_\oplus$ consisting of the functions $f_\oplus$ ($f \in \mathcal{A}$) and the function $\circledoplus : (x, y) \longmapsto (x, x \oplus y)$.*

Remark 7.1. In addition to the "useful" result $F(x)$, the indicated map produces some "garbage" $G(x)$.

Remark 7.2. The gate $\circledoplus$ is usually called "Controlled NOT" for reasons that will become clear later. Note that $\circledoplus = I_\oplus$, where I is the identity map on a single bit. The essential meaning of the operation $\circledoplus$ is reversible copying of the bit x (if the initial value of y is 0).

Remark 7.3. The gate $\circledoplus$ allows one to interchange bits in memory, since the function $(\leftrightarrow) : (a, b) \longmapsto (b, a)$ can be represented as follows:

$$(\leftrightarrow)[j, k] = \circledoplus[j, k] \circledoplus [k, j] \circledoplus [j, k].$$

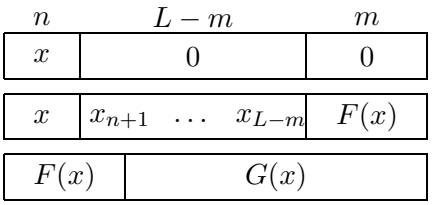
Proof of Lemma 7.1. Consider the Boolean circuit that computes F. Let the input variables be $x_1, \ldots, x_n$, and the auxiliary variables (including the result bits) $x_{n+1}, \ldots, x_{n+L}$. A reversible circuit we are to construct will also have $n + L$ bits; the bits $x_{n+1}, \ldots, x_{n+L}$ are initialized by 0.

Each assignment in the original (Boolean) circuit has the form $x_{n+k} := f_k(x_{j_k}, \ldots, x_{l_k})$, $f_k \in \mathcal{A}$, $j_k, \ldots, l_k < n + k$. In the corresponding reversible circuit, the analogue of this assignment will be the action of the permutation $(f_k)_\oplus$, i.e., $x_{n+k} := x_{n+k} \oplus f_k(x_{j_k}, \ldots, x_{l_k})$.

Since the initial values of the auxiliary variables were equal to 0, their final values will be just as in the original circuit. In order to obtain the required form of the result, it remains to change positions of the bits.

In this argument, we may assume that the original circuit has a layered structure, so that several assignments can occur simultaneously. However, the concurrent assignments *should not share their input variables*. If this is not the case, we need to insert explicit copy gates between the layers; each copy gate will be replaced by $\circledoplus$ in the reversible circuit. This results in depth increase by at most constant factor, due to the bounded fan-out condition.

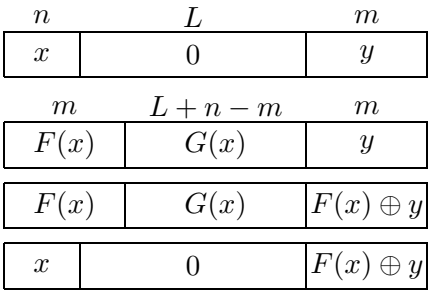
The entire computational process is conveniently represented by the following diagram (above the rectangles is written the number of bits and, inside, their content).



□

Lemma 7.2 (Garbage removal). *Under the conditions of Lemma* 7.1*, one can realize the function* $F_\oplus$ *by a reversible circuit of size* $O(L+n+m)$ *and depth* $O(d)$ *using ancillas.*

Proof. We perform the computation from the proof of Lemma 7.1, add each bit of the result to the corresponding bit of y with $\oplus$, and undo the above computation.



□

Remark 7.4. Reversible computation provides an answer to the following question: how much energy is required to compute a given Boolean function [**10, 45**]? Theoretically, reversible operations can be performed at no energy cost. On the other hand, irreversible operations, like bit erasure, pose a fundamental problem. When such an operation is performed, two different logical states (0 and 1) become identical (0). However, physical laws on a micro-scale are reversible. The solution to this apparent paradox is that the difference between the initial states, 0 and 1, is converted into a difference between two physical states that both correspond to the same logical value 0. This may be interpreted as an increase in disorder (entropy) in physical degrees of freedom beyond our control, which eventually appears in the surrounding environment in the form of heat. The amount of energy required to erase a single bit is very small ($kT \ln 2$), but still nonzero. The theoretical energy cost of information erasure on a hard disk of capacity 1 gigabyte is equal to $2.4 \cdot 10^{-11}$ Joules, which corresponds to the energy spent in moving the disk head by a fraction of the size of an atom. This is many orders of magnitude smaller than the actual displacement of the head through formatting.

On the other hand, if the capacity of disks were to continue growing as fast as now, then at the end of the twenty-third century formatting of a hard disk would require as much energy as the Sun generates in a year.

The garbage removal lemma shows that it is possible to avoid such losses of energy connected with irreversible operations.

It is likewise possible to show that arbitrary computation performed with memory s, can be realized in a reversible manner through the use of memory not exceeding $s^{O(1)}$. We will give a sketch of the proof. However, the reader should keep in mind that computation with bounded space is not easily defined in terms of circuits. Indeed, if a circuit is allowed to be exponentially large (though of polynomial "width"), it can contain the value table of the desired function, which makes its computation trivial. Therefore, a rigorous proof should either deal with circuits of some regular structure, or involve a definition of a reversible Turing machine.

An arbitrary computation with a given memory s can be reduced to solving a poly(s)-size instance of TQBF, since TQBF is PSPACE-complete. We will show how to compute reversibly, with a small amount of memory, the value of the formula

$$\exists x_1 \forall y_1 \cdots \exists x_M \forall y_M f(x_1, y_1, \ldots, x_M, y_M, z), \tag{7.1}$$

where $f(\cdot)$ is computed by a Boolean circuit of size L. Actually, in this case the value of the formula (7.1) can be represented by a reversible circuit with $O(L + M)$ bits. The computation will be organized recursively, beginning with the innermost quantifiers.

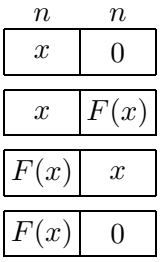
In order to compute $\forall x\, F(x, z)$, we compute $F(0, z)$ and put the result into a supplementary bit. Then we compute $F(1, z)$ and put the result into another bit. Next we compute $\forall x\ F(x, z) = F(0, z) \wedge F(1, z)$ and save the result in a third bit. In order to remove the garbage, we undo all calculations, except for the final step.

Dealing with the formula $\exists x\, F(x, y)$ in a similar manner, we arrive at the following result: adding a quantifier in one Boolean variable increases the required memory by at most a constant number of bits.

In conclusion, we formulate a theorem on computation of reversible functions, which is a direct generalization of Lemma 7.2.

Theorem 7.3. *Let F and F^{-1} be computed by Boolean circuits of size $\leq L$ and depth $\leq d$. Then F can be realized by a reversible circuit of size $O(L+n)$ and depth $O(d)$ using ancillas.*

Proof. The computation is performed according to the following scheme (for simplicity we do not show the ancillas that are used in the computation from Lemma 7.2).



— computation of $F_\oplus$ by the circuit from the proof of Lemma 7.2

— permutation of bits

— applying $(F^{-1})_\oplus$ (by the circuit from the proof of Lemma 7.2) yields $x \oplus F^{-1}(F(x)) = 0$ in the right register

□

[1!] **Problem 7.1.** Prove that negation and the Toffoli gate form a complete basis for reversible circuits.

8. Bases for quantum circuits

How do we choose a basis (gate set) for computation by quantum circuits? There are uncountably many unitary operators. So, either a complete basis must contain an infinite (uncountable) number of gates, or else we have to weaken the condition of exact realization of an operator by a circuit, changing it to a condition of approximate realization. We will examine both possibilities.

8.1. Exact realization.

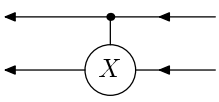
Theorem 8.1. *The basis consisting of all one-qubit and two-qubit unitary operators allows the realization of an arbitrary unitary operator.*

The rest of the section constitutes a proof of this theorem.

8.1.1. *Operators with quantum control.*

Definition 8.1. For each operator $U : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes n}$, an operator $\Lambda(U) : \mathcal{B} \otimes \mathcal{B}^{\otimes n} \to \mathcal{B} \otimes \mathcal{B}^{\otimes n}$ ("*controlled U*") is defined by the following relations:

$$\begin{aligned} \Lambda(U)|0\rangle \otimes |\xi\rangle &= |0\rangle \otimes |\xi\rangle, \\ \Lambda(U)|1\rangle \otimes |\xi\rangle &= |1\rangle \otimes U|\xi\rangle. \end{aligned} \tag{8.1}$$



Graphically, we represent an operator $\Lambda(X)$ as shown in the figure. The top line corresponds to the control qubit (the first tensor factor in (8.1)) while the bottom line represents the other qubits. The direction of the arrows corresponds to the order in which operators act on an input vector. For example, in Figure 8.1 (see below) the first operator is $\Lambda(Y^{-1})$. In this book, we draw arrows from right to left, which is consistent with the convention that $AB|\xi\rangle$ means "take $|\xi\rangle$, apply B, then apply A".

We will also need operators with several controlling qubits:

$$\Lambda^k(U)|x_1,\dots,x_k\rangle\otimes|\xi\rangle=\begin{cases}|x_1,\dots,x_k\rangle\otimes|\xi\rangle & \text{if } x_1\cdots x_k=0,\\ |x_1,\dots,x_k\rangle\otimes U|\xi\rangle & \text{if } x_1\cdots x_k=1.\end{cases} \tag{8.2}$$

Example 8.1. Let $\sigma^x \stackrel{\text{def}}{=} \neg = \begin{pmatrix} 0 & 1\\ 1 & 0\end{pmatrix}$. Then $\Lambda(\sigma^x)=\widehat{\oplus}$, and $\Lambda^2(\sigma^x)=\widehat{\Lambda_\oplus}$ (the Toffoli gate).

8.1.2. ***The realization of the Toffoli gate.*** Now we construct the Toffoli gate using transformations on two qubits. To start, we find a pair of operators that satisfy the relation $XYX^{-1}Y^{-1}=i\sigma^x$. For example, the following pair will do:

$$X=\frac{1}{\sqrt{2}}\begin{pmatrix}-i & -1\\ 1 & i\end{pmatrix},\quad Y=\begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix}. \tag{8.3}$$

Let us clarify the geometric meaning of this construction. The unitary group $\mathbf{U}(2)$ acts on three-dimensional Euclidean space. To define this action, we note that 2×2 Hermitian matrices with zero trace form a three-dimensional Euclidean space: the inner product between A and B is given by $\frac{1}{2}\operatorname{Tr}(AB)$ and an orthonormal basis is formed by the *Pauli matrices*

$$\sigma^x=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix},\quad \sigma^y=\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix},\quad \sigma^z=\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}. \tag{8.4}$$

A unitary operator $U\in\mathbf{U}(2)$ acts on this space by this rule: $U: E\mapsto UEU^{-1}$. It is possible to show (see [**44**, §11.12]) that the action we have just defined yields an isomorphism $\mathbf{U}(2)/\mathbf{U}(1)\cong\mathbf{SO}(3)$, where $\mathbf{U}(1)=\{c\in\mathbb{C}: |c|=1\}$ is the subgroup of phase shifts, and $\mathbf{SO}(3)$ is the group of rotations of three-dimensional space (i.e., the group of orthogonal transformations with determinant 1).

Under this action, σ^x corresponds to a rotation about the x axis by 180°, X to a rotation about the vector $(0,1,1)$ by 180°, and Y to a rotation about the y axis by 180°.

Shown in Figure 8.1 is a circuit that realizes the Toffoli gate by using the operators $\Lambda(X)$, $\Lambda(Y)$ and $\Lambda^2(-i)$. The last of these is a phase shift (multiplication by $-i$) controlled by two bits.

Let us test this circuit.

Suppose the input vector is $|a,b,c\rangle=|a\rangle\otimes|b\rangle\otimes|c\rangle$, where $a,b,c\in\mathbb{B}$. If $a=b=1$, then the operator $-iXYX^{-1}Y^{-1}=\sigma^x$ is applied to $|c\rangle$, which changes $|0\rangle$ to $|1\rangle$ and vice versa. However, if at least one of the controlling bits is 0, then $|c\rangle$ is multiplied by the identity operator. This is exactly how the Toffoli gate acts on basis vectors. This action extends to the whole space $\mathcal{B}^{\otimes 3}$ by linearity.

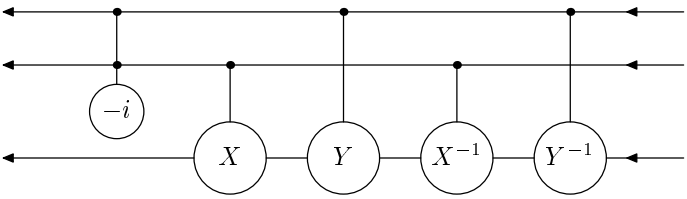


Fig. 8.1. Implementation of the Toffoli gate.

8.1.3. ***The realization of*** $\Lambda^k(U)$ ***for*** $U \in \mathbf{U}(\mathcal{B})$**.** Let U be a unitary operator acting on one qubit. We will show how to realize the operator $\Lambda^k(U)$ for arbitrary k by acting only on pairs of qubits. Our first solution uses ancillas. We actually construct an operator W which acts on the space of $N+1$ qubits $\mathcal{B}^{\otimes(N+1)}$ and satisfies the condition

$$W\big(|\eta\rangle \otimes |0^{N-k}\rangle\big) = \Lambda(U)|\eta\rangle \otimes |0^{N-k}\rangle.$$

(Caution: this condition **does not mean** that $W = \Lambda(U) \otimes I$.)

There exists a reversible circuit P of size $O(k)$ and depth $O(\log k)$ that computes the product of k input bits $x_1 \cdots x_k$ (the result being a single bit), and also produces some garbage $G(x_1, \ldots, x_k)$ ($N-1$ bits). It is represented graphically in Figure 8.2 (written above each box is the number of bits in the corresponding memory segment).

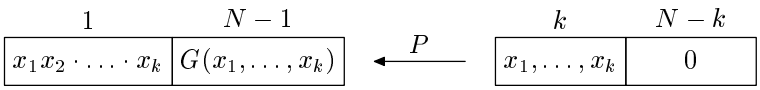


Fig. 8.2

Figure 8.3 shows how to construct the operator W using the circuit P and an operator with one controlling qubit. The circuit P is applied first, followed by the reverse circuit P^{-1}, so that all N bits return to their initial state. In the meantime, the first bit (the top line in Figure 8.3) takes the value $x_1 \cdots x_k$. It is used as the control qubit for $\Lambda(U)$, whereas qubit $N+1$ is the target. The circuit in the figure can also be described by the equation $W = P^{-1}\Lambda(U)P$ or, more explicitly,

$$W[\underbrace{1,\ldots,k,N{+}1}_{\Lambda(U)}, \underbrace{k{+}1,\ldots,N}_{\text{ancillas}}] = P^{-1}[1,\ldots,N]\ \Lambda(U)[1,N{+}1]\ P[1,\ldots,N].$$

The use of ancillas can be avoided at the cost of an increase in the circuit size. Let us consider the operator $\Lambda^k(i\sigma^x)$ first. A circuit C_k for the realization of this operator can be constructed recursively: it consists of two copies of the circuit $C_{\lceil k/2 \rceil}$, two copies of the circuit $C_{\lfloor k/2 \rfloor}$, and a constant

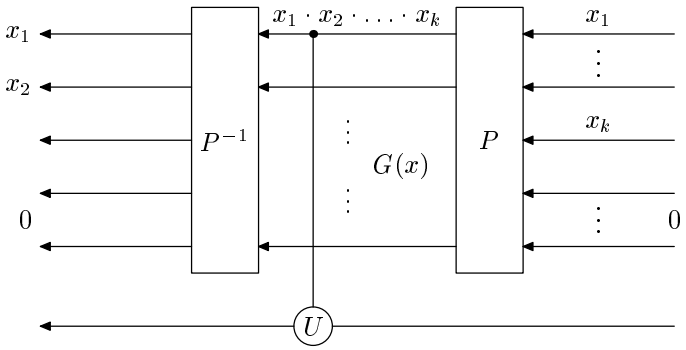


Fig. 8.3. Implementation of the operator $\Lambda^k(U)$ using ancillas.

number c of one-qubit gates. Therefore we get a recurrence relation for the circuit size, $L_k = 2L_{\lfloor k/2 \rfloor} + 2L_{\lceil k/2 \rceil} + c$, so that $L_k = O(k^2)$. The concrete construction is shown in Figure 8.4 (cf. Figure 8.1). We again use the operators X and Y (see (8.3)) satisfying $XYX^{-1}Y^{-1} = i\sigma^x$. Now we apply them with multiple control qubits: Y is controlled by the qubits $1, \ldots, \lceil k/2 \rceil$, whereas X is controlled by the qubits $\lceil k/2 \rceil + 1, \ldots, k$. It remains to notice that X and Y are conjugate to $i\sigma^x$, i.e., $X = V(i\sigma^x)V^{-1}$, $Y = W(i\sigma^x)W^{-1}$ for some unitary V and W. Hence $\Lambda^b(X)$ and $\Lambda^a(Y)$ (where $a = \lceil k/2 \rceil$, $b = \lfloor k/2 \rfloor$) can be obtained if we conjugate $\Lambda^b(i\sigma^x)$ and $\Lambda^a(i\sigma^x)$ by V and W (resp.) applied on the last qubit.

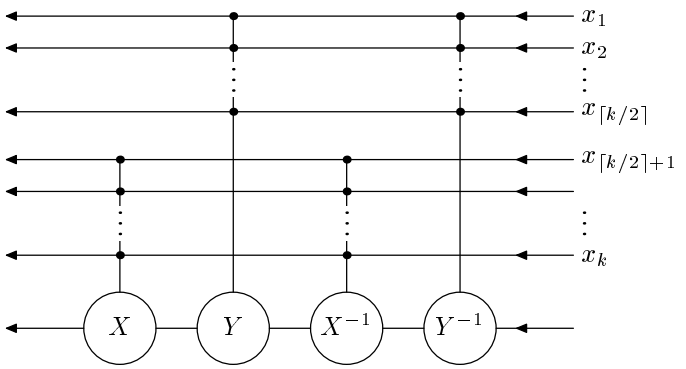


Fig. 8.4. Implementation of the operator $\Lambda^k(i\sigma^x)$ without ancillas.

The operator $\Lambda^k(Z)$ for an arbitrary $Z \in \mathbf{SU}(2)$ can be realized by two applications of $\Lambda^k(\sigma^x)$ and four applications of one-qubit gates, as in the solution to Problem 8.2 (see Figure S8.1a). Note that one copy of $\Lambda^k(\sigma^x)$ can be replaced by $\Lambda^k(i\sigma^x)$, and the other by $\Lambda^k(-i\sigma^x)$.

Consider now the general case, $\Lambda^k(U)$, where $U \in \mathbf{U}(2)$. Let $U = U_0 = e^{i\varphi_1} Z_0$, where $Z_0 \in \mathbf{SU}(2)$. Then $\Lambda^k(e^{i\varphi_1}) = \Lambda^{k-1}(U_1)$, where $U_1 =$

$\Lambda(e^{i\varphi_1}) \in \mathbf{U}(2)$. Thus we have

$$\Lambda^k(U)[1,\dots,k,k{+}1] = \Lambda^{k-1}(U_1)[1,\dots,k]\ \Lambda^k(Z_0)[1,\dots,k,k{+}1].$$

We proceed by induction, obtaining the equation

$$\begin{aligned}\Lambda^k(U)&[1,\dots,k,k{+}1]\\ &= U_k[1]\ \Lambda^1(Z_{k-1})[1,2]\ \cdots\ \Lambda^k(Z_0)[1,\dots,k,k{+}1].\end{aligned} \tag{8.5}$$

It is represented graphically in Figure 8.5. The size of the resulting circuit is $O(n^3)$. (This construction can be made more efficient; see Problem 8.6.)

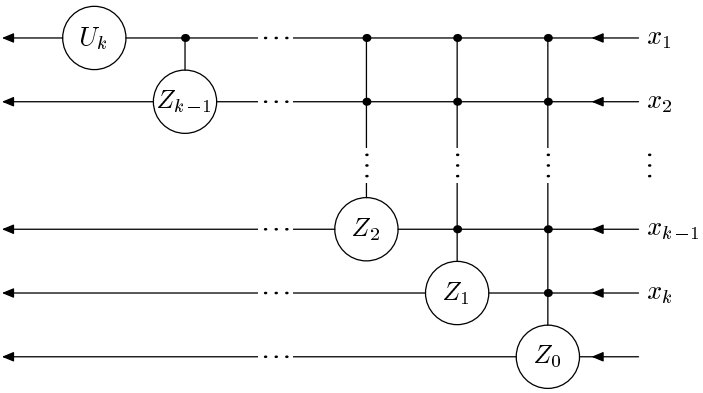


Fig. 8.5. Ancilla-free realization of $\Lambda^k(U)$, $U \in \mathbf{U}(2)$.

8.1.4. ***The realization on an arbitrary operator.*** We continue the proof of Theorem 8.1. The action of $\Lambda^k(U)$ may be described as follows: the operator U acts on the subspace generated by the vectors $|1,\dots,1,0\rangle$ and $|1,\dots,1,1\rangle$, and the identity operator acts on the orthogonal complement of this subspace. Our next task is to realize a similar operator in which a nontrivial action is carried out on the subspace spanned by an arbitrary pair of basis vectors. Suppose we want to realize an arbitrary operator on the subspace spanned by $|x\rangle$ and $|y\rangle$, where $x = (x_1,\dots,x_n)$, $y = (y_1,\dots,y_n)$, $x_j, y_j \in \mathbb{B}$. Let f be a permutation such that $f(x) = (1,\dots,1,0)$, $f(y) = (1,\dots,1,1)$. We may assume that f is linear, i.e., $f : x \mapsto Ax + b$, where A is an invertible matrix, and b is a vector over the two-element field $\mathbb{F}_2$. Such permutations can be realized by reversible circuits over the basis $\{\neg, \oplus\!\!\!\bigcirc\}$ without ancillas. Then the operator we need is represented in the form $\widehat{f}^{-1}\Lambda^{n-1}(U)\widehat{f}$. (Recall that $\widehat{f}$ is the operator corresponding to the permutation f.)

Therefore we can act arbitrarily on pairs of basis vectors. Since we only used circuits of size $\mathrm{poly}(n)$, the constructed actions are realized efficiently. The final part in the proof of Theorem 8.1 is not efficient. Now we forget about qubits (i.e., the tensor product structure of our space), so we just have

a Hilbert space of dimension $M = 2^n$. We want to represent an arbitrary unitary operator U by the actions on pairs of basis vectors. This will be polynomial in M, hence exponential in n.

Lemma 8.2. *An arbitrary unitary operator U on the space $\mathbb{C}^M$ can be represented as a product of $M(M-1)/2$ matrices of the form*

$$\begin{pmatrix} 1 & 0 & \dots & & & \dots \\ \vdots & \ddots & 0 & & & \dots \\ 0 & \dots & 1 & 0 & & \dots \\ 0 & \dots & & \begin{pmatrix} a & b \\ c & d \end{pmatrix} & 0 & \dots \\ 0 & \dots & & & 1 & 0 \\ \dots & & & & \ddots & 0 \\ 0 & \dots & & & & 1 \end{pmatrix}, \text{ where } \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbf{U}(2). \tag{8.6}$$

Proof. First we note that for any numbers c_1, c_2 there exists a 2×2 unitary matrix V such that

$$V \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} \sqrt{|c_1|^2 + |c_2|^2} \\ 0 \end{pmatrix}.$$

Consequently, for a unit vector $|\xi\rangle \in \mathbb{C}^M$ there exists a sequence of unitary operators $V^{(1)}, \dots, V^{(M-1)}$ such that $V^{(1)} \cdots V^{(M-1)}|\xi\rangle = |1\rangle$, where $V^{(s)}$ acts on the subspace $\mathbb{C}(|s\rangle, |s+1\rangle)$ (as the matrix (8.6)) and leaves the remaining basis vectors unchanged.

Now let an $M \times M$ unitary matrix U be given. Multiplying U^{-1} on the left by suitable matrices $U^{(1,1)}, \dots, U^{(1,M-1)}$, we can transform the first column into the vector $|1\rangle$. Since the columns remain orthogonal, the first row becomes $\langle 1|$. Acting in the same way on the remaining columns, we obtain a set of matrices $U^{(j,s)}$, $1 \le j \le s \le M-1$, (where $U^{(j,s)}$ acts on $|s\rangle$ and $|s+1\rangle$) satisfying the condition

$$U^{(M-1,M-1)} \left(U^{(M-2,M-2)} U^{(M-2,M-1)}\right) \cdots \left(U^{(1,1)} \cdots U^{(1,M-1)}\right) U^{-1} = I.$$

This proof is constructive, i.e., it provides an algorithm for finding the matrices $U^{(j,s)}$. The running time of this algorithm depends on M and another parameter δ, the precision of arithmetic operations with real numbers. Specifically, the algorithm complexity is $O(M^3) \cdot \mathrm{poly}(\log(1/\delta))$. □

Problems

[2!] **8.1.** Prove that any operator $U \in \mathbf{U}(\mathcal{B})$ can be realized (without ancillas) by a constant size circuit over the basis $\{\Lambda(e^{i\varphi}) : \varphi \in \mathbb{R}\} \cup \{H\}$.

[1!] **8.2.** Prove that any operator of the form $\Lambda(U)$, $U \in \mathbf{U}(\mathcal{B})$ can be realized (without ancillas) by a constant size circuit over the basis of one-qubit gates and the gate $\Lambda(\sigma^x)$.

(Therefore, this basis allows the realization of an arbitrary unitary operator. Indeed, in the proof of Theorem 8.1 we only use gates of the form $\Lambda(U)$ and one-qubit gates.)

[2!] **8.3.** Suppose that a basis $\mathcal{A}$ is closed under inversion and allows the realization of any one-qubit operator up to a phase factor (e.g., $\mathcal{A} = \mathbf{SU}(2)$). Prove that the multiplication by a phase factor can be realized over $\mathcal{A}$ using one ancilla.

[2!] **8.4.** Suppose that a unitary operator $U : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes n}$ satisfies the condition $U|0\rangle = |0\rangle$. Construct a circuit of size $6n + 1$ realizing $\Lambda(U)$ over the basis $\{U, \Lambda^2(\sigma^x)\}$, using ancillas. (The gate U should be applied only once.)

[3] **8.5.** Realize the operator $\Lambda^k(\sigma^x)$ $(k \geq 2)$ by a circuit of size $O(k)$ consisting of Toffoli gates. It is allowed to use a constant number of ancillas in such a way that their initial state does not matter, i.e., the circuit should actually realize the operator $W = \Lambda^k(\sigma^x) \otimes I_{\mathcal{B}^{\otimes r}}$, where $r = \text{const}$.

[2] **8.6.** Realize the operator $\Lambda^k(U)$ by a circuit of size $O(k^2)$ over the basis of all two-qubit gates. The use of ancillas is not allowed.

8.2. Approximate realization. We now pass to finite bases. In this case it is only possible to obtain an approximate representation of operators as products of basis elements. In order to define the approximate realization, we need a norm on the operator space.

On the space of vectors there is the Euclidean norm $\big\||\xi\rangle\big\| = \sqrt{\langle\xi|\xi\rangle}$. By the definition of a norm, it satisfies the following conditions:

$$\big\||\xi\rangle\big\| \begin{cases} = 0 & \text{if } |\xi\rangle = 0, \\ > 0 & \text{if } |\xi\rangle \neq 0, \end{cases} \tag{8.7}$$

$$\big\||\xi\rangle + |\eta\rangle\big\| \leq \big\||\xi\rangle\big\| + \big\||\eta\rangle\big\|, \tag{8.8}$$

$$\big\|c|\xi\rangle\big\| = |c|\,\big\||\xi\rangle\big\|. \tag{8.9}$$

We now introduce a norm on the space of operators $\mathbf{L}(\mathcal{N})$.

Definition 8.2. The norm of an operator X (the so-called *operator norm*; in general, there are others) is

$$\big\|X\big\| = \sup_{|\xi\rangle \neq 0} \frac{\big\|X|\xi\rangle\big\|}{\big\||\xi\rangle\big\|}.$$

We note that $\|X\|^2$ is the largest eigenvalue of the operator $X^\dagger X$.

This norm possesses all the properties of norms indicated above and, beyond these, several special properties:

$$\|XY\| \le \|X\|\,\|Y\|, \tag{8.10}$$

$$\|X^\dagger\| = \|X\|, \tag{8.11}$$

$$\|X \otimes Y\| = \|X\|\,\|Y\|, \tag{8.12}$$

$$\|U\| = 1 \qquad \text{if } U \text{ is unitary.} \tag{8.13}$$

Now we give the definition of approximate realization. If the operator in question is U, then its approximate realization will be denoted by $\tilde{U}$.

Definition 8.3. The operator $\tilde{U}$ *approximates the operator* U *with precision* δ if

$$\|\tilde{U} - U\| \le \delta. \tag{8.14}$$

This definition has two noteworthy corollaries. First, if $\tilde{U}$ approximates U with precision δ, then $\tilde{U}^{-1}$ approximates U^{-1} with the same precision δ. Indeed, if we multiply the expression $\tilde{U} - U$ by $\tilde{U}^{-1}$ on the left and by U^{-1} on the right, the norm does not increase (due to the properties (8.10) and (8.13)). Thus we obtain a corollary of the inequality (8.14): $\|U^{-1} - \tilde{U}^{-1}\| \le \delta$.

The second property is as follows. Consider the product of several operators, $U = U_L \cdots U_2 U_1$. If each U_k has an approximation $\tilde{U}_k$ with precision δ_k, then the product of these approximations, $\tilde{U} = \tilde{U}_L \cdots \tilde{U}_2 \tilde{U}_1$, approximates U with precision $\sum \delta_k$ (i.e., **errors accumulate linearly**):

$$\left\|\tilde{U}_L \cdots \tilde{U}_2 \tilde{U}_1 - U_L \cdots U_2 U_1\right\| \le \sum_j \delta_j.$$

It suffices to look at the example with two operators:

$$\begin{aligned}
\left\|\tilde{U}_2\tilde{U}_1 - U_2U_1\right\| &= \left\|\tilde{U}_2(\tilde{U}_1 - U_1) + (\tilde{U}_2 - U_2)U_1\right\| \\
&\le \left\|\tilde{U}_2(\tilde{U}_1 - U_1)\right\| + \left\|(\tilde{U}_2 - U_2)U_1\right\| \\
&\le \left\|\tilde{U}_2\right\|\left\|\tilde{U}_1 - U_1\right\| + \left\|\tilde{U}_2 - U_2\right\|\left\|U_1\right\| \\
&= \left\|\tilde{U}_1 - U_1\right\| + \left\|\tilde{U}_2 - U_2\right\|.
\end{aligned}$$

Note that we have used the fact the the norm of a unitary operator is 1. (With nonunitary operators, the approximation errors could accumulate much faster, e.g., exponentially.)

Remark 8.1. Every model that aims at solving computational problems by real physical processes, has to be scrutinized for stability to approximation errors. (In real life the parameters of any physical process can be given only

with certain precision.) In particular, computation with exponential error accumulation is almost definitely useless from the practical point of view.

Now we generalize Definition 8.3 to allow the use of ancillas.

Definition 8.4. The operator $U : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes n}$ *is approximated by the operator* $\tilde{U} : \mathcal{B}^{\otimes N} \to \mathcal{B}^{\otimes N}$ *with precision* δ *using ancillas* if, for arbitrary $|\xi\rangle$ in $\mathcal{B}^{\otimes n}$, the inequality

$$\left\|\tilde{U}\left(|\xi\rangle \otimes |0^{N-n}\rangle\right) - U|\xi\rangle \otimes |0^{N-n}\rangle\right\| \leq \delta \||\xi\rangle\| \tag{8.15}$$

is satisfied.

We can formulate this definition in another way. Let us introduce a linear map $V : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes N}$ which acts by the rule $V : |\xi\rangle \mapsto |\xi\rangle \otimes |0^{N-n}\rangle$. The map V is not unitary, but isometric, i.e., $V^\dagger V = I_{\mathcal{B}^{\otimes n}}$. The condition from the last definition may be written as

$$\left\|\tilde{U}V - VU\right\| \leq \delta. \tag{8.16}$$

The basic properties of approximation remain true for approximation using ancillas (which, of course, should be verified; see Problem 8.8).

What bases allow the realization of an arbitrary unitary operator with arbitrary precision? What is the size of the circuit that is needed to achieve a given precision δ? How to construct this circuit efficiently? Unfortunately, we cannot give a universal answer to these questions. In constructing quantum algorithms, we will use the following (widely adopted) *standard basis*.

Definition 8.5. The basis $\mathcal{Q} = \{H, K, K^{-1}, \Lambda(\sigma^x), \Lambda^2(\sigma^x)\}$, where

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad K = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix},$$

is called *standard*.

Theorem 8.3. *Any unitary operator U on a fixed number of qubits can be realized with precision δ by a* $\mathrm{poly}(\log(1/\delta))$*-size,* $\mathrm{poly}(\log\log(1/\delta))$*-depth circuit over the standard basis, using ancillas. There is a polynomial algorithm that constructs this circuit on the description of U.*

This theorem will be proved and generalized in Section 13; see Theorem 13.5 on page 134 for a sharper result. The proof is based on a so-called phase estimation procedure (cf. Problem 13.4 — quantum Fourier transform).

As far as general bases are concerned, we will use the following definition.

Definition 8.6. Let $\mathcal{A}$ be a gate set that is closed under inversion. We call $\mathcal{A}$ a *complete basis* (or a universal gate set) if the applications of its elements

generate a dense subgroup in the group $\mathbf{U}(\mathcal{B}^{\otimes k})/\mathbf{U}(1)$ for some $k \geq 2$. (Here $\mathbf{U}(1)$ corresponds to multiplication by phase factors.)

(The phase factors are unimportant from the physical point of view, as well as for the definition of quantum computation that will be given in Section 9. If we really need to realize phase shifts, we can use the result of Problem 8.3.)

Remark 8.2. Why don't we accept ancillas in the definition of a complete basis? Indeed, it seems more natural to call a basis $\mathcal{A}$ complete if any unitary operator U can be realized with an arbitrary precision δ by a quantum circuit over this basis, using ancillas. Unfortunately, with this definition it is not clear how to estimate the size of the circuit in question. On the contrary, Definition 8.6 provides a rather general way of obtaining such an estimate; see Section 8.3. It is not known whether the two definitions of a complete basis are equivalent.

Remark 8.3. There is yet another definition of a complete basis, which is based on an even more general notion of realization of a unitary operator than the realization using ancillas. A basis is called complete if it can effect an arbitrary unitary operator on "encoded qubits" with any given precision (see Section 15 for exposition of quantum codes). The idea is that the quantum state of each qubit is represented by a state of several qubits; it is even permitted to have multiple representations of the same state.[4] This situation is characterized by an isometric map $V : \mathcal{B} \otimes \mathcal{F} \to \mathcal{B}^{\otimes k}$, in which case we say that a single *logical qubit* is represented by k *physical qubits* (the space $\mathcal{F}$ corresponds to the nonuniqueness of the representation). The gates of the basis act on physical qubits, whereas the operator we want to realize acts on logical qubits.

In such a general model, it is again possible to estimate the size of the circuit that is needed to achieve the given precision δ. Moreover, the gates of the basis can be specified with a constant precision δ_0, yet arbitrarily accurate realization is possible. This fundamental result is called the *threshold theorem for fault-tolerant quantum computation* [**65, 42, 3, 36**].

Theorem 8.4 (cf. [36]). *The standard basis $\mathcal{Q}$ is complete.*

Note that this theorem does not follow from Theorem 8.3. The proof of the theorem is contained in the solutions to Problems 8.10–8.12.

[4] Traditionally, a quantum code is defined as a single preferred representation, whereas the other representations are regarded as the preferred one, subjected to a "correctable error". Whatever the terminology, multiple representations allow us to perform computation with inaccurate gates. Such gates introduce "errors", or uncertainty in the resulting state, but one can arrange that it is only the choice of representation that is uncertain, the encoded state remaining intact.

Remark 8.4. If we remove the Toffoli gate from the basis $\mathcal{Q}$, it ceases to be complete. However, many important computations can be done even with such a reduced basis. In particular, as will be evident later, error-correcting circuits for quantum codes can be realized without the Toffoli gate.

Problems

[1] **8.7.** Prove the properties (8.10)–(8.13) of the operator norm.

[1] **8.8.** Prove the two basic properties of approximation with ancillas:

a) If $\tilde{U}$ approximates U with precision δ, then $\tilde{U}^{-1}$ approximates U^{-1} with the same precision δ.

b) If unitary operators $\tilde{U}_k$ approximate unitary operators U_k $(1 \le k \le L)$ with precision δ_k, then $\tilde{U}_L \cdots \tilde{U}_1$ approximates $U_L \cdots U_1$ with precision $\sum_k \delta_k$.

[3] **8.9.** Suppose that a unitary operator $\tilde{U}$ approximates a unitary operator U with precision δ, using ancillas. Prove that there exists an operator W that realizes U precisely (i.e., the equality $W\left(|\xi\rangle \otimes |0^{N-n}\rangle\right) = (U|\xi\rangle) \otimes |0^{N-n}\rangle$ holds) and satisfies $\left\|W - \tilde{U}\right\| \le O(\delta)$.

[2!] **8.10.** Suppose X and Y are noncommuting elements of the group $\mathbf{SO}(3)$ that rotate by angles incommensurate with π. Prove that the group generated by X and Y is an everywhere dense subset of $\mathbf{SO}(3)$.

[3!] **8.11.** Let $\mathcal{M}$ be a Hilbert space of finite dimension $M \ge 3$. Consider the subgroup $H \subset \mathbf{U}(\mathcal{M})$, the stabilizer of the 1-dimensional subspace generated by some unit vector $|\xi\rangle \in \mathcal{M}$. Let V be an arbitrary unitary operator not fixing the subspace $\mathbb{C}(|\xi\rangle)$. Prove that the set of operators $H \cup V^{-1}HV$ generates the whole group $\mathbf{U}(\mathcal{M})$.

(Note that under the conditions of this problem $\mathbf{U}(\mathcal{M})$ and H may be factored by the subgroup of phase shifts $\mathbf{U}(1)$.)

[3!] **8.12.** Prove that the applications of the operators from the standard basis generate an everywhere dense subset of $\mathbf{U}\left(\mathcal{B}^{\otimes 3}\right)/\mathbf{U}(1)$.

[2] **8.13.** Let $R = -i\exp(\pi i\alpha\sigma^x)$, α irrational. Prove that the negation σ^x, the *Deutsch gate* $\Lambda^2(R)$ and its inverse[5] form a complete basis.

8.3. Efficient approximation over a complete basis. How can one estimate the complexity of realizing a unitary operator U over a complete basis $\mathcal{A}$ with a given precision δ? How to construct the corresponding circuit efficiently? This questions arise if we want to simulate circuits over another

[5]The inverse of the Deutsch gate is not really necessary; it is included solely to conform to Definition 8.6.

basis $\mathcal{C}$ by circuits over $\mathcal{A}$. We would like to prove that such simulation does not increase the size of the circuit too much. In this regard, we may assume that $U \in \mathcal{C}$ is fixed, while δ tends to zero.

Let $U : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes n}$ be an arbitrary unitary operator. It can be represented by a matrix with complex entries, where each entry is a pair of real numbers, and each number is an infinite sequence of binary digits. We set the question of computing these digits aside. Instead, we assume that each matrix entry is specified with a suitable precision, namely, $\delta/2^{n+1}$. In this case the overall error in U, measured by the operator norm, does not exceed $\delta/2$. (Taking this input error into account, the algorithm itself should work with precision $\delta/2$, but we will rather ignore such details.)

The problem can be divided into two parts. First, we realize U over the infinite basis $\mathcal{A}_0$ that consists of all one-qubit and two-qubit gates. Second, we approximate each gate V of the resulting circuit C_0 by a circuit C over the basis $\mathcal{A}$. The first part is mostly done in the proof of Theorem 8.1; we just need to add some details. By examining the proof, we find that the circuit C_0 has size $L_0 = \exp(O(n))$. If we want to represent U with precision δ, we need to compute all gates of the circuit with precision $\delta' = \delta/L = \exp(-O(n))\,\delta$, which amounts to computing the entries of the corresponding matrices with precision $\delta'' = \delta'/2^n = \exp(-O(n))\,\delta$. This can be done in time $T = \exp(O(n)) \cdot \mathrm{poly}(\log(1/\delta))$. The presence of the exponential factor should not bother us, since in the practical application U is fixed, and so is n. Thus the first part is finished and we proceed to the second part.

8.3.1. ***Initial (nonconstructive) stage.*** Let us consider the problem of approximating an element $V \in \mathbf{U}(\mathcal{B}^{\otimes 2}) \subseteq \mathbf{U}(\mathcal{B}^{\otimes k})$ by a circuit C over the basis $\mathcal{A}$ with precision δ (the number k came from Definition 8.6). We are looking for approximations up to a phase factor; therefore we may assume that $V \in \mathbf{SU}(M)$, $\mathcal{A} \subseteq \mathbf{SU}(M)$, where $M = 2^k$. Then the circuit C is simply a sequence of elements $U_1, \ldots, U_L \in \mathcal{A}$ such that $\|V - U_L \cdots U_1\| \leq \delta$. Definition 8.6 guarantees that such a sequence exists, but its minimum length L can be arbitrary large (e.g., in the case where the elements of $\mathcal{A}$ are very close to the identity). So, before we can construct C in an effective fashion, some initial setup is required. We will now describe it briefly. In this description, we refer to some new concepts and facts that will be explained later.

First, we generate sufficiently many products of elements of $\mathcal{A}$ so that they form an ε-net, where ε is a suitable constant independent of M. This may take an arbitrary long time. The net may come out too crowded, but we can make it "α-sparse" ($\alpha = \mathrm{const}$) by removing redundant points. Such a net has at most $\exp(O(M^2))$ elements; see Problem 8.14. (This bound is tight. For sufficiently small ε, any ε-net has at least $\exp(\Omega(M^2))$ elements;

this is due to the fact that $\mathbf{SU}(M)$ is a manifold of dimension $M^2 - 1$.) Then we consider the net as a new basis. It is possible to obtain an upper bound for the approximation complexity relative to this basis, but not to the original one. As a part of our main theorem we will prove that any ε-net in $\mathbf{SU}(M)$ generates a dense subgroup, provided ε is small enough.[6]

It is the moment to recall some basic concepts from geometry. A *distance function* on a set S is a function $d : S \times S \to \mathbb{R}$, such that (i) $d(x, x) = 0$, (ii) $d(x, y) > 0$ if $x \neq y$, (iii) $d(x, y) = d(y, x)$, and (iv) $d(x, z) \leq d(x, y) + d(y, z)$.

A *δ-net for $R \subseteq S$* is a set $\Gamma \subseteq S$ whose δ-neighborhood contains R, i.e., for any $x \in R$ there is a $y \in \Gamma$ such that $d(x, y) \leq \delta$. We say that Γ *has no extra points* if any point of Γ belongs to the δ-neighborhood of R. The net Γ is called *α-sparse* ($0 < \alpha < 1$) if it has no extra points, and the distance between any two distinct points of Γ is greater than $\alpha\delta$.

The group $\mathbf{SU}(M)$ is equipped with the distance given by the operator norm, $d(U, V) = \|U - V\|$. Note that the diameter of $\mathbf{SU}(M)$ (the maximum possible distance) is 2. Let $r > \delta > 0$. Then an *(r, δ)-net* in $\mathbf{SU}(M)$ is a δ-net for the r-neighborhood of the identity; this neighborhood is denoted by S_r. The ratio $q = r/\delta > 1$, called *quality*, will play an important role in our arguments.

[2!] **Problem 8.14** ("Sparse nets").

a) ("Removing redundant points"). Let Γ be a δ-net for $R \subseteq S$. Prove that there is a subset $\Gamma_\alpha \subseteq \Gamma$ which is an α-sparse $\delta/(1 - \alpha)$-net for R.

b) ("Few points are left"). Prove that any α-sparse $(r, r/q)$-net in $\mathbf{SU}(M)$ has at most $(q/\alpha)^{O(M^2)}$ elements.

8.3.2. ***Main theorem.*** Let $\mathcal{A} \subset \mathbf{SU}(M)$ be a finite subset which is closed under inversion. The elements of $\mathcal{A}$ will be called *generators*. They can be treated in two ways: as elements of $\mathbf{SU}(M)$ (represented by matrices with a suitable precision), or as abstract symbols — references to the initially specified elements. A circuit is a sequence of such symbols, i.e., a word in the alphabet "$\mathcal{A}$". (We indicate the abstract use of generators by quotation marks, e.g., "U" $\in$ "$\mathcal{A}$".) The words constitute the *free group* F["$\mathcal{A}$"]. The product of two words is obtained by concatenation, whereas the inverse of "U_1" $\cdots$ "U_n" is "U_n^{-1}" $\cdots$ "U_1^{-1}").

Theorem 8.5. *There exists a universal constant $\varepsilon > 0$ such that for any $\nu > 0$ the following is true:*

[6] A more general statement is true: there is a universal constant $\varepsilon' > 0$ such that any ε'-net in any compact semisimple Lie group generates a dense subgroup, where the distance is measured by the operator norm for the adjoint action [**27**].

1. *For any M, an ε-net $\mathcal{A} \subset \mathbf{SU}(M)$ (closed under inversion), a number $\delta > 0$, and an element $V \in \mathbf{SU}(M)$, there is a sequence of generators $U_1, \dots, U_L \in \mathcal{A}$, $L = O\big((\log(1/\delta))^{3+\nu}\big)$, such that $\|V - U_L \cdots U_1\| \le \delta$.*
2. *Assuming that the net $\mathcal{A}$ is not too redundant, $|\mathcal{A}| = \exp(O(M^2))$, the function $(M, \delta, \mathcal{A}, V) \mapsto$ "U_L" $\cdots$ "U_1" can be computed by an algorithm with running time $T = \exp(O(M^2))\,(\log(1/\delta))^3 + O(L)$.*

Corollary 8.5.1. *Let $\mathcal{A}$ be a complete basis, and $\mathcal{C}$ an arbitrary finite basis. Then any circuit C of size L and depth d over the basis $\mathcal{C}$ can be simulated by a circuit C' of size $L' = O\big(L\,(\log(L/\delta))^c\big)$ and depth $d' = O\big(d\,(\log(L/\delta))^c\big)$ over the basis $\mathcal{A}$. (Here $c = 3+\nu$, whereas δ denotes the simulation precision: C realizes a unitary operator U, C' realizes U', and $\|U - U'\| \le \delta$.)*

The corollary is obvious: each gate of C should be approximated with precision δ/L. The simulation is very efficient in terms of size, but it is not so good in terms of depth. In a common situation $d \sim (\log L)^k$, $k \ge 1$, so that $d' = O(d^{1+c/k})$ (assuming that $\delta = \text{const}$).

Remark 8.5. The upper bound on the number of generators L in Theorem 8.5 can be improved if we drop the second condition (the existence of an efficient algorithm). Let $\mathcal{A} \subseteq \mathbf{SU}(M)$ be an arbitrary subset that is closed under inversion and generates a dense subgroup in $\mathbf{SU}(M)$. Then any element $U \in SU(M)$ can be approximated with precision δ by a product of $L \le C(\mathcal{A}) \log(1/\delta)$ generators [**31**]. On the other hand, the lower bound $L \ge \Omega(M^2) \log(1/\delta)/\log|\mathcal{A}|$ follows from a volume consideration.

Remark 8.6. The presence of the exponential $\exp(O(M^2))$ in the algorithm complexity bound is rather disturbing (recall that $M = 2^k$, where k is the number of qubits). As far as the asymptotic behavior at $\delta \to 0$ is concerned, it seems possible to make the computation polynomial in M, that is, the exponential may become an additive term rather than a factor. (To this end, one may try to use bases in the tangent space instead of nets — the reader is welcome to explore this idea.) However, it is a challenge to eliminate the exponential altogether. This may be only possible if one changes the assumptions of the theorem, e.g., by saying that products of $\text{poly}(M)$ elements from $\mathcal{A}$ constitute an ε-net (rather than $\mathcal{A}$ being an ε-net itself). Such a basis $\mathcal{A}$ can consist of only $\text{poly}(M)$ elements, so it is reasonable to ask whether there is an approximation algorithm with running time $\text{poly}(M \log(1/\delta))$. This appears to be a difficult question in global unitary geometry.

8.3.3. ***Idea of the proof and geometric lemmas.*** The proof of Theorem 8.5 is based on four geometric properties of the group $\mathbf{SU}(M)$ endowed with the operator norm distance d. First, the distance is *biinvariant*, i.e.

$d(WU,WV) = d(U,V) = d(UW,VW)$. The second property is the result of Problem 8.14b; this is the only place where the number M comes in. The remaining two properties are related to the group commutator, $[\![U,V]\!] = UVU^{-1}V^{-1}$. Let us consider the application of the group multiplication and the group commutator to a pair of subsets $A, B \subseteq \mathbf{SU}(M)$,

$$AB = \bigl\{UV \,:\, U \in A,\ V \in B\bigr\}, \qquad [\![A,B]\!] = \bigl\{[\![U,V]\!] \,:\, U \in A,\ V \in B\bigr\}.$$

Then the following inclusions hold:

$$[\![S_a, S_b]\!] \subseteq S_{2ab}, \tag{8.17}$$

$$S_{ab/4} \subseteq [\![S_a, S_b]\!]\, S_{O(ab(a+b))}. \tag{8.18}$$

(Recall that S_r denotes the r-neighborhood of the identity. The right-hand side of the last inclusion represents the $O(ab(a+b))$-neighborhood of $[\![S_a, S_b]\!]$. Indeed, the r-neighborhood of any set T can be expressed as TS_r.)

The inclusion (8.17) is easy to prove:

$$\begin{aligned}\bigl\|[\![U,V]\!] - I\bigr\| &= \bigl\|UV - VU\bigr\| = \bigl\|(U-I)(V-I) - (V-I)(U-I)\bigr\| \\ &\le 2\,\|U-I\|\,\|V-I\|.\end{aligned}$$

Formula (8.18) can be proved by approximating the group commutator by the corresponding Lie algebra bracket.[7] In our case, the Lie algebra is formed by skew-Hermitian $M \times M$ matrices with zero trace,

$$\mathfrak{su}(M) = \bigl\{X \,:\, X^\dagger = -X,\ \operatorname{Tr} X = 0\bigr\}, \qquad [X,Y] = XY - YX.$$

The exponential map $\exp : \mathfrak{su}(M) \to \mathbf{SU}(M)$ is simply the matrix exponentiation.

The eigenvalues of $X \in \mathfrak{su}(M)$ have the form

$$\text{eigenvalues}(X) = \bigl\{ix_1, \ldots, ix_M\bigr\}, \qquad \sum_{k=1}^{M} x_k = 0, \qquad x_k \in \mathbb{R}.$$

Using a basis in which X is diagonal, one can see that if $\|X\| = t \le \pi$ then $\|\exp(X) - I\| = 2\sin(t/2)$. Therefore $\|\exp(X) - I\| \approx \|X\|$ if either of these numbers is small. Let R_t denote the t-neighborhood of 0 in $\mathfrak{su}(M)$ (with respect to the operator norm). The map $\exp : R_t \to S_{2\sin t/2}$ is bijective for $t < \pi$. So we may represent group elements near the identity as $\exp(X)$ and try to replace $[\![\exp(X), \exp(Y)]\!]$ by $\exp([X,Y])$; see inequality (8.21) below. Thus the inclusion (8.18) can be obtained from the result of the following problem.

[3!] **Problem 8.15.** Prove that $R_{ab/4} \subseteq [R_a, R_b] \subseteq R_{2ab}$.

[7] We assume that the reader knows some basic facts about Lie groups and Lie algebras, which can be found in the first chapter of any textbook (see e.g. [**1, 15, 34, 56**]).

(Note that for $\mathfrak{su}(2)$, $[R_a, R_b] = R_{2ab}$ — this follows from the standard representations of the bracket in $\mathfrak{su}(2) \cong \mathfrak{so}(3)$ as the vector product in $\mathbb{R}^3$.)

[2!] **Problem 8.16.** Prove that for any $X, Y \in \mathfrak{su}(M)$

$$\big\|\exp(X) - I - X\big\| \le O\big(\|X\|^2\big), \tag{8.19}$$

$$\big\|\exp(X)\exp(Y) - \exp(X+Y)\big\| \le O\big(\|X\|\,\|Y\|\big), \tag{8.20}$$

$$\big\|[\![\exp(X), \exp(Y)]\!] - \exp([X,Y])\big\| \le O\big(\|X\|\,\|Y\|\,\big(\|X\| + |Y\|\big)\big). \tag{8.21}$$

(The implicit constants in $O(\dots)$ should not depend on M.)

How does one use the above four properties in an effective procedure? We are going to define three important operations with nets in $\mathbf{SU}(M)$ (in addition to removing redundant points; see Problem 8.14a). Operations, called "shrinking" and "telescoping", are used to build increasingly tight nets $\Gamma_0, \Gamma_1, \dots, \Gamma_n$, $n = O(\log(1/\delta))$ in increasingly small neighborhoods of the identity. Specifically, each Γ_j is an $(r_j, r_j/q)$-net, where $r_j = r_0\lambda^{-j}$, $q > \lambda > 1$ (q and λ are some constants). Elements of Γ_j are products of $L_j = O(j^{2+\nu})$ generators. Then we use the constructed nets to approximate an arbitrarily element $V \in \mathbf{SU}(M)$ in a procedure called "zooming in" (think of the nets as magnifying glasses of different strength).

"Shrinking" is the operation that employs the group commutator. It does what its name suggests, namely makes smaller nets from bigger ones. An $(r, r/q)$-net shrinks to an $(r^2/4, 5r^2/q)$-net. Suppose that elements of the original net were products of l generators. Taking the commutator multiplies l by 4, whereas the radius r gets approximately squared, and so does the precision $\delta = r/q$ (we assume that q is bounded by a constant). Repetition of this procedure could yield the desired rate of compression, and even better, $\delta \sim r \sim \exp(-l^{1/2})$. However, the quality of the net degrades at each step, $q \mapsto q/20$. The "telescoping" comes to rescue, but at some cost. Also, we need to select a sparse subnet after each shrinking to keep the number of points in each net bounded by $\exp(O(M^2))$. The resulting rate of compression is slightly lower, $\delta \sim r \sim \exp(-l^{1/(2+\nu)})$, where ν can be arbitrary small. Therefore $l = O\big((\log(1/\delta))^{2+\nu}\big)$.

[2!] **Problem 8.17.** In items (b) and (c) below, assume that G is an arbitrary group with a biinvariant distance function d. The result of (a) is specific to $\mathbf{SU}(M)$.

(a) ("Shrinking"). Let $\Gamma_1 \subseteq \mathbf{SU}(M)$ be an $(r_1, r_1/q)$-net, and $\Gamma_2 \subseteq \mathbf{SU}(M)$ an $(r_2, r_2/q)$-net. Denote by $[\![\Gamma_1, \Gamma_2]\!]_\alpha$ an α-sparse subnet selected from $[\![\Gamma_1, \Gamma_2]\!] = \big\{[\![U_1, U_2]\!] : U_1 \in \Gamma_1,\ U_2 \in \Gamma_2\big\}$ (see Problem 8.14a). Prove that $[\![\Gamma_1, \Gamma_2]\!]_{1/6}$ is an $(r_1 r_2/4, 5r_1 r_2/q)$-net provided $q > 20$ and $r_1, r_2 \le O(q^{-1})$.

(b) ("Telescoping" — combining two nets into one of higher quality). Let $\Gamma_1 \subseteq G$ be an (r_1, δ_1)-net, and $\Gamma_2 \subseteq G$ an (r_2, δ_2)-net, where $\delta_1 \le r_2$. Prove that the set $\Gamma_1\Gamma_2 = \{U_1U_2 \,:\, U_1 \in \Gamma_1,\ U_2 \in \Gamma_2\}$ is an (r_1, δ_2)-net.

(c) ("Zooming in" — iterative approximation). Let $\Gamma_0, \Gamma_1, \dots, \Gamma_n \subseteq G$ be a sequence of nets: Γ_0 is a δ_0-net for the entire G, whereas for $j \ge 1$ each Γ_j is an (r_j, δ_j)-net. Suppose that $r_j = r_0\lambda^{-j}$, $\delta_j = \delta_0\lambda^{-j}$, where $r_0/\delta_0 = q > \lambda > 1$. Prove that any element $V \in G$ can be approximated by $Z = Z_0Z_1 \cdots Z_n$ $(Z_j \in \Gamma_j)$ so that $d(V, Z) \le \delta_n$.

8.3.4. ***The algorithm.*** Without loss of generality, we may assume that $\nu = 1/p$, where p is an integer. The algorithm consists of three stages.

Preprocessing. Computation at this stage does not depend on V. We build increasingly tight nets in increasingly small neighborhoods of the identity. This is done by shrinking an initial net p times and "telescoping" it with one of the previous nets to regain the original quality; then the cycle repeats. More precisely, we construct a set of nets $\Gamma_{j,k}$, $j = 0, \dots, n = O(\log(1/\delta))$, $k = 0, \dots, p$, according to the recursion rules

$$\Gamma_{j,k} = \big[\!\big[\Gamma_{\lceil j/2\rceil,k-1}, \Gamma_{\lfloor j/2\rfloor,k-1}\big]\!\big]_{1/6} \qquad \text{for } k = 1, \dots, p,$$
$$\Gamma_{j,0} = \Gamma_{j-1,p}\,\Gamma_{j,p}.$$

Each $\Gamma_{j,k}$ is an $(r_{j,k},\, r_{j,k}/q_k)$-net, where

$$r_{j,k} = r_{0,k}\lambda^{-j}, \quad r_{0,k} = 4C^{-p2^k/(2^p-1)}, \quad q_k = C^{2p-k}, \quad \lambda = C^p, \quad C = 20.$$

The recursion relations work only for sufficiently large j (namely, $r_{j,k}$ should be small enough to satisfy the condition of Problem 8.17a). The first few nets have to be obtained by picking points from the initial net $\mathcal{A}$; hence we need to set the constant ε small enough. (According to this rule, the constant ε depends on p. To avoid such dependence, we need to run the first few steps using $p = 1$, and then switch to the desired p.)

Each element of $\Gamma_{j,k}$ is a product of $L_{j,k}$ generators. The numbers $L_{j,k}$ satisfy the relations

$$L_{j,k} = 2L_{\lceil j/2\rceil,k-1} + 2L_{\lfloor j/2\rfloor,k-1} \ \ (k = 1, \dots, p), \qquad L_{j,0} = L_{j-1,p} + L_{j,p}.$$

An upper bound $L_{j,k} \le j^{2+1/p}\, 2^{-k/p}\big(u_0 - u_1/j\big)$ (with constant u_0 and u_1) can be obtained by induction; hence $L_{j,k} = O(j^{2+\nu})$.

When constructing elements of the nets $\Gamma_{j,k}$, we do not actually write them as sequences of generators. Instead, we represent them as $M \times M$ matrices and keep record of the way each element was obtained. This stage involves $\exp(O(M^2))\log(1/\delta)$ matrix multiplications, which amounts to $\exp(O(M^2))(\log(1/\delta))^3$ bit operations.

Iterative approximation. We use the nets $\Gamma_{j,0}$ as indicated in Problem 8.17c. This yields an element $Z = Z_0 \cdots Z_n$ $(Z_j \in \Gamma_{j,0})$ such that $\|V - Z_0 \cdots Z_n\| \le \delta$. The complexity of this stage is also $\exp(O(M^2))(\log(1/\delta))^3$.

Expansion. Now we need to represent each Z_j as a word in the alphabet "$\mathcal{A}$". We have already computed Z_j as matrices, so we know the sequence of matrix multiplications and inversions that have led to Z_j. In other words, we have a classical circuit over the basis $\{\text{MULTIPLICATION}, \text{INVERSION}\}$. This circuit will perform computation over the free group as well. Thus we plug symbols of the alphabet "$\mathcal{A}$" to the inputs of the circuit, and get some words w_j as the output; then we concatenate them to obtain the word w representing Z. When computing w_j, we operate with exponentially increasing words; therefore the complexity is dominated by the last step. So, the number of operations is $O(|w_j|) = O(L_{j,0}) = O(j^{2+\nu})$. Summing over j, we conclude that w is computed in $O(L)$ steps, where $L = |w| = O(n^{3+\nu})$ (recall that $n = O(\log(1/\delta))$).

9. Definition of Quantum Computation. Examples

9.1. Computation by quantum circuits. Until now, we have been describing the work of a quantum computer. Now it is time to define when this work leads to the solution of problems that are interesting to us. The definition will resemble the definition of probabilistic computation.

Consider a function $F : \mathbb{B}^n \to \mathbb{B}^m$. We examine a quantum circuit operating with n bits, $U = U_L \cdots U_2 U_1 : \mathcal{B}^{\otimes N} \to \mathcal{B}^{\otimes N}$. Loosely speaking, this circuit computes F if, after having applied U to the initial state $|x, 0^{N-n}\rangle$ and "having looked" at the first m bits, we "see" $F(x)$ *with high probability.* (The remaining qubits can contain arbitrary garbage.)

We only need to discuss the nature of that probability. The precise meaning of the words "having looked" and "see" is that a *measurement* of the values of the corresponding qubits is performed. Several different answers can be obtained as the result of this measurement, each with its own probability. Later (in Section 10) this question will be considered in more details. To give a definition of quantum computation of a function F, it suffices (without injecting physical explanations of this fact) to accept the following: *the probability of getting a basis state x in the measurement of the state $|\psi\rangle = \sum_x c_x |x\rangle$ equals*

$$\mathbf{P}(|\psi\rangle, x) = |c_x|^2. \tag{9.1}$$

We are interested in the probability that the computer will finish its work in a state of the form $(F(x), z)$, where z is arbitrary.

Definition 9.1. The circuit $U = U_L \cdots U_2 U_1$ *computes* F if for any x we have

$$\sum_z |\langle F(x), z|U|x, 0^{N-n}\rangle|^2 \geq 1 - \varepsilon,$$

where ε is some fixed number smaller than $1/2$. (Note that $F(x)$ and x consist of different numbers of bits, although the total lengths of $(F(x), z)$ and $(x, 0^{N-n})$ must be equal to N.)

Just as for probabilistic computation, the choice of ε is unimportant, inasmuch as it is possible to effect several copies of the circuit independently and to choose the result that is most frequently obtained. From the estimate (4.1) on p. 37 it follows that in order to decrease the probability of failure by a factor of a, we need to take $k = \Theta(\log a)$ copies of the circuit U. The choice of the most frequent result is realized by a classical circuit, using the majority function $\mathrm{MAJ}(x_1, \ldots, x_k)$ (which takes value 1 when more than half of its arguments equal 1 and value 0 otherwise). The function $\mathrm{MAJ}(x_1, \ldots, x_k)$ can be realized over a complete basis by a circuit of size $O(k)$ (see Problem 2.15). Therefore the a-fold reduction of the error probability is achieved at the cost of increasing the circuit size by the factor $O(\log a)$.

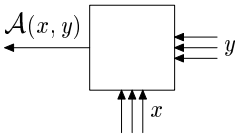
[1] **Problem 9.1.** Prove that the above argument is actually correct if we incorporate the function MAJ into the quantum circuit. Specifically, show that we may use the function $\mathrm{MAJ}_\oplus$ realized by a reversible circuit, so that its input bits are the output qubits of k copies of the circuit U.

[2!] **Problem 9.2.** Suppose that each gate of the circuit U_k is approximated by $\tilde{U}_k$ with precision δ. Prove that the resulting circuit $\tilde{U} = \tilde{U}_L \cdots \tilde{U}_2 \tilde{U}_1$ satisfies the inequality from Definition 9.1, with ε replaced by $\tilde{\varepsilon} = \varepsilon + 2L\delta$.

(The suggested solution is based on the general notion of quantum probability; see Section 10, especially Remark 10.1.)

Now that we have the definition of quantum computation, we can make a comparison of the effectiveness of classical and quantum computing. In the Introduction we mentioned three fundamental examples where quantum computation appears to be more effective than classical. We begin with the example where the greater effectiveness of quantum computation has been proved (although the increase in speed is only polynomial).

9.2. Quantum search: Grover's algorithm. We will give a definition of a *general search problem* in classical and quantum formulations. It belongs to a class of computational problems *with oracle*, in which the input is given as a function (called a "black box", or an oracle) rather than a binary word.

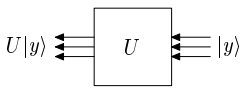


Suppose we have a device (see the diagram) that receives inputs x and y and determines the value of some predicate $\mathcal{A}(x,y)$. We are interested in the predicate $F(x) = \exists y \mathcal{A}(x,y)$. This resembles the definition of the class NP, but now the internal structure of the device calculating the predicate $\mathcal{A}$ is inaccessible to us. Under such conditions, it is not possible to complete the calculation faster than in $N = 2^n$ steps on a classical computer, where n is the number of bits in the binary word y.

The problem can be formulated even without x: we need to compute the value of the "functional" $\mathcal{F}(\mathcal{A}) = \exists y \mathcal{A}(y)$. If x is present, we can regard it as a part of the oracle, i.e., replace $\mathcal{A}$ with the predicate $\mathcal{A}_x$ such that $\mathcal{A}_x(y) = \mathcal{A}(x,y)$. Then $\mathcal{F}(\mathcal{A}_x) = F(x) = \exists y \mathcal{A}(x,y)$.

Remark 9.1. The version of the problem without x has another interpretation, which is quite rigorous (unlike the analogy with NP). Let us think of $\mathcal{A}$ as a bit string: y is the index of a bit, whereas $\mathcal{A}(y)$ is the value of that bit (cf. the paragraph preceeding Definition 2.2 on page 26). Then $\mathcal{F}(\mathcal{A}) = \bigvee_{y \in \mathbb{B}^n} \mathcal{A}(y)$, the *OR* function with $N = 2^n$ inputs.

It turns out that a quantum computer can determine the value of $\mathcal{F}(\mathcal{A})$ and even find a y for which $\mathcal{A}(y)$ is satisfied, in time $O(\sqrt{N})$. The lower bound $\Omega(\sqrt{N})$ has also been obtained, showing that in this situation quantum computers give only a quadratic speed-up in comparison with classical ones.



In the quantum formulation, the problem looks as follows. A quantum computer can query the oracle by sending y so that different values of y may form superpositions, and the oracle will return superpositions accordingly. Interestingly enough, the oracle can encode the answer into a phase factor. Specifically, our oracle (or "black box") is defined as an operator U acting by the rule

$$U|y\rangle = \begin{cases} |y\rangle & \text{if } \mathcal{A}(y) = 0, \\ -|y\rangle & \text{if } \mathcal{A}(y) = 1. \end{cases}$$

We assume that the computer can choose whether to query the oracle or not, which corresponds to applying the operator $\Lambda(U)$.

The goal is to compute the value $\mathcal{F}(\mathcal{A})$ and find an "answer" y for which $\mathcal{A}(y)$ is satisfied. This should be done by a quantum circuit, using $\Lambda(U)$ as a gate (in addition to the standard basis).

The results that we have already mentioned are formulated as follows (cf. [**29, 75**]): *there exist two constants C_1, C_2 such that there is a circuit of size $\le C_1\sqrt{N}$, deciding the problem for an arbitrary predicate $\mathcal{A}$; and, for*

an arbitrary circuit of size $\le C_2\sqrt{N}$*, there exists a predicate* $\mathcal{A}$ *for which the problem is not decided by this circuit (i.e., the circuit gives an incorrect answer with probability* $> 1/3$*).*

We will construct a quantum circuit for a simplified version of the problem: we assume that the "answer" exists and is unique, and we denote it by y_0; we need to find y_0. The circuit will be described in terms of operator action on the basis vectors.

Consider two operators

$$U = I - 2|y_0\rangle\langle y_0|,$$
$$V = I - 2|\xi\rangle\langle\xi|, \quad \text{where } |\xi\rangle = \frac{1}{\sqrt{N}}\sum_y |y\rangle.$$

The operator U is given to us (it is the oracle). The operator V is represented by the matrix

$$V = \begin{pmatrix} 1-\frac{2}{N} & \dots & -\frac{2}{N} \\ \vdots & \ddots & \vdots \\ -\frac{2}{N} & \dots & 1-\frac{2}{N} \end{pmatrix}$$

(recall that $N = 2^n$).

Let us realize V by a quantum circuit. We will proceed as follows: we transform $|\xi\rangle$ to $|0^n\rangle$ by some operator W, then apply the operator $I - 2|0^n\rangle\langle 0^n|$, and finally apply W^{-1}.

It is easy to construct an operator W that takes $|\xi\rangle$ to $|0^n\rangle$. The following will do: $W = H^{\otimes n}$, where H is the Hadamard gate from the standard basis (see Definition 8.5). In fact, $|\xi\rangle = \frac{1}{\sqrt{2^n}}(|0\rangle+|1\rangle)^{\otimes n}$, and $H : \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \mapsto |0\rangle$.

Now we have to implement the operator $I - 2|0^n\rangle\langle 0^n|$. We will use a reversible classical circuit that realizes the operator $Z : \mathcal{B}^{n+1} \to \mathcal{B}^{n+1}$,

$$Z|a_0, \dots, a_n\rangle = |a_0 \oplus f(a_1, a_2, \dots, a_n),\, a_1, \dots, a_n\rangle;$$
$$f(a_1, \dots, a_n) = \begin{cases} 1 & \text{if } a_1 = \dots = a_n = 0, \\ 0 & \text{if } \exists\, j : a_j \ne 0. \end{cases}$$

(Up to a permutation of the arguments, $Z = \widehat{f_\oplus}$.) Since f has Boolean circuit complexity $O(n)$, Z can be realized by a reversible circuit of size $O(n)$ (see Lemma 7.2).

The circuit that realizes the operator V is shown in Figure 9.1. The central portion, incorporating Z, σ^z and Z, realizes the operator $I - 2|0^n\rangle\langle 0^n|$. In this circuit, the operator $\sigma^z = K^2$ (K from the standard basis) is used.

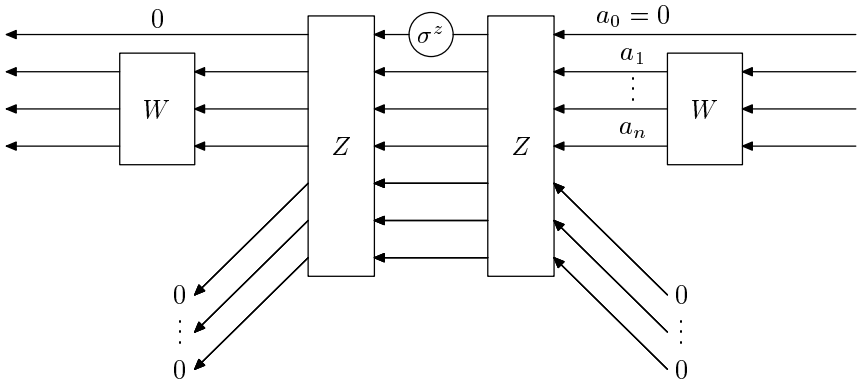


Fig. 9.1. Implementation of the operator V.

We note that W^2 and Z^2 act trivially (as the identity operator) on vectors with zero-valued borrowed qubits. Therefore the decisive role is played by the operator σ^z acting on an auxiliary qubit, which likewise returns to its initial value 0 in the end.

We must not be confused by the fact that although σ^z acts only on an $f_\oplus$-controlled qubit, the whole vector changes as a result. In general, the distinction between "reading" and "writing" in the quantum case is not absolute and depends on the choice of basis. Let us give a relevant example.

Let us find the matrix of $\Lambda(\sigma^x) : |a, b\rangle \mapsto |a, a \oplus b\rangle$ relative to the basis $\frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$ for each of the qubits. In other words, we need to write the matrix of the operator $X = (H \otimes H)\,\Lambda(\sigma^x)\,(H \otimes H)$ relative to the classical basis. The circuit for this operator is shown in Figure 9.2. Using the equality $\Lambda(\sigma^x)|c, d\rangle = |c,\, c \oplus d\rangle$, we find the action of X on any basis vector:

$$\begin{aligned} X|a, b\rangle &= \frac{1}{2}(H \otimes H)\Lambda(\sigma^x) \sum_{c,d} (-1)^{ac+bd}|c, d\rangle \\ &= \frac{1}{2}(H \otimes H) \sum_{c,d} (-1)^{ac+bd}|c, c \oplus d\rangle \\ &= \frac{1}{4} \sum_{a',b',c,d} (-1)^{a'c+b'(c+d)}(-1)^{ac+bd}|a', b'\rangle \\ &= \frac{1}{4} \sum_{a',b'} 2\delta_{b,b'} \cdot 2\delta_{a,\,a'\oplus b'}|a', b'\rangle = |a \oplus b, b\rangle. \end{aligned}$$

Thus, in the basis $\frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$, the controlling and the controlled qubits have changed places. Which bit is "controlling" (is "read") and which is "controlled" (is "modified") depends on the choice of basis. Of course, such a situation goes against our classical intuition. It is hard to imagine that by

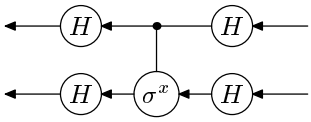
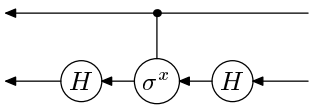


Fig. 9.2. $\Lambda(\sigma^x)$ in a different basis.

passing to a different basis, a quantum printer suddenly becomes a quantum scanner.

[1] **Problem 9.3.** What will happen if we change the basis only in one of the qubits?

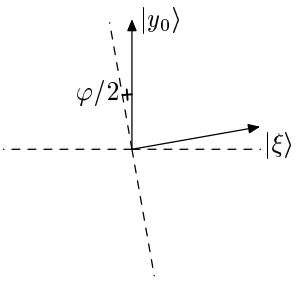
For example, what will the matrix of the operator with the circuit shown in the diagram look like? Also try to change the basis in the other qubit.



Let us return to the construction of a circuit for the general search problem. What follows is the main part of Grover's algorithm. The oracle $U = I - 2|y_0\rangle\langle y_0|$ was given to us, and we have realized the operator $V = I - 2|\xi\rangle\langle\xi|$. We start computation with the vector $|\xi\rangle$, which can be obtained from $|0^n\rangle$ by applying the operator W. Now, with the aid of the operators U and V, we are going to transform $|\xi\rangle$ to the solution vector $|y_0\rangle$. For this, we will apply alternately the operators U and V:

$$\cdots VUVU|\xi\rangle = (VU)^s|\xi\rangle.$$

What do we get from this? Geometrically, both operators are reflections through hyperplanes. The subspace $\mathcal{L} = \mathbb{C}\,(|\xi\rangle, |y_0\rangle)$ is invariant under both operators, and thus, under VU. Since the initial vector $|\xi\rangle$ belongs to $\mathcal{L}$, it suffices to consider the action of VU on this subspace.



The composition of two reflections with respect to two lines is a rotation by twice the angle between those lines. The angle is easy to calculate: $\langle\xi|y_0\rangle = \frac{1}{\sqrt{N}} = \sin\frac{\varphi}{2}$, i.e., the lines are almost perpendicular. Therefore we may write $VU = -R$, where R is the rotation by the small angle φ. But then $(VU)^s = (-1)^s R^s$, where R^s is the rotation through the angle $s\varphi$. The sign does not interest us (phase factors do not affect probabilities). For large N, we have $\varphi \approx 2/\sqrt{N}$. Then, after $s \approx (\pi/4)\sqrt{N}$ iterations, the initial vector is turned by an angle $s\varphi \approx \pi/2$ and becomes close to the solution vector. This also indicates that the system ends up in the state $|y_0\rangle$ with probability close to one.

To solve the search problem in the most general setting (when there may be several answers, or there may be none), additional technical devices are needed. Note that the number of steps for the rotation from the initial vector to some vector of the subspace spanned by the solution vectors is inversely proportional to the square root of the number of solutions.

Problem 9.4. For a given n, construct poly(n)-size quantum circuits (over the basis of all two-qubit gates) which perform the following tasks.

[2] a) For a given number q, $1 \le q \le 2^n$, transform the state $|0^n\rangle$ into the state $|\eta_{n,q}\rangle = \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle$.

[2] b) Transform $|q-1, 0^n\rangle$ into $|q-1\rangle \otimes |\eta_{n,q}\rangle$ for all q, assuming that q is expressed in n binary digits.

[3!] c) Realize the Fourier transform operator F_q over the group $\mathbb{Z}_q$:

$$F_q|x\rangle = \frac{1}{\sqrt{q}} \sum_{y=0}^{q-1} \exp\left(2\pi i \frac{xy}{q}\right) |y\rangle,$$

where x and y are expressed in n binary digits. Consider the case $q = 2^n$.

(The case of arbitrary q requires some extra tools, so we will consider it later; see Problem 13.4.)

9.3. A universal quantum circuit. The second of the examples mentioned in the Introduction was simulation of a quantum mechanical system. This is a vaguely posed problem since the choice of particular systems and distinguishing "essential" degrees of freedom play an important role. The problem has been actually solved in several settings. With high confidence, we may claim that every physical quantum system can be efficiently simulated on a quantum computer, but we can never prove this statement. The situation resembles that of Turing's thesis (see Section 1.3). Recall that the validity of Turing's thesis is partially justified by the existence of the universal Turing machine. In this vein, we may examine universality of our quantum computation model by purely mathematical means. Let us try to simulate many circuits by one.

We will not limit the type of gates we use to any particular basis. General quantum circuits have manageable description if the gates are specified as matrices with entries given by binary fractions to certain precision δ_1. Then the inaccuracy of an r-qubit gate (in the operator norm) does not exceed $\delta = M\delta_1$, where $M = 2^r$ is the size of the matrix. Suppose we have a description Z of a quantum circuit of size $\le L$ and precision δ. Each gate of the circuit acts on at most r qubits, so that the total length of the description does not exceed poly $(L2^r \log(1/\delta))$. The operator realized by the circuit will

be denoted by $\mathrm{Op}(Z)$. We will try to simulate all circuits with the given parameters L, r, δ.

Using the algorithm from the proof of Theorem 8.1, we reduce the problem to the case $r = 2$. Then we apply Theorem 8.3. Thus we can realize each operator in Z by a circuit of size $\mathrm{poly}(2^r \log(1/\delta))$ over the standard basis using $O(r)$ ancillas. This yields (a description of) a circuit $R(Z)$ over the standard basis, which has size $S = \mathrm{poly}\,(L2^r \log(1/\delta))$, operates on $N = L + O(r)$ qubits, and approximates $\mathrm{Op}(Z)$ with precision $O(L\delta)$. The transformation $Z \mapsto R(Z)$ is performed by a Boolean circuit of size $\mathrm{poly}\,(L2^r \log(1/\delta))$. Hence simulating a general circuit is not much harder than simulating circuits over the standard basis.

The result is as follows. There is a *universal quantum circuit* U of size $\mathrm{poly}\,(L2^r \log(1/\delta))$ that simulates the work of an arbitrary quantum circuit in the following way: for any circuit description Z and input vector $|\xi\rangle$, U satisfies the condition

$$\left\| U\Big(|Z\rangle \otimes |\xi\rangle \otimes |0^k\rangle\Big) - |Z\rangle \otimes \big(\mathrm{Op}(Z)|\xi\rangle\big) \otimes |0^k\rangle \right\| = O(L\delta).$$

That is, U works as a "programmable quantum computer", with Z being the "program".

The qubits of the circuit U include N "controlled" qubits that correspond to the qubits of $R(Z)$. Another subset of qubits holds $|Z\rangle$. There is also a number of auxiliary qubits, some of which are called "controlling". The key component of the circuit U is a circuit V, the product of the operators $V_j = \Lambda(X)[j, k_j]$ (or $V_j = \Lambda(X)[j, k_j, l_j]$, or $V_j = \Lambda(X)[j, k_j, l_j, m_j]$), with X from the standard basis, applied to each one (or pair, or triple) of the controlled qubits in an arbitrary order. The controlling qubits j are all different. If we set one controlling qubit to 1 and all the others to 0, then the circuit V realizes an operator of the form $X[k]$ (or $X[k, l]$, or $X[k, l, m]$) on the controlled qubits. Hence the composition of S copies of V with different controlling qubits can simulate an arbitrary circuit of size S over the standard basis, provided that the controlling qubits are set appropriately. To set the controlling qubits, we need to compute $R(Z)$ by a reversible circuit (with garbage) and arrange the output in a certain way. This computation should be reversed at the end.

9.4. Quantum algorithms and the class BQP. Up until now we have been studying nonuniform quantum computation (i.e., computation of Boolean functions). Algorithms compute functions on words of arbitrary length. A definition of a quantum algorithm can be given using quantum circuits that have been already introduced. Roughly speaking, a classical Turing machine builds a quantum circuit that computes the value of the function on one or many inputs. Actually, there are several equivalent definitions,

the following being the standard one. Let $F : \mathbb{B}^* \to \mathbb{B}^*$ be a function such that the length of the output is polynomial in the length of the input. It is composed of a sequence of Boolean functions $F_n : \mathbb{B}^n \to \mathbb{B}^{m(n)}$ (restrictions of F to inputs of length $n = 0, 1, 2, \dots$). A *quantum algorithm* for the computation of F is a uniform sequence of quantum circuits that compute F_n. *Uniform* means that the description Z_n of the corresponding circuit is constructed by a classical Turing machine which takes n as the input. We will say that the quantum algorithm *computes F in time $T(n)$* if building the circuit takes at most $T(n)$ steps. The size of the circuit L is obviously not greater than $T(n)$.

A subtle point in this definition is what basis to use. It is safe to stick to the standard basis. Alternatively, the basis may consist of all unitary operators. In this case, each r-qubit gate should be specified as a list of all its matrix elements with precision $c2^{-r}L^{-1}$, so that the precision of the matrix (in the operator norm) is cL^{-1}, where c is a small constant. If $\varepsilon + 2c < 1/2$ (see Definition 9.1 and Problem 9.2) then the approximate circuit works fine. Using the algorithm of Theorems 8.1 and 8.3, this circuit can be transformed to an equivalent circuit of size $\mathrm{poly}(T(n))$ over the standard basis (note that $T(n)$ includes the factor 2^r). The converse is obvious.

Remark 9.2. The use of an arbitrary complete basis could lead to "pathologies". For example, let the basis contain the gate

$$X = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix},$$

where θ is a noncomputable number, e.g., the n-th digit of θ says whether the universal Turing machine terminates at input n (cf. Problem 1.3). Then $p = \sin^2\theta$ is also noncomputable. If we apply X to the state $|0\rangle$ and measure the qubit, we will get 1 with probability p and 0 with probability $1-p$. Repeating this procedure $\exp(\Theta(n))$ times and counting the number of 0s and 1s, we can find the n-th digit of p with very small error probability. Thus the gate X enables us to solve the halting problem! (This argument has nothing to do with quantum mechanics. A classical probabilistic computer could also get extra power if random numbers with arbitrary p were allowed.) Of course, we want to avoid such things in our theory, so we must be careful about the choice of basis. However, in the real world "superpowerful" gates might exist. Experimentalists measure dimensionless physical constants (such as the fine structure constant) with increasingly high precision, getting new digits of the number theoreticians cannot compute. If we ever learn where the fundamental physical constants come from, we will probably know whether they are computable, and if they are not, whether they carry some mathematically meaningful information, e.g., allow one to solve the halting problem.

Remark 9.3. It is possible to define a quantum Turing machine directly, through superpositions of various states of a classical Turing machine (the original definition of D. Deutsch [**20**] was just like this). The standard definition turns out to be equivalent but more convenient.

Using the universal quantum circuit, we can simplify the standard definition even further. It suffices to have a classical Turing machine that generates a description of a quantum circuit $Z(x)$ which is only good to compute $F(x)$ *for a single value of* x. In this case, x is the input word for the TM whereas the circuit does not have input data at all (i.e., it operates on supplementary qubits initialized by the state $|0^N\rangle$). Indeed, if we have such a machine M, then we can build a machine M' which receives n and constructs a Boolean circuit which computes $Z(x)$ for all values of x, $|x| = n$ (see Theorem 2.3). By a certain polynomial algorithm, the Boolean circuit can be transformed into a reversible circuit (with garbage) and combined with the universal quantum circuit, so that the output of the former (i.e., $Z(x)$) becomes the "program" for the latter. This yields a quantum circuit that computes F_n.

Thus we will adopt the following definition.

Definition 9.2. A *quantum algorithm* for the computation of a function $F : \mathbb{B}^* \to \mathbb{B}^*$ is a classical algorithm (i.e., a Turing machine) that computes a function of the form $x \mapsto Z(x)$, where $Z(x)$ is a description of a quantum circuit which computes $F(x)$ on empty input. The function F is said to belong to class BQP if there is a quantum algorithm that computes F in time poly(n).

How does the class BQP relate to the complexity classes introduced earlier?

[3!] **Problem 9.5.** Prove that

$$\mathrm{BPP} \subseteq \mathrm{BQP} \subseteq \mathrm{PP} \subseteq \mathrm{PSPACE}.$$

The class PP consists of predicates of the form

$$Q(x) = \Big(\big|\{y : R_0(x,y)\}\big| < \big|\{y : R_1(x,y)\}\big| \Big),$$

where $R_0, R_1 \in \mathrm{P}$, and y runs through all words of length bounded by some polynomial $q(x)$.

This is almost all that is known about the correspondence between BQP and the other complexity classes. Indirect evidence in favor of the strict inclusion BPP $\subset$ BQP is given by the existence of effective quantum algorithms for some number-theoretic problems traditionally regarded as difficult (see Section 13).

We also remark that there have recently appeared interesting results concerning quantum analogs of some stronger complexity classes (not described in Part 1).

What's next? (A note to the impatient reader). We have spent four sections defining what quantum computation is, but have given only few nontrivial examples so far. The reader may want to see more examples right now. If so, you may skip some material and read Section 13.1 (Simon's algorithm). There will be some references to "mixed states", but all calculations can be done with state vectors as well. However, most other results are based (not as much formally as conceptually) upon the general notion of quantum probability and measurement. We will now proceed to these topics.

10. Quantum probability

10.1. Probability for state vectors. Let us discuss several "physical" aspects of quantum computation. Let a system of n qubits be in the state $|\psi\rangle = \sum_x c_x|x\rangle$. The coefficients of the expansion relative to the classical basis are called amplitudes. The square of the modulus of the amplitude, $|c_x|^2$, equals the probability of finding the system in a given state x (compare with (9.1)). In other words, under a *measurement* of the state of this quantum system, a classical state will be obtained, according to the probability distribution $|c_x|^2$.

The quantity determined by formula (9.1) possesses the basic properties of ordinary probability. The fact that the square of the modulus of the amplitude is the probability of observing the system in state x agrees with the fact that the physical states of quantum mechanics correspond to vectors of unit length, and transformations of these states do not change the length, i.e., they are unitary. Indeed, $\langle\psi|\psi\rangle = \sum_x |c_x|^2 = 1$ (the sum of probabilities equals 1), and the application of physically realizable operators must preserve this relation, i.e., the operator must be unitary.

Formula (9.1) is sufficient for the definition of quantum computation and the class BQP. There are, however, situations for which this definition turns out to be inconvenient or inapplicable. Two fundamental examples are measurement operators and algorithms that are based on them, and the problem of constructing reliable quantum circuits from unreliable gates (error correction).

We therefore give a definition of quantum probability which generalizes both what we observe (the state of the system) and the result of the observation. We will arrive at this general definition by analysing a series of examples. To begin with, we rewrite the expression for the probability

already obtained in the form

$$|c_x|^2 = |\langle\psi|x\rangle|^2 = \langle\psi \overbrace{|x\rangle\langle x|}^{\Pi_x} \psi\rangle,$$

where Π_x denotes the projection to the subspace spanned by $|x\rangle$.

To make the next step toward the general definition of quantum probability, we compute the probability that the first m bits have a given value $y = (y_1, \ldots, y_m)$. Let us represent basis states in the form of two blocks: $x = \overset{m}{\boxed{y}}\overset{n-m}{\boxed{z}}$. We obtain

$$\begin{aligned}\mathbf{P}(|\psi\rangle, y) &= \sum_z \mathbf{P}(|\psi\rangle, (y,z)) = \sum_z \langle\psi\,|y,z\rangle\langle y,z|\,\psi\rangle \\ &= \langle\psi|\left(|y\rangle\langle y| \otimes I\right)|\psi\rangle = \langle\psi|\Pi_{\mathcal{M}}|\psi\rangle.\end{aligned} \tag{10.1}$$

Here $\Pi_{\mathcal{M}}$ denotes the operator of orthogonal *projection* onto the subspace $\mathcal{M} = |y\rangle \otimes \mathcal{B}^{\otimes(n-m)}$. Formula (10.1) gives the definition of quantum probability also in the case where $\mathcal{M}$ is an arbitrary subspace. In this case the projection onto the subspace $\mathcal{M} \subseteq \mathcal{N}$ is given by the formula $\Pi_{\mathcal{M}} = \sum_j |e_j\rangle\langle e_j|$, where e_j runs over an arbitrary orthonormal basis for $\mathcal{M}$.

Remark 10.1. The quantity $\sum_z |\langle F(x), z|U|x, 0^{N-n}\rangle|^2$, which appears in the definition of the evaluation of a function $F : \mathbb{B}^n \to \mathbb{B}^m$ by a quantum circuit (Definition 9.1), equals $\mathbf{P}(U|x, 0^{N-n}\rangle, \mathcal{M})$, where $\mathcal{M} = |F(x)\rangle \otimes \mathcal{B}^{N-m}$. Recall once again the meaning of this definition: the circuit $U = U_L \cdots U_2 U_1$ computes F if for each x the probability to observe the correct result for $F(x)$ after application of the circuit to the initial state $|x, 0^{N-n}\rangle$ is at least $1 - \varepsilon$.

Projections do not represent physically realizable operators; more precisely, they do not describe the evolution of one state of a system to another over a fixed time period. Such evolution is described by unitary operators. Nonetheless, taking some liberty, it is possible to bestow physical meaning on projection. A projection selects a portion of system states from among all possible states. Imagine a filter, i.e., a physical device which passes systems in states belonging to $\mathcal{M}$ but destroys the system if its state is orthogonal to $\mathcal{M}$. (For example, a polarizer does this to photons.) If we submit a system in state $|\psi\rangle$ to the input of such a filter, then the system at the output will be in the state $|\xi\rangle = \Pi_{\mathcal{M}}|\psi\rangle$. The probability associated to this state is generally smaller than one; it is $p = \langle\xi|\xi\rangle = \langle\psi|\Pi_{\mathcal{M}}|\psi\rangle$. The number $1 - p$ determines the probability that the system will not pass through the filter.

Let us compare classical and quantum probability.

Classical probability	**Quantum probability**
Definition	
An event is a subset M of a fixed finite set N.	An event is a subspace $\mathcal{M}$ of some finite-dimensional Hilbert space $\mathcal{N}$.
A probability distribution is given by a function $w\colon N \to R$ with the properties a) $\sum_j w_j = 1$; b) $w_j \geq 0$.	A probability distribution is given by a state vector $\lvert\psi\rangle$, $\langle\psi\lvert\psi\rangle = 1$.
Probability: $\mathbf{Pr}(w, M) = \sum_{j\in M} w_j$.	Probability: $\mathbf{P}(\lvert\psi\rangle, \mathcal{M}) = \langle\psi\lvert\Pi_{\mathcal{M}}\rvert\psi\rangle$.
Properties	
1. If $M_1 \cap M_2 = \varnothing$, then $\mathbf{Pr}(w, M_1 \cup M_2) = \mathbf{Pr}(w, M_1) + \mathbf{Pr}(w, M_2)$.	1^{q}. If $\mathcal{M}_1 \perp \mathcal{M}_2$, then $\mathbf{P}(\lvert\psi\rangle, \mathcal{M}_1 \oplus \mathcal{M}_2) = \mathbf{P}(\lvert\psi\rangle, \mathcal{M}_1) + \mathbf{P}(\lvert\psi\rangle, \mathcal{M}_2)$.
2. (in the general case) $\mathbf{Pr}(w, M_1 \cup M_2) = \mathbf{Pr}(w, M_1) + \mathbf{Pr}(w, M_2) - \mathbf{Pr}(w, M_1 \cap M_2)$.	2^{q}. If $\Pi_{\mathcal{M}_1}\Pi_{\mathcal{M}_2} = \Pi_{\mathcal{M}_2}\Pi_{\mathcal{M}_1}$, then $\mathbf{P}(\lvert\psi\rangle, \mathcal{M}_1 + \mathcal{M}_2) = \mathbf{P}(\lvert\psi\rangle, \mathcal{M}_1) + \mathbf{P}(\lvert\psi\rangle, \mathcal{M}_2) - \mathbf{P}(\lvert\psi\rangle, \mathcal{M}_1 \cap \mathcal{M}_2)$.

Note that the condition $\mathcal{M}_1 \perp \mathcal{M}_2$ (mutually exclusive events) is equivalent to the condition $\Pi_{\mathcal{M}_1}\Pi_{\mathcal{M}_2} = \Pi_{\mathcal{M}_2}\Pi_{\mathcal{M}_1} = 0$.

If we have two nonorthogonal subspaces with zero intersection, the quantum probability is not necessarily additive. We give a simple example where $\mathbf{P}(\lvert\psi\rangle, \mathcal{M}_1 + \mathcal{M}_2) \neq \mathbf{P}(\lvert\psi\rangle, \mathcal{M}_1) + \mathbf{P}(\lvert\psi\rangle, \mathcal{M}_2)$.

Let $\lvert\xi\rangle = \lvert 0\rangle$, $\mathcal{M}_1 = \mathbb{C}\,(\lvert 0\rangle)$ (the linear subspace generated by the vector $\lvert 0\rangle$), $\mathcal{M}_2 = C(\lvert\eta\rangle)$, where $\langle\xi\lvert\eta\rangle$ is close to 1. Then

$$1 = \mathbf{P}(\lvert\xi\rangle, \mathcal{M}_1 + \mathcal{M}_2) \neq \mathbf{P}(\lvert\xi\rangle, \mathcal{M}_1) + \mathbf{P}(\lvert\xi\rangle, \mathcal{M}_2) \approx 1 + 1.$$

10.2. Mixed states (density matrices). Thus, we have defined, in the most general way, what *quantity* we measure. Now we need to generalize what *object* we perform the measurement on. Such objects will be something more general than state vectors or probability distributions. This will give us a definition of probability that generalizes both classical and quantum probability.

Consider a probability distribution on a finite set of quantum states $\{\lvert\xi_1\rangle, \ldots, \lvert\xi_s\rangle\}$. The probability of the state $\lvert\xi_j\rangle$ will be denoted by p_j; clearly $\sum_j p_j = 1$. We will calculate the probability of observing a state in the subspace $\mathcal{M}$:

$$\begin{aligned}\sum_k p_k \mathbf{P}(\lvert\xi\rangle, \mathcal{M}) &= \sum_k p_k \langle\xi_k\lvert\Pi_{\mathcal{M}}\rvert\xi_k\rangle \\ &= \sum_k p_k \operatorname{Tr}\left(\lvert\xi_k\rangle\langle\xi_k\rvert\Pi_{\mathcal{M}}\right) = \operatorname{Tr}(\rho\Pi_{\mathcal{M}}),\end{aligned} \tag{10.2}$$

where ρ denotes the *density matrix*[8] $\rho = \sum_k p_k |\xi_k\rangle\langle\xi_k|$. The final expression in (10.2) is what we take as the general definition of probability.

[1!] **Problem 10.1.** Prove that the operators of the form $\rho = \sum_k p_k |\xi_k\rangle\langle\xi_k|$ are precisely the Hermitian nonnegative operators with trace 1, i.e., operators that satisfy the conditions

$$1)\ \rho = \rho^\dagger; \qquad 2)\ \forall |\eta\rangle\ \langle\eta|\rho|\eta\rangle \geq 0; \qquad 3)\ \operatorname{Tr}\rho = 1.$$

From now on, by a *density matrix* we will mean an arbitrary operator with these properties.

The arguments about the "probability distribution on quantum states" were of an ancillary nature. The problem is how to generalize the notion of a quantum state to include classical probability distributions. The result we have obtained (the last expression in (10.2)) depends only on the density matrix, so that we may postulate that generalized quantum states and density matrices be the same. If a state is given by a density matrix of rank 1 (i.e., $\rho = |\xi\rangle\langle\xi|$), then it is said to be *pure*; if it is given by a general density matrix, it is called *mixed*.

Definition 10.1. For a quantum state given by a density matrix ρ and a subspace $\mathcal{M}$, the *probability of the "event"* $\mathcal{M}$ equals $\mathbf{P}(\rho, \mathcal{M}) = \operatorname{Tr}(\rho\Pi_{\mathcal{M}})$.

Diagonal matrices correspond to classical probability distributions on the set of basis vectors. Indeed, consider the quantum probability associated with the diagonal matrix $\rho = \sum_j w_j |j\rangle\langle j|$ and the subspace $\mathcal{M}$ spanned by a subset of basis vectors M. This probabilitity can also be obtained by the classical formula: $\mathbf{P}(\rho, \mathcal{M}) = \mathbf{Pr}(w, M)$. From the physical point of view, a classical system is a quantum system that supports only diagonal density matrices (see discussion of decoherence in the next section). A state of such a system may be denoted as

$$\rho = \sum_j w_j \cdot (j). \tag{10.3}$$

Mathematically, this is just a different notation of the probability distribution w. It is convenient when we need to simultaneously deal with classical and quantum systems.

Now we continue the comparison of the properties of classical and quantum probability; for the latter we shall now understand the general definition in terms of a density matrix. (Properties 1^{q} and 2^{q} remain valid.)

[8] Actually, this is an operator rather than a matrix, although the term "density matrix" is traditional. In the sequel, we will often have in mind a matrix, i.e., an operator expressed in a particular basis.

Classical probability	Quantum probability
Properties	
3. Suppose a probability distribution of the form $w_{jk} = w_j^{(1)} w_k^{(2)}$ is specified on the set $N = N_1 \times N_2$. Consider two sets of outcomes, $M_1 \subseteq N_1$, $M_2 \subseteq N_2$. Then the probabilities multiply: $\mathbf{Pr}(w, M_1 \times M_2) = \mathbf{Pr}(w^{(1)}, M_1)\ \mathbf{Pr}(w^{(2)}, M_2)$.	3q. Suppose a density matrix of the form $\rho_1 \otimes \rho_2$ is defined on the space $\mathcal{N} = \mathcal{N}_1 \otimes \mathcal{N}_2$. Consider two subspaces, $\mathcal{M}_1 \subseteq \mathcal{N}_1$, $\mathcal{M}_2 \subseteq \mathcal{N}_2$. Then the probabilities likewise multiply: $\mathbf{P}(\rho_1 \otimes \rho_2, \mathcal{M}_1 \otimes \mathcal{M}_2) = \mathbf{P}(\rho_1, \mathcal{M}_1)\,\mathbf{P}(\rho_2, \mathcal{M}_2)$.
4. Consider a joint probability distribution on the set $N_1 \times N_2$. The event we are interested in does not depend on the outcome in the second set, i.e., $M = M_1 \times N_2$. The probability of such an event is expressed by a "projection" of the distribution onto the first set: $\mathbf{Pr}(w, M_1 \times N_2) = \mathbf{Pr}(w', M_1)$, where $w'_j = \sum_k w_{jk}$.	4q. In the quantum case, the restriction to one of the subsystems is described by taking a *partial trace* (see below). Thus, even if the initial state was pure, the resulting state of the subsystem may turn out to be mixed: $\mathbf{P}(\rho, \mathcal{M}_1 \otimes \mathcal{N}_2) = \mathbf{P}(\mathrm{Tr}_{\mathcal{N}_2}\, \rho, \mathcal{M}_1)$.

Definition 10.2. Let $X \in \mathbf{L}(\mathcal{N}_1 \otimes \mathcal{N}_2) = \mathbf{L}(\mathcal{N}_1) \otimes \mathbf{L}(\mathcal{N}_2)$. The *partial trace* of the operator X over the space $\mathcal{N}_2$ is defined as follows: if $X = \sum_m A_m \otimes B_m$, then $\mathrm{Tr}_{\mathcal{N}_2} X = \sum_m A_m (\mathrm{Tr}\, B_m)$.

Due to the universality property of the tensor product (see p. 55), the partial trace does not depend on the choice of summands in the representation $X = \sum_m A_m \otimes B_m$. This may seem somewhat obscure, so we will give a direct proof. Let us choose orthonormal bases in the spaces $\mathcal{N}_1$, $\mathcal{N}_2$ and express the partial trace in terms of the matrix elements $X_{jj'kk'} = \langle j, j' | X | k, k' \rangle$. Let

$$A_m = \sum_{j,k} a_{jk}^m |j\rangle\langle k| \quad \text{and} \quad B_m = \sum_{j',k'} b_{j'k'}^m |j'\rangle\langle k'|.$$

Then

$$X = \sum_{j,j',k,k'} X_{jj'kk'} |j, j'\rangle\langle k, k'| = \sum_m A_m \otimes B_m = \sum_{j,j',k,k',m} a_{jk}^m b_{j'k'}^m |j, j'\rangle\langle k, k'|,$$

so the partial trace equals

$$\mathrm{Tr}_{\mathcal{N}_2} X = \sum_m \sum_{j,k} a_{jk}^m \Big(\sum_l b_{ll}^m \Big) |j\rangle\langle k| = \sum_{j,k} \sum_l X_{jlkl} |j\rangle\langle k|.$$

Let us consider an example where taking the partial trace of the density matrix corresponding to a pure state leads to the density matrix corresponding to a mixed state.

Let $\mathcal{N}_1 = \mathcal{N}_2 = \mathcal{B}$ and $\rho = |\psi\rangle\langle\psi|$, where $|\psi\rangle = \frac{1}{\sqrt{2}}(|0,0\rangle + |1,1\rangle)$. In this case $\rho = \frac{1}{2}\sum_{a,b}|a,a\rangle\langle b,b|$, thus we obtain

$$\mathrm{Tr}_{\mathcal{N}_2}\,\rho = \frac{1}{2}\sum_a |a\rangle\langle a| = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}.$$

This matrix corresponds to a mixed state (pure states correspond to matrices of rank 1). Moreover, this mixed state is equivalent to a classical probability distribution: 0 and 1 have probabilities 1/2. Thus, discarding the second qubit yields a purely classical probability distribution on the first qubit.

Proposition 10.1. *An arbitrary mixed state $\rho \in \mathbf{L}(\mathcal{N})$ can be represented as the partial trace $\mathrm{Tr}_{\mathcal{F}}(|\psi\rangle\langle\psi|)$ of a pure state of a larger system, $|\psi\rangle \in \mathcal{N}\otimes\mathcal{F}$. Such $|\psi\rangle$ is called a* purification *of ρ. (We may assume that $\dim\mathcal{F} = \dim\mathcal{N}$.)*

Proof. Set $\mathcal{F} = \mathcal{N}^*$. Since ρ is a nonnegative (= positive semidefinite) Hermitian operator, there exists $\sqrt{\rho} \in \mathbf{L}(\mathcal{N}) = \mathcal{N}\otimes\mathcal{N}^*$. More explicitly, let us choose an orthonormal basis $\{|\xi_j\rangle\}$ in which ρ is diagonal, i.e., $\rho = \sum_j p_j|\xi_j\rangle\langle\xi_j|$. Then $\sqrt{\rho} = \sum_j \sqrt{p_j}|\xi_j\rangle\langle\xi_j|$.

Let us regard $\sqrt{\rho}$ as a vector of the space $\mathcal{N}\otimes\mathcal{N}^*$:

$$|\sqrt{\rho}\rangle = |\psi\rangle = \sum_j \sqrt{p_j}|\xi_j\rangle\otimes|\eta_j\rangle, \quad \text{where } |\eta_j\rangle = \langle\xi_j| \in \mathcal{N}^*.$$

This vector satisfies the desired requirements, i.e., $\mathrm{Tr}_{\mathcal{F}}(|\psi\rangle\langle\psi|) = \rho$. Indeed,

$$|\psi\rangle\langle\psi| = \sum_{jk}\sqrt{p_jp_k}\big(|\xi_j\rangle\otimes|\eta_j\rangle\big)\big(\langle\xi_k|\otimes\langle\eta_k|\big).$$

Only terms with $j = k$ contribute to the partial trace. Therefore

$$\mathrm{Tr}_{\mathcal{N}^*}(|\psi\rangle\langle\psi|) = \sum_j p_j|\xi_j\rangle\langle\xi_j| = \rho.$$

□

[2!] **Problem 10.2.** Consider a pure state $|\psi\rangle \in \mathcal{N}\otimes\mathcal{F}$. Show that the so-called *Schmidt decomposition* holds:

$$|\psi\rangle = \sum_j \lambda_j|\xi_j\rangle\otimes|\eta_j\rangle,$$

where $0 < \lambda_j \le 1$, and the sets of vectors $\{|\xi_j\rangle\} \subset \mathcal{N}$ and $\{|\eta_j\rangle\} \subset \mathcal{F}$ are orthonormal.

Note that the numbers λ_j^2 are the nonzero eigenvalues of the partial traces $\rho = \mathrm{Tr}_{\mathcal{F}}(|\psi\rangle\langle\psi|)$ and $\rho' = \mathrm{Tr}_{\mathcal{N}}(|\psi\rangle\langle\psi|)$. (Hence the nonzero eigenvalues of ρ and ρ' coincide.) The number of such eigenvalues equals the rank of ρ and ρ'. For example, if $\mathrm{rank}(\rho) = 1$, the Schmidt decomposition consists

of one term, and vice versa. Thus the state $\rho = \mathrm{Tr}_{\mathcal{F}}(|\psi\rangle\langle\psi|)$ is pure if and only $|\psi\rangle$ is a product state, i.e., $|\psi\rangle = |\xi\rangle \otimes |\eta\rangle$. In general, $\mathrm{rank}(\rho)$ is the smallest dimension of the auxiliary space $\mathcal{F}$ which allows a purification of ρ.

[2!] **Problem 10.3** ("Purification is unique up to unitary equivalence"). Let $|\psi_1\rangle, |\psi_2\rangle \in \mathcal{N} \otimes \mathcal{F}$ be two pure states such that $\mathrm{Tr}_{\mathcal{F}}(|\psi_1\rangle\langle\psi_1|) = \mathrm{Tr}_{\mathcal{F}}(|\psi_2\rangle\langle\psi_2|)$. Prove that $|\psi_2\rangle = (I_{\mathcal{N}} \otimes U)|\psi_1\rangle$ for some unitary operator U on the space $\mathcal{F}$.

10.3. Distance functions for density matrices. In practice, various mixed states are always specified with some precision, so we need to somehow measure "distance" between density matrices. What would the most natural definition of this distance be? To begin with, let us ask the same question for probability distributions.

Let w be the probability distribution of an outcome produced by some device. Suppose that the device is faulty, i.e., with some probability ε it goes completely wrong, but with probability $1 - \varepsilon$ it works as expected. What can one tell about the actual probability distribution w' of the outcome of such a device? The answer is

$$\sum_j |w'_j - w_j| \le 2\varepsilon. \tag{10.4}$$

Conversely, if the inequality (10.4) is true, we can represent w' as the probability distribution produced by a pipeline of two processes: the first generates j according to the distribution w, whereas the second alters j with total probability $\le \varepsilon$. We conclude that the natural distance between probability distributions is given by the ℓ^1 norm, $\|w - w'\|_1 = \sum_j |w'_j - w_j|$. Now we will generalize this definition to arbitrary density matrices.

Definition 10.3. The *trace norm* of an operator $A \in \mathbf{L}(\mathcal{N})$ is

$$\|A\|_{\mathrm{tr}} = \mathrm{Tr}\left(\sqrt{A^\dagger A}\right). \tag{10.5}$$

For Hermitian operators, the trace norm is the sum of the moduli of the eigenvalues.

[2] **Problem 10.4.** Verify that (10.5) actually defines a norm. Prove that

$$\|A\|_{\mathrm{tr}} = \sup_{B \neq 0} \frac{|\operatorname{Tr} AB|}{\|B\|} = \max_{U \in \mathbf{U}(\mathcal{N})} |\operatorname{Tr} AU|, \tag{10.6}$$

$$\|A\|_{\mathrm{tr}} = \inf\left\{ \sum_k \big\||\xi_k\rangle\big\| \, \big\||\eta_k\rangle\big\| \;:\; \sum_k |\xi_k\rangle\langle\eta_k| = A \right\} \tag{10.7}$$

($\|X\|$ denotes the operator norm; cf. Definition 8.2 on p. 71).

[1] **Problem 10.5.** Verify that the trace norm has the following properties:
a) $\|AB\|_{\rm tr}\,, \|BA\|_{\rm tr} \le \|A\|_{\rm tr}\|B\|$, b) $|\operatorname{Tr} A| \le \|A\|_{\rm tr}$,
c) $\|\operatorname{Tr}_{\mathcal M} A\|_{\rm tr} \le \|A\|_{\rm tr}$, d) $\|A\otimes B\|_{\rm tr} = \|A\|_{\rm tr}\|B\|_{\rm tr}$.

The following lemma shows why the trace norm for density matrices can be regarded as the analogue of the ℓ^1-norm for probability distributions.

Lemma 10.2. *Let $\mathcal N = \bigoplus_j \mathcal N_j$ be a decomposition of $\mathcal N$ into the direct sum of mutually orthogonal subspaces. Then for any pair of density matrices ρ and γ,*

$$\sum_j |\mathbf P(\rho,\mathcal N_j) - \mathbf P(\gamma,\mathcal N_j)| \le \|\rho-\gamma\|_{\rm tr}.$$

Proof. The left-hand side of the inequality can be represented in the form $\operatorname{Tr}((\rho-\gamma)U)$, where $U = \sum_j(\pm\Pi_{\mathcal N_j})$. It is clear that U is unitary. We then apply the representation of the trace norm in the form (10.6). □

There is another commonly used distance function on density matrices, called the fidelity distance. Let $\rho,\gamma\in\mathbf L(\mathcal N)$. Consider all possible purifications of ρ and γ over an auxiliary space $\mathcal F$ of dimension $\dim\mathcal F = \dim\mathcal N$; these are pure states $|\xi\rangle, |\eta\rangle \in \mathcal N\otimes\mathcal F$. Then the *fidelity distance* between ρ and γ is

$$d_F(\rho,\gamma) \stackrel{\rm def}{=} \min\Big\{\big\||\xi\rangle - |\eta\rangle\big\| \;:\; \operatorname{Tr}_{\mathcal F}(|\xi\rangle\langle\xi|) = \rho,\ \operatorname{Tr}_{\mathcal F}(|\eta\rangle\langle\eta|) = \gamma\Big\}.$$

It is related to a quantity called *fidelity*:

$$F(\rho,\gamma) \stackrel{\rm def}{=} \max\Big\{\big|\langle\xi|\eta\rangle\big|^2 \;:\; \operatorname{Tr}_{\mathcal F}(|\xi\rangle\langle\xi|) = \rho,\ \operatorname{Tr}_{\mathcal F}(|\eta\rangle\langle\eta|) = \gamma\Big\}. \tag{10.8}$$

(One can show that the condition $\dim\mathcal F = \dim\mathcal N$ in these definitions can be relaxed: it is sufficient to require that $\dim\mathcal F \ge \max\{\operatorname{rank}(\rho),\operatorname{rank}(\gamma)\}$. Thus, any auxiliary space $\mathcal F$ will do, as long as it allows purifications of ρ and γ.)

Problem 10.6. Prove that

[1] a) $d_F(\rho,\gamma) = \sqrt{2\left(1-\sqrt{F(\rho,\gamma)}\right)}$;

[2] b) $F(\rho,\gamma) = \left\|\sqrt\rho\sqrt\gamma\right\|_{\rm tr}^2$;

[3] c) $\left(1-\dfrac{\|\rho-\gamma\|_{\rm tr}}{2}\right)^2 \le F(\rho,\gamma) \le 1-\left(\dfrac{\|\rho-\gamma\|_{\rm tr}}{2}\right)^2$.

11. Physically realizable transformations of density matrices

In this section we introduce a formalism for the description of irreversible quantum processes. We will not use it in full generality (so some of the results are superfluous), but the basic concepts and examples will be helpful.

11.1. Physically realizable superoperators: characterization. All transformations of density matrices we will encounter can be represented by linear maps between operator spaces, $\mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{M})$. A general linear map of this type is called a *superoperator*. We now describe those superoperators that are admissible from the physical point of view.

1. A unitary operator takes the density matrix of a pure state $\rho = |\xi\rangle\langle\xi|$ to the matrix $\rho' = U|\xi\rangle\langle\xi|U^\dagger$. It is natural to assume (by linearity) that such a formula also yields the action of a unitary operator on an arbitrary density matrix:

$$\rho \overset{U}{\longmapsto} U\rho U^\dagger.$$

2. A second type of transformation is the operation of taking the partial trace. If $\rho \in \mathbf{L}(\mathcal{N}\otimes\mathcal{F})$, then the operation of *discarding the second subsystem* is described by the superoperator

$$\mathrm{Tr}_{\mathcal{F}} : \ \rho \mapsto \mathrm{Tr}_{\mathcal{F}}\, \rho.$$

3. We recall that it has been useful to us to borrow qubits in the state $|0\rangle$. Let the state $\rho \in \mathbf{L}(\mathcal{B}^{\otimes n})$. We consider the isometric (preserving the inner product) embedding $V : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes N}$ in a space of larger dimension, given by the formula $|\xi\rangle \overset{V}{\longmapsto} |\xi\rangle \otimes |0^{N-n}\rangle$. The density matrix ρ is transformed thereby into $\rho\otimes|0^{N-n}\rangle\langle 0^{N-n}|$. For any isometric embedding V we similarly obtain a superoperator $V \cdot V^\dagger$ that acts as follows:

$$V \cdot V^\dagger : \ \rho \mapsto V\rho V^\dagger.$$

We postulate that a *physically realizable superoperator* is a composition of an arbitrary number of transformations of types 2 and 3 (type 1 is a special case of 3).

[3!] **Problem 11.1.** Prove that a superoperator T is physically realizable if and only if it has the form

$$T = \mathrm{Tr}_{\mathcal{F}}(V \cdot V^\dagger) \ : \ \rho \mapsto \mathrm{Tr}_{\mathcal{F}}(V\rho V^\dagger), \tag{11.1}$$

where $V : \mathcal{N} \to \mathcal{N} \otimes \mathcal{F}$ is an isometric embedding.

[2!] **Problem 11.2** ("Operator sum decomposition"). Prove that a superoperator T is physically realizable if and only if it can be represented in the form

$$T = \sum_m A_m \cdot A_m^\dagger \ : \ \rho \mapsto \sum_m A_m \rho A_m^\dagger, \qquad \text{where } \sum_m A_m^\dagger A_m = I. \tag{11.2}$$

The operation of taking the partial trace means forgetting (discarding) one of the subsystems. We show that such an interpretation is reasonable, specifically that the subsequent fate of the discarded system in no way influences the quantities characterizing the remaining system. Let us take a system consisting of two subsystems, which is in some state $\rho \in \mathbf{L}(\mathcal{N} \otimes \mathcal{F})$. If we discard the second subsystem (to the trash), then it will be subjected to uncontrollable influences. Suppose we apply some operator U to the first subsystem. We will then obtain a state $\gamma = (U \otimes Y)\rho(U \otimes Y)^\dagger$, where Y is an arbitrary unitary operator (the action of the trash bin on the trash). If we wish to find the probability for some subspace $\mathcal{M} \subseteq \mathcal{N}$ pertaining to the first subsystem (the trash doesn't interest us), then the result does not depend on Y and equals

$$\mathbf{P}(\gamma, \mathcal{M} \otimes \mathcal{F}) = \mathbf{P}(\operatorname{Tr}_\mathcal{F} \gamma, \mathcal{M}) = \mathbf{P}\big(U(\operatorname{Tr}_\mathcal{F} \rho)U^\dagger, \mathcal{M}\big).$$

Here the first equality is the property 4^{q} of quantum probability, whereas the second equality represents a new property:

$$\operatorname{Tr}_\mathcal{F}\big((U \otimes Y)\rho(U \otimes Y)^\dagger\big) = U(\operatorname{Tr}_\mathcal{F} \rho)U^\dagger. \tag{11.3}$$

[1] **Problem 11.3.** Prove the identity (11.3).

[2!] **Problem 11.4.** Let us write a superoperator $T : \mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{M})$ in the coordinate form:

$$T\big(|j\rangle\langle k|\big) = \sum_{j',k'} T_{(j'j)(k'k)} |j'\rangle\langle k'|.$$

Prove that the physical realizability of T is equivalent to the set of three conditions:

a) $\sum_l T_{(lj)(lk)} = \delta_{jk}$ (Kronecker symbol);

b) $T^*_{(j'j)(k'k)} = T_{(k'k)(j'j)}$;

c) The matrix $\big(T_{(j'j)(k'k)}\big)$ is nonnegative (each of the index pairs is regarded as a single index).

[3!] **Problem 11.5.** Prove that a superoperator $T : \mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{M})$ is physically realizable if and only if it satisfies the following three conditions:

a) $\operatorname{Tr}(TX) = \operatorname{Tr} X$ for any $X \in \mathbf{L}(\mathcal{N})$;

b) $(TX)^\dagger = TX^\dagger$ for any $X \in \mathbf{L}(\mathcal{N})$;

c) T is completely positive. Namely, for any additional space $\mathcal{G}$ the superoperator $T \otimes I_{\mathbf{L}(\mathcal{G})} : \mathbf{L}(\mathcal{N} \otimes \mathcal{G}) \to \mathbf{L}(\mathcal{M} \otimes \mathcal{G})$ maps nonnegative operators to nonnegative operators.

11.2. Calculation of the probability for quantum computation. Now, since we have the general definitions of quantum probability and of a physically realizable transformation of density matrices, there are two ways to calculate the probability that enters the definition of quantum computation. Suppose we use a supplementary subsystem. After we no longer need it, we can discard it to the trash and, in counting the probability, take the partial trace over the state space of the supplementary subsystem. Or else we may hold all the trash until the very end and consider the probability of an event of the form $\mathcal{M}_1 \otimes \mathcal{N}_2$ (once we have stopped using the second subsystem, no details of its existence are of any importance to us and we are not interested in what precisely happens to it in the trash bin). As already stated, these probabilities are equal: $\mathbf{P}(\rho, \mathcal{M}_1 \otimes \mathcal{N}_2) = \mathbf{P}(\mathrm{Tr}_{\mathcal{N}_2} \rho, \mathcal{M}_1)$.

Remark 11.1. It is not difficult to define a more general model of quantum computation in which suitable physically realizable superoperators (not necessarily corresponding to unitary operators) serve as the elementary gates. Such a model of *computation with mixed states* is more adequate in the physical situation where the quantum computer interacts with the surrounding environment. In particular, one can consider things like combination of classical and quantum computation. From the complexity point of view, the new model is polynomially equivalent to the standard one, if a complete basis is used in both cases. (Completeness in the new model is most comprehensively defined as the possibility to effect arbitrary unitary operators on "encoded qubits"; cf. Remark 8.2.) We also note that in the model of computation with mixed states a more natural definition of a probabilistic subroutine is possible. We will not give this definition here, but refer interested readers to [**4**].

11.3. Decoherence. The term "decoherence" is generally used to denote irreversible degradation of a quantum state caused by its interaction with the environment. This could be an arbitrary physically realizable superoperator that takes pure states to mixed states. For the purpose of our discussion, *decoherence* means the specific superoperator D that "forgets" off-diagonal matrix elements:

$$\rho = \sum_{j,k} \rho_{jk} |j\rangle\langle k| \ \stackrel{D}{\longmapsto} \ \sum_{k} \rho_{kk} |k\rangle\langle k|.$$

This superoperator is also known as an extreme case of a "phase damping channel". We will show that it is physically realizable. For simplicity, let us assume that D acts on a single qubit.

The action of D on a density matrix ρ can be performed in three steps. First, we append a null qubit:

$$\rho \mapsto \rho \otimes |0\rangle\langle 0|.$$

Then we "copy" the original qubit into the ancilla. This can be achieved by applying the operator $\Lambda(\sigma^x) : |a, b\rangle \mapsto |a, a \oplus b\rangle$. We get

$$\rho \otimes |0\rangle\langle 0| \xmapsto{\Lambda(\sigma^x)} \sum_k \rho_{jk}|j, j\rangle\langle k, k|.$$

Finally, we take the partial trace over the ancilla, which yields the diagonal matrix

$$\sum_k \rho_{kk}|k\rangle\langle k|.$$

Warning. The "copying operation" we considered:

$$|j\rangle \mapsto |j, j\rangle, \qquad \sum_{j,k} \rho_{jk}|j\rangle\langle k| \mapsto \sum_k \rho_{jk}|j, j\rangle\langle k, k|$$

(the composition of the first two transformations) in fact copies only the basis states. We note that the copying of an *arbitrary* quantum state $|\xi\rangle \mapsto |\xi\rangle \otimes |\xi\rangle$ is a nonlinear operator and so cannot be physically realized. (This statement is called a "no-cloning theorem".) We will take the liberty of calling the operator of the form

$$\sum_j c_j|\eta_j\rangle \mapsto \sum_j c_j|\eta_j\rangle \otimes |\eta_j\rangle$$

copying relative to the orthonormal basis $\{|\eta_j\rangle\}$.

So, the decoherence superoperator D translates any state into a classical one (with diagonal density matrix) by copying qubits. This can be interpreted as follows: *if we constantly observe a quantum system (make copies), then the system will behave like a classical one.* Thus the copying operation, together with "forgetting" about the copy (i.e., the partial trace), provides a conceptual link between quantum mechanics and the classical picture of the world.

Remark 11.2 (Decoherence in physics). In Nature, decoherence by "copying to the environment" is very common and, of course, does not require a human observer. Let us consider one famous example of quantum phenomenon — an interference pattern formed by a single photon. It is known from classical optics that a light beam passing through two parallel slits forms a pattern of bright and dark stripes on a screen placed behind the slits. This pattern can be recorded if one uses a photographic film as the screen. When the light is dim, the photographic image consists of random dots produced by individual photons (i.e., quanta of light). The

probability for a dot to appear at a given position[9] x is the probability that a photon hits the film at the point x. What will happen if the light is so dim that only one photon reaches the film? Quantum mechanics predicts that the photon arrives in a certain superposition $|\psi\rangle = \sum_x c_x|x\rangle$, so the above probability equals $|c_x|^2$. Thus, the quantum state of the photon is transformed into a classical object — a dot, located at a particular place (although the appearance of the dot at a given position x occurs with the probability related to the corresponding amplitude c_x). When and how does this transition happen?

When the photon hits the film, it breaks a chemical bond and generates a defect in a light-sensitive grain (usually, a small crystal of silver compound). The photon is delocalized in space, so a superposition of states with the defect located at different places is initially created. Basically, this is the same state $|\psi\rangle = \sum_x c_x|x\rangle$, but x now indicates the position of the defect. The transition from the quantum state $|\psi\rangle$ to the classical state $\sum_x |c_x|^2|x\rangle\langle x|$ is decoherence. It occurs long before anyone sees the image, even before the film is developed. About every 10^{-12} seconds since the defect was created, it scatters a phonon (a quantum of sonic vibrations). This has the effect of "copying" the state of the defect to phonon degrees of freedom relative to the position basis.

The above explanation is based on the assumption that the phonon scattering (or whatever causes the decoherence) is irreversible. But what does this assumption mean if the scattering is just a unitary process which occurs in the film? In the preceding mathematical discussion, the irreversible step was the partial trace; it was justified by the fact that the copy was "discarded", i.e., never used again. On the contrary, the scattered phonon stays around and can, in principle, scatter back to "undo the copying". In reality, however, the scattering does not reverse by itself. One reason is that the phonons interact with other phonons, causing the "copies" to multiply. The quantum state quickly becomes so entangled that it cannot be disentangled. (Well, this argument is more empirical than logical; it basically says that things can be lost and never found — a true fact with no "proof" whatsoever. For some particular classes of Hamiltonians, some assertions about irreversibility, like "information escapes to infinity", can be formulated mathematically. Proving this kind of statements is a difficult and generally unsolved problem.)

[9] We think of the position on the film as a discrete variable; specifically, it refers to a grain of light-sensitive substance. The whole grain either turns dark (if it caught a photon) or stays white when the film is developed. Speaking about photons, we have oversimplified the situation for illustrative purposes. In modern films, a single photon does not yet produce a sufficient change to be developed, but several (3 or more) photons per grain do. For single photon detection, physicists use other kinds of devices, e.g., ones based on semiconductors.

Some irreversibility postulate or assumption is necessary to give an *interpretation of quantum mechanics*, i.e., to introduce a classical observer. It seems that the exact nature of irreversibility is the real question behind many philosophical debates surrounding quantum mechanics. Another thing that is not fully understood, is the meaning of probability in the physical world. Both problems exist in classical physics as well; quantum mechanics just makes them more evident.

Fortunately (especially to mathematicians), the theory of quantum computation deals with an ideal world where nothing gets lost. If we observe something, we can also "un-observe", unless we explicitly choose to discard the result or to keep it as the computation output. As far as probabilities are concerned, we deal with them formally rather than trying to explain them by something else. *[End of remark]*

In the case of a single qubit, the decoherence superoperator (the off-diagonal matrix elements set to zero) can be also obtained if we apply the operator σ^z with probability $1/2$:

$$\rho \mapsto \frac{1}{2}\rho + \frac{1}{2}\sigma^z \rho \sigma^z.$$

Such a process is called random dephasing: the state $|1\rangle$ is multiplied by the phase factor -1 with probability $1/2$. Thus, the dephasing leads likewise to the situation that the system behaves classically.

[2!] **Problem 11.6.** Suppose we have a physically realizable superoperator $T : \mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{N} \otimes \mathcal{F})$ with the following property: $\mathrm{Tr}_{\mathcal{F}}(T\rho) = \rho$ for any pure state ρ. Prove that $TX = X \otimes \gamma$ (for any operator X), where γ is a fixed density matrix on the space $\mathcal{F}$.

Think of $\mathcal{N}$ as a system one wants to observe, and $\mathcal{F}$ as an "observation record". Then the condition $\mathrm{Tr}_{\mathcal{F}}(T\rho) = \rho$ indicates that the superoperator T does not perturb the system, whereas $TX = X \otimes \gamma$ means that the obtained record γ does not carry any information about ρ. Thus, *it is impossible to get any information about an unknown quantum state without perturbing the state.*

11.4. Measurements. In describing quantum algorithms, it is often natural (though not mathematically necessary) to assume that, together with a quantum computational system, a classical one might also be used. An important type of interaction between quantum and classical parts is *measurement* of a quantum register. It yields a classical "record" (outcome), while the quantum register may remain in a modified state, or may be destroyed.

Consider a system consisting of two parts, a quantum part ($\mathcal{N}$) and a classical part ($\mathcal{K}$). The density matrix is diagonal with respect to the classical indices, i.e.,

$$\rho = \sum_{j,k,l} \rho_{jkll}(|j\rangle\langle k|) \otimes (|l\rangle\langle l|) = \sum_l w_l \gamma^{(l)} \otimes |l\rangle\langle l|,$$

where $w_l = \sum_j \rho_{jjll}$ is the probability of having the classical state l, and the operator $\gamma^{(l)} = w_l^{-1} \sum_{j,k} \rho_{jkll}$ possesses all the properties of a density matrix. In this manner, quantum-classical states are always decomposed into "conditional" (by analogy with conditional probability) density matrices $\gamma^{(l)}$. In such a case we will use in such a case a special notation similar to that of equation. (10.3): $\rho = \sum_l w_l \cdot (\gamma^{(l)}, l) = \sum_l (w_l \gamma^{(l)}, l)$. (Here the dot does not have special meaning as, e.g., in (11.1); it just indicates that w_l is a factor rather than a function.)

Suppose we have a set of mutually exclusive possibilities, which is expressed as a decomposition of the state space into a direct sum of pairwise orthogonal subspaces, $\mathcal{N} = \bigoplus_{j\in\Omega} \mathcal{L}_j$, where $\Omega = \{1, \dots, r\}$ is the set of corresponding classical outcomes. (We say "mutually exclusive" because, if a subspace $\mathcal{L}'$ is orthogonal to a subspace $\mathcal{L}''$, and $\rho \in \mathbf{L}(\mathcal{L}')$, then $\mathbf{P}(\rho, \mathcal{L}'') = 0$.)

A transformation of density matrices, that we will call a *projective measurement*, is such that for states of the subspace $\mathcal{L}_j$, a "measuring device" puts the number of the state j into the classical register:

$$\text{if } |\xi\rangle \in \mathcal{L}_j, \text{ then } |\xi\rangle\langle\xi| \mapsto (|\xi\rangle\langle\xi|, j). \tag{11.4}$$

Although the measurement maps the space $\mathbf{L}(\mathcal{N})$ to $\mathbf{L}(\mathcal{N} \otimes \mathcal{K})$, the result is always diagonal with respect to the classical basis in $\mathcal{K}$. Therefore we can assume that the measurement is a linear map $R : \mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{N}) \times \{1, \dots, r\} = \bigoplus_{j=1}^r \mathbf{L}(\mathcal{N})$; such linear maps will also be called superoperators.

By linearity, equation (11.4) implies that $R\rho = (\rho, j)$ for any $\rho \in \mathbf{L}(\mathcal{L}_j)$. However, to define the action of R on an arbitrary ρ, we have to use the condition that R is physically realizable.

[3] **Problem 11.7.** Prove that the superoperator

$$R : \ \rho \mapsto \sum_j \big(\Pi_{\mathcal{L}_j} \rho \Pi_{\mathcal{L}_j}, j\big) \tag{11.5}$$

is the only physically realizable superoperator of the type $\mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{N}) \times \{1, \dots, r\}$ that is consistent with (11.4).

Thus we arrive at our final definition.

Definition 11.1. A *projective measurement* is a superoperator of the form (11.5), which can also be written as follows:

$$\rho \mapsto \sum_j \mathbf{P}(\rho, \mathcal{L}_j) \cdot \big(\gamma^{(j)}, j\big), \tag{11.6}$$

where $\gamma^{(j)} = \dfrac{\Pi_{\mathcal{L}_j} \rho \Pi_{\mathcal{L}_j}}{\mathbf{P}(\rho, \mathcal{L}_j)}$.

We may say that $\mathbf{P}(\rho, \mathcal{L}_j)$ is the probability of getting a specified outcome j. If the outcome j is actually obtained, the state of the quantum system after the measurement is $\gamma^{(j)}$. If we measure pure states, i.e., if $\rho = |\xi\rangle\langle\xi|$, then $\gamma^{(j)} = |\eta_j\rangle\langle\eta_j|$, where

$$|\eta_j\rangle = \frac{\Pi_{\mathcal{L}_j}|\xi\rangle}{\sqrt{\mathbf{P}(|\xi\rangle, \mathcal{L}_j)}}.$$

Let us give a simple example of a measurement. We copy a qubit (relative to the classical basis) and apply the decoherence superoperator to the copy. In this example, $\Pi_{\mathcal{L}_0} = |0\rangle\langle 0|$, $\Pi_{\mathcal{L}_1} = |1\rangle\langle 1|$, and the measurement superoperator is

$$\rho = \begin{pmatrix} \rho_{00} & \rho_{01} \\ \rho_{10} & \rho_{11} \end{pmatrix} \mapsto \rho_{00} \cdot \Big(|0\rangle\langle 0|, 0\Big) + \rho_{11} \cdot \Big(|1\rangle\langle 1|, 1\Big).$$

We have considered nondestructive measurements. A *destructive measurement* can be described as a nondestructive one after which the measured system is discarded. This corresponds to the transition from a quantum state ρ to the classical state $\sum_j \mathbf{P}(\rho, \mathcal{L}_j) \cdot (j)$ (where (j) is a short notation for $|j\rangle\langle j|$). A general physically realizable transformation of a quantum state to a classical state is given by the formula

$$\rho \mapsto \sum_k \operatorname{Tr}(\rho X_k) \cdot (k), \tag{11.7}$$

where X_k are nonnegative Hermitian operators satisfying $\sum_k X_k = I$. Such a set of operators $\{X_k\}$ is called a POVM (positive operator-valued measure), whereas the superoperator (11.7) is called a *POVM measurement*.

Remark 11.3. Nondestructive POVM measurements could also be defined, but there is no such definition that would be natural enough. An exception is the case where the operators X_k commute with each other. Then they can be represented as linear combination of projections Π_j with nonnegative coefficients, which can be interpreted as "conditional probabilities". Such measurements (in the nondestructive form) and their realizations will be studied in the next section.

[2!] **Problem 11.8.** Prove that any POVM measurement can be represented as an isometric embedding into a larger space, followed by a projective measurement.

[3!] **Problem 11.9** ("Quantum teleportation"; cf. [**4**]). Suppose we have three qubits: the first is in an arbitrary state ρ (not known in advance), whereas the second and third are in the state $|\xi_{00}\rangle = \frac{1}{\sqrt{2}}(|0,0\rangle + |1,1\rangle)$. On the first two qubits we perform the measurement corresponding to the orthogonal decomposition

$$\mathcal{B}^{\otimes 2} = \mathbb{C}\,(|\xi_{00}\rangle) \oplus \mathbb{C}(|\xi_{01}\rangle) \oplus \mathbb{C}(|\xi_{10}\rangle) \oplus \mathbb{C}(|\xi_{11}\rangle),$$

where

$$|\xi_{ab}\rangle = \frac{1}{\sqrt{2}} \sum_c (-1)^{bc} |c, c \oplus a\rangle.$$

Find a way to restore the initial state ρ from the remaining third qubit and the measurement outcome (a, b). Write the whole sequence of actions (the measurement and the recovery) in the form of a quantum circuit.

(Informally, this procedure can be described as follows. Suppose that Alice wants to transmit to Bob[13] a quantum state ρ by a classical communication channel, e.g., over the phone. It turns out that this is possible, provided Alice and Bob have prepared in advance the state $|\xi_{00}\rangle$ so that each of them keeps half of the state, i.e., a single qubit. Alice performs the measurement and tells the result to Bob. Then Bob translates his qubit to the state ρ. Thus the unknown state gets "teleported".)

11.5. The superoperator norm. At first sight, it seems natural to define a norm for superoperators of type $\mathbf{L}(\mathcal{N}, \mathcal{M})$ by analogy with the operator norm,

$$\|T\|_1 = \sup_{X \neq 0} \frac{\|TX\|_{\mathrm{tr}}}{\|X\|_{\mathrm{tr}}}, \tag{11.8}$$

and to use this norm to measure distance between physically realizable transformations of density matrices. (Of course, the norm is applied to the difference between two physically realizable superoperators, which is not physically realizable.) However, the use of the norm (11.5) turns out to be inconvenient because it is "unstable" with respect to the tensor product. Let us explain this in more detail.

Suppose we want to characterize the distance between physically realizable superoperators $P, R \in \mathbf{L}(\mathcal{N}, \mathcal{M})$. From the physical point of view, both superoperators pertain to some quantum system, which is a part of the Universe. Certainly, we do not expect that the answer to our problem would

[13] These two characters are encountered in almost every paper on quantum information theory.

depend on what happens in some other galaxy, and even on the existence of that galaxy. In other words, we expect that the distance between P and R is the same as the distance between $P \otimes I_{\mathbf{L}(\mathcal{G})}$ and $R \otimes I_{\mathbf{L}(\mathcal{G})}$, whatever additional space $\mathcal{G}$ we choose. But this is not always the case if we use $\|P - R\|_1$ as the distance function.

Example 11.1. Consider the superoperator which transposes a matrix,

$$T : |j\rangle\langle k| \mapsto |k\rangle\langle j| \quad (j,k = 0,1).$$

It is obvious that $\|T\|_1 = 1$. However, $\left\|T \otimes I_{\mathbf{L}(\mathcal{B})}\right\|_1 = 2$. Indeed, let the superoperators $T \otimes I_{\mathbf{L}(\mathcal{B})}$ act on the operator $X = \sum_{j,k} |j,j\rangle\langle k,k|$; then $\|X\|_{\mathrm{tr}} = 2$ but $\left\|(T \otimes I_{\mathbf{L}(\mathcal{B})})X\right\|_{\mathrm{tr}} = 4$. Hence $\left\|T \otimes I_{\mathbf{L}(\mathcal{B})}\right\|_1 \geq 2$. The upper bound $\left\|T \otimes I_{\mathbf{L}(\mathcal{B})}\right\|_1 \leq 2$ is also easily obtained.

One may argue that this is not quite "bad" a counterexample yet, since T does not have the form $P-R$, where P and R are physically realizable (for example, because $\mathrm{Tr}(TI_{\mathcal{B}}) \neq 0$). Consider, however, another superoperator,

$$Q : \rho \mapsto (T\rho) \otimes \sigma^z.$$

It is easy to see that $\|Q\|_1 = \|T\|_1\|\sigma^z\|_{\mathrm{tr}} = 2$; similarly, $\left\|Q \otimes I_{\mathbf{L}(\mathcal{B})}\right\|_1 = 4$. The superoperator Q satisfies the following two conditions:

$$\text{a) } \mathrm{Tr}(QX) = 0, \qquad \text{and} \qquad \text{b) } (QX)^\dagger = QX^\dagger \qquad (\text{for any } X).$$

It is possible to show (using the result of Problem 11.4) that any superoperator with these properties can be represented as $c(P - R)$, where P and R are physically realizable, and c is a positive real number.

Fortunately, it turns out (as we will prove below) that the pathology of Example 11.1 has a restriction by dimension. Specifically, if $\dim \mathcal{G} \geq \dim \mathcal{N}$, then $\|T \otimes I_{\mathcal{G}}\|_1 = \|T\|_\diamond$, where the quantity $\|T\|_\diamond$ does not depend on $\mathcal{G}$. Before proving this assertion, let us examine its consequences.

First, it is clear that the quantity $\|T\|_\diamond$ defined in this manner is a norm.

Second, let us notice that $\|T \otimes R\|_1 \geq \|T\|_1\|R\|_1$, since the trace norm is multiplicative with respect to the tensor product. Substituting the identity operator for R, we obtain $\|T\|_\diamond \geq \|T\|_1$. Similarly, if we replace T by $T \otimes I_{\mathbf{L}(\mathcal{G})}$, and R by $R \otimes I_{\mathbf{L}(\mathcal{G})}$ (where the dimension of $\mathcal{G}$ is large enough), we get $\|T \otimes R\|_\diamond \geq \|T\|_\diamond\|R\|_\diamond$.

Third, it follows from the definition that $\|TR\|_1 \leq \|T\|_1\|R\|_1$; therefore

$$\|T \otimes R\|_1 = \|(T \otimes I)(I \otimes R)\|_1 \leq \|T \otimes I\|_1\|I \otimes R\|_1.$$

Replacing T and R by $T \otimes I_{\mathbf{L}(\mathcal{G})}$ and $R \otimes I_{\mathbf{L}(\mathcal{G})}$ (resp.), we get $\|T \otimes R\|_\diamond \leq \|T\|_\diamond\|R\|_\diamond$, which is the opposite to the previous inequality. Hence the norm $\|\cdot\|_\diamond$ is multiplicative with respect to the tensor product,

$$\|T \otimes R\|_\diamond = \|T\|_\diamond\|R\|_\diamond. \tag{11.9}$$

In order to prove that $\big\|T \otimes I_{\mathbf{L}(\mathcal{G})}\big\|_1$ stabilize when $\dim \mathcal{G} \geq \dim \mathcal{N}$, we give another definition of the *stable superoperator norm* $\|T\|_\diamond$.

First, we note that an arbitrary superoperator $T : \mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{M})$ can be represented in the form $T = \mathrm{Tr}_{\mathcal{F}}(A \cdot B^\dagger)$, where $A, B \in \mathbf{L}(\mathcal{N}, \mathcal{M} \otimes \mathcal{F})$ (recall that $A \cdot B^\dagger$ denotes the superoperator $X \mapsto AXB^\dagger$; cf. Problem 11.1). Without loss of generality we may assume that $\dim \mathcal{F} = (\dim \mathcal{N})(\dim \mathcal{M})$. In fact, the dimension of $\mathcal{F}$ can be made as low as the rank of the matrix $(T_{(j'j)(k'k)})$ defined in Problem 11.4.

Definition 11.2. Consider all representations of $T : \mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{M})$ in the form $T = \mathrm{Tr}_{\mathcal{F}}(A \cdot B^\dagger)$. Then

$$\|T\|_\diamond = \inf\Big\{ \|A\| \, \|B\| \ : \ \mathrm{Tr}_{\mathcal{F}}(A \cdot B^\dagger) = T \Big\}.$$

It follows from Theorem 11.1 below that this quantity does not depend on the choice of the auxiliary space $\mathcal{F}$, provided at least one representation $T = \mathrm{Tr}_{\mathcal{F}}(A_0 \cdot B_0^\dagger)$ exists. For the minimization of $\|A\| \, \|B\|$, it suffices to consider operators with norms $\|A\| \leq \|A_0\|$ and $\|B\| \leq \|B_0\|$. The set of such pairs (A, B) is compact, hence the infimum is achieved.

Theorem 11.1. *If* $\dim \mathcal{G} \geq \dim \mathcal{N}$, *then* $\|T \otimes I_{\mathcal{G}}\|_1 = \|T\|_\diamond$.

Proof. Let $T = \mathrm{Tr}_{\mathcal{F}}(A \cdot B^\dagger)$. Using the properties of the trace norm from Problem 10.5, we obtain

$$\begin{aligned} \big\|(T \otimes I_{\mathbf{L}(\mathcal{G})})X\big\|_{\mathrm{tr}} &= \big\|\mathrm{Tr}_{\mathcal{F}}\big((A \otimes I_{\mathcal{G}})X(B^\dagger \otimes I_{\mathcal{G}})\big)\big\|_{\mathrm{tr}} \\ &\leq \big\|(A \otimes I_{\mathcal{G}})X(B^\dagger \otimes I_{\mathcal{G}})\big\|_{\mathrm{tr}} \leq \|A\| \, \|B\| \, \|X\|_{\mathrm{tr}}. \end{aligned}$$

Hence $\|T\|_\diamond \geq \big\|T \otimes I_{\mathbf{L}(\mathcal{G})}\big\|_1$.

Proving the opposite inequality is somewhat more complicated. Without loss of generality we may assume that $\|T\|_\diamond = 1$, and the inifimum in Definition 11.2 is achieved when $\|A\| = \|B\| = 1$.

We show at first that there exist three density matrices $\rho, \gamma \in \mathbf{L}(\mathcal{N})$ and $\tau \in \mathbf{L}(\mathcal{F})$ such that $\mathrm{Tr}_{\mathcal{M}}(A\rho A^\dagger) = \mathrm{Tr}_{\mathcal{M}}(B\gamma B^\dagger) = \tau$. Let

$$\begin{aligned} &\mathcal{K} = \mathrm{Ker}(A^\dagger A - I_{\mathcal{N}}), && \mathcal{L} = \mathrm{Ker}(B^\dagger B - I_{\mathcal{N}}), \\ &E = \Big\{ \mathrm{Tr}_{\mathcal{M}}(A\rho A^\dagger) : \rho \in \mathbf{D}(\mathcal{K}) \Big\}, && F = \Big\{ \mathrm{Tr}_{\mathcal{M}}(B\gamma B^\dagger) : \gamma \in \mathbf{D}(\mathcal{L}) \Big\}, \end{aligned}$$

where $\mathbf{D}(\mathcal{L})$ denotes the set of density matrices on the subspace $\mathcal{L}$. Then $E, F \subseteq \mathbf{D}(\mathcal{F})$, so that in place of τ we can put any element of $E \cap F$.

We prove that $E \cap F \neq \emptyset$. Since E and F are compact convex sets, it suffices to prove that there is no hyperplane that would separate E from F. In other words, there is no Hermitian operator $Z \in \mathbf{L}(\mathcal{F})$ such that $\mathrm{Tr}(XZ) > \mathrm{Tr}(YZ)$ for all pairs of $X \in E$ and $Y \in F$. This follows from

the condition that the value of $\|A\|\,\|B\|$ is minimal; in particular, it cannot decrease under the transformation

$$A \mapsto (I_{\mathcal{M}} \otimes e^{-tZ})\,A, \qquad B \mapsto (I_{\mathcal{M}} \otimes e^{tZ})\,B,$$

where t is a small positive number.

Thus, let $\operatorname{Tr}_{\mathcal{M}}(A\rho A^\dagger) = \operatorname{Tr}_{\mathcal{M}}(B\rho B^\dagger) = \tau \in \mathbf{D}(\mathcal{F})$, where $\rho, \gamma \in \mathbf{D}(\mathcal{N})$. We can use the additional space $\mathcal{G}$ to construct purifications of ρ and γ, i.e., to represent them in the form $\rho = \operatorname{Tr}_{\mathcal{G}}(|\xi\rangle\langle\xi|)$, $\gamma = \operatorname{Tr}_{\mathcal{G}}(|\eta\rangle\langle\eta|)$, where $|\xi\rangle, |\eta\rangle \in \mathcal{N} \otimes \mathcal{G}$ are unit vectors (see Proposition 10.1). This is possible due to the condition $\dim \mathcal{G} \geq \dim \mathcal{N}$. We set $X = |\xi\rangle\langle\eta|$. It is obvious that $\|X\|_{\mathrm{tr}} = 1$.

We prove that $\big\|(T \otimes I_{\mathbf{L}(\mathcal{G})})X\big\|_{\mathrm{tr}} \geq 1$. If we set

$$X' = (T \otimes I_{\mathbf{L}(\mathcal{G})})X, \qquad |\xi'\rangle = (A \otimes I_{\mathcal{G}})|\xi\rangle, \qquad |\eta'\rangle = (B \otimes I_{\mathcal{G}})|\eta\rangle,$$

then

$$X' = \operatorname{Tr}_{\mathcal{F}}(|\xi'\rangle\langle\eta'|), \qquad \operatorname{Tr}_{\mathcal{M}\otimes\mathcal{G}}(|\xi'\rangle\langle\xi'|) = \operatorname{Tr}_{\mathcal{M}\otimes\mathcal{G}}(|\eta'\rangle\langle\eta'|) = \tau.$$

From this it follows, first, that the vectors $|\xi'\rangle$ and $|\eta'\rangle$ have unit length. Second, there is a unitary operator U acting on the space $\mathcal{M} \otimes \mathcal{G}$ such that $(U \otimes I_{\mathcal{F}})\,|\xi'\rangle = |\eta'\rangle$ (see Problem 10.3). Consequently,

$$\|X'\|_{\mathrm{tr}} \geq |\operatorname{Tr} UX'| = \big|\operatorname{Tr}\big((U \otimes I_{\mathcal{F}})|\xi'\rangle\langle\eta'|\big)\big| = \big|\operatorname{Tr}(|\eta'\rangle\langle\eta'|)\big| = 1.$$

□

Surprisingly enough, the superoperator norm is connected not only to the trace norm, but also to the fidelity (see (10.8)).

[3] **Problem 11.10.** Let $T = \operatorname{Tr}_{\mathcal{F}}(A \cdot B^\dagger)$, where $A, B : \mathcal{N} \to \mathcal{F} \otimes \mathcal{M}$. Prove that

$$\|T\|_\diamondsuit^2 = \max\Big\{F\big(\operatorname{Tr}_{\mathcal{M}}(A\rho A^\dagger),\, \operatorname{Tr}_{\mathcal{M}}(B\gamma B^\dagger)\big) \;:\; \rho, \gamma \in \mathbf{D}(\mathcal{N})\Big\},$$

where $\mathbf{D}(\mathcal{N})$ denotes the set of density matrices on $\mathcal{N}$.

Note that the operators $\rho' = \operatorname{Tr}_{\mathcal{F}}(A \cdot B^\dagger)$ and $\gamma' = \operatorname{Tr}_{\mathcal{M}}(B\gamma B^\dagger)$ are not density matrices: the condition $\operatorname{Tr}\rho' = \operatorname{Tr}\gamma' = 1$ is not satisfied. However, the defininion of fidelity and the result of Problem 10.6b (but not 10.6a or 10.6c) is valid for arbitrary nonnegative Hermitian operators.

The result of Problem 11.10 has been used in the study of the complexity class QIP [**38**].

12. Measuring operators

A measuring operator is a generalization of an operator with quantum control. Such operators are very useful in constructing quantum algorithms. After mastering this tool, we will be ready to tackle difficult computational problems.

12.1. Definition and examples. Consider a state space $\mathcal{N} \otimes \mathcal{K}$ and fix a decomposition of the first factor into a direct sum of pairwise orthogonal subspaces: $\mathcal{N} = \bigoplus_{j\in\Omega} \mathcal{L}_j$ $(\Omega = \{1, \dots, r\})$. Each operator of the form

$$W = \sum_j \Pi_{\mathcal{L}_j} \otimes U_j$$

is called a *measuring operator*, where $\Pi_{\mathcal{L}_j}$ is the projection onto the subspace $\mathcal{L}_j$, and $U_j \in \mathbf{L}(\mathcal{K})$ is unitary.

In order to justify the name "measuring", we consider the following process. Let $\rho \in \mathbf{L}(\mathcal{N})$ be a quantum state we want to measure. We connect it to an *instrument* in the initial state $|0\rangle$ (we assume that in the state space of the instrument, $\mathcal{K}$, some fixed basis is chosen, e.g., $\mathcal{K} = \mathcal{B}^{\otimes n}$). Then the joint state of the system is described by the density matrix $\rho \otimes |0^m\rangle\langle 0^m|$.

Now we apply the measuring operator W. We obtain the state

$$W\Big(\rho \otimes |0^m\rangle\langle 0^m|\Big)W^\dagger = \sum_j \big(\Pi_{\mathcal{L}_j}\rho\Pi_{\mathcal{L}_j}\big) \otimes \big(U_j|0\rangle\langle 0|U_j^\dagger\big)$$

(we have used the defining properties of a projection: $\Pi^\dagger = \Pi,\ \Pi^2 = \Pi$).

Finally, we make the instrument classical by applying the decoherence transformation. This means that the matrix is diagonalized with respect to the second tensor factor. Let us see how the factor $U_j|0\rangle\langle 0|U_j^\dagger$ in the above sum changes:

$$U_j|0\rangle\langle 0|U_j^\dagger \mapsto \sum_k \big|\langle k|U_j|0\rangle\big|^2 |k\rangle\langle k|.$$

Thus we obtain a bipartite mixed state, which is classical in the second component,

$$\sum_j \sum_k \Big(\Pi_{\mathcal{L}_j}\rho\Pi_{\mathcal{L}_j}\big|\langle k|U_j|0\rangle\big|^2,\ k\Big) = \sum_j \sum_k \mathbf{P}(k|j)\cdot\big(\Pi_{\mathcal{L}_j}\rho\Pi_{\mathcal{L}_j}, k\big).$$

Here we have introduced the *conditional probabilities* $\mathbf{P}(k|j) = \big|\langle k|U_j|0\rangle\big|^2$. We will see that they obey the usual rules of probability theory, provided all measuring operators we use are defined with respect to *the same* orthogonal decomposition of the space $\mathcal{N}$.

Summing up, the whole procedure corresponds to the transformation of density matrices

$$T:\ \rho \mapsto \sum_{k,j} \mathbf{P}(k|j) \cdot \big(\Pi_{\mathcal{L}_j} \rho \Pi_{\mathcal{L}_j}, k\big).$$

We note that a projective measurement (as defined in the preceding section) is a special case of this, $\mathbf{P}(k|j) = \delta_{kj}$. The more general process just described can be called a "probabilistic projective measurement." It can also be viewed as a nondestructive version of the POVM measurement

$$\rho \mapsto \sum_k \operatorname{Tr}(\rho X_k) \cdot (k), \quad \text{where } X_k = \sum_j \mathbf{P}(k|j) \Pi_{\mathcal{L}_j}.$$

Let us give a few examples of measuring operators.

1. The operator $\Lambda(U) = \Pi_0 \otimes I + \Pi_1 \otimes U$, acting on the space $\mathcal{B} \otimes \mathcal{N}$, is measuring.

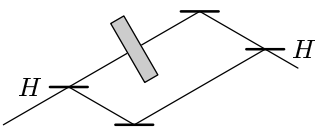
1′. It is more interesting that $\Lambda(U)$ is measuring also with respect to the second subsystem, $\mathcal{N}$. Since U is a unitary operator, it can be decomposed into the sum of projections onto the eigenspaces: $U = \sum_j \lambda_j \Pi_{\mathcal{L}_j}$, $|\lambda_j| = 1$. Then

$$\Lambda(U) = \sum_j (\Pi_0 + \lambda_j \Pi_1) \otimes \Pi_{\mathcal{L}_j} = \sum_j \begin{pmatrix} 1 & 0 \\ 0 & \lambda_j \end{pmatrix} \otimes \Pi_{\mathcal{L}_j}.$$

However, the conditional probabilities are trivial: $\mathbf{P}(0|j) = 1$, $\mathbf{P}(1|j) = 0$. Thus such an operator, even though measuring according to the definition, does not actually measure anything.

Now we will try to modify the operator $\Lambda(U)$ to make the conditional probabilities nontrivial.

Physicist's approach. Let U be the operator of phase shift for light as it passes through a glass plate. We can split the light beam into two parts by having it pass through a semitransparent mirror. Then one of the two beams passes through the glass plate, after which the beams merge at another semitransparent mirror (see the diagram). The resulting interference will allow us to determine the phase shift.



A mathematical variant of the preceding example. The operator

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

serves as an analog of the semitransparent mirror. As is evident from the diagram above, we need to apply it at the beginning and at the end. The

middle part is represented by the operator $\Lambda(U)$ (the controlling qubit corresponds to whether the photon passes through the plate or not). Thus we obtain the operator

$$\Xi(U) = (H \otimes I)\,\Lambda(U)\,(H \otimes I) : \quad \mathcal{B} \otimes \mathcal{N} \to \mathcal{B} \otimes \mathcal{N}. \tag{12.1}$$

If the initial vector has the form $|\psi\rangle = |\eta\rangle \otimes |\xi\rangle$ ($|\xi\rangle \in \mathcal{L}_j$), then $\Xi(U)|\psi\rangle = |\eta'\rangle \otimes |\xi\rangle$, where

$$|\eta'\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & \lambda_j\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}|\eta\rangle = \frac{1}{2}\begin{pmatrix}1+\lambda_j & 1-\lambda_j\\ 1-\lambda_j & 1+\lambda_j\end{pmatrix}|\eta\rangle.$$

Therefore

$$\Xi(U) = \sum_j \overbrace{\frac{1}{2}\begin{pmatrix}1+\lambda_j & 1-\lambda_j\\ 1-\lambda_j & 1+\lambda_j\end{pmatrix}}^{R_j} \otimes \Pi_{\mathcal{L}_j}.$$

Now we calculate the conditional probabilities. The eigenvalues have modulus 1, so that we can write $\lambda_j = \exp(2\pi i\varphi_j)$. As a consequence, we have

$$\mathbf{P}(0|j) = \bigl|\langle 0|R_j|0\rangle\bigr|^2 = \left|\frac{1+\lambda_j}{2}\right|^2 = \frac{1+\cos(2\pi\varphi)}{2}. \tag{12.2}$$

In the next section, the measuring operator (12.1) will be used to estimate the eigenvalues of unitary operators. For this, we will need to apply the operator $\Xi(U)$ several times to the same "object" (the space $\mathcal{N}$), but with different "instruments" (copies of the space $\mathcal{B}$). But first we must make sure that this is correct, in the sense that the probabilities multiply as they should.

12.2. General properties. We will consider measuring operators that correspond to a *fixed* orthogonal decomposition $\mathcal{N} = \bigoplus_j \mathcal{L}_j$.

1. *The product of measuring operators is a measuring operator.* Indeed, let two such operators be

$$W^{(1)} = \sum_j R_j^{(1)} \otimes \Pi_{\mathcal{L}_j} \quad \text{and} \quad W^{(2)} = \sum_j R_j^{(2)} \otimes \Pi_{\mathcal{L}_j}.$$

Inasmuch as $\Pi_{\mathcal{L}_j}\Pi_{\mathcal{L}_k} = \delta_{jk}\Pi_{\mathcal{L}_k}$, we have

$$W^{(2)}W^{(1)} = \sum_j R_j^{(2)}R_j^{(1)} \otimes \Pi_{\mathcal{L}_j}.$$

2. *The conditional probabilities for products of measuring operators with "different instruments" are multiplicative.* Specifically, if $R^{(1)} = \tilde{R}^1 \otimes I_{\mathcal{K}_2}$, $R^{(2)} = I_{\mathcal{K}_1} \otimes \tilde{R}^2$ (both operators act on $\mathcal{K}_1 \otimes \mathcal{K}_2$), then $\mathbf{P}(k_1, k_2|j) =$

$\mathbf{P}(k_1|j)\,\mathbf{P}(k_2|j)$. This equality follows immediately from the definition of conditional probabilities and the obvious identity

$$\big(\langle\xi_1|\otimes\langle\xi_2|\big)\,\big(U_1\otimes U_2\big)\,\big(|\eta_1\rangle\otimes|\eta_2\rangle\big)\;=\;\langle\xi_1|U_1|\eta_1\rangle\,\langle\xi_2|U_2|\eta_2\rangle.$$

3. *Formula of total probability.* Let $W=\sum R_j\otimes\Pi_{\mathcal{L}_j}$ be a measuring operator. If we apply it to the state $|0\rangle\langle 0|\otimes\rho$, where $\rho\in\mathbf{L}(\mathcal{N})$, then the resulting probability of the state k can be written in the form:

$$\mathbf{P}\Big(W\big(|0\rangle\langle 0|\otimes\rho\big)W^\dagger,\,\mathbb{C}\,(|k\rangle)\otimes\mathcal{N}\Big)=\sum_j\mathbf{P}(k|j)\,\mathbf{P}(\rho,\mathcal{L}_j).$$

Proof. We have

$$W\big(|0\rangle\langle 0|\otimes\rho\big)W^\dagger=\gamma=\sum_j\big(R_j|0\rangle\langle 0|R_j^\dagger\big)\otimes\Pi_{\mathcal{L}_j}\rho\Pi_{\mathcal{L}_j}.$$

It was proved earlier that $\mathbf{P}\big(\gamma,\,\mathbb{C}(|k\rangle)\otimes\mathcal{N}\big)=\mathbf{P}\big(\mathrm{Tr}_{\mathcal{N}}(\gamma),\,\mathbb{C}(|k\rangle)\big)$. Further,

$$\mathrm{Tr}_{\mathcal{N}}(\gamma)=\sum_j\Big(R_j|0\rangle\langle 0|R_j^\dagger\Big)\;\mathrm{Tr}\big(\Pi_{\mathcal{L}_j}\rho\Pi_{\mathcal{L}_j}\big).$$

Since

$$\mathrm{Tr}(\Pi_{\mathcal{L}_j}\rho\Pi_{\mathcal{L}_j})=\mathrm{Tr}(\Pi_{\mathcal{L}_j}^2\rho)=\mathrm{Tr}(\Pi_{\mathcal{L}_j}\rho)\stackrel{\text{def}}{=}\mathbf{P}(\rho,\mathcal{L}_j),$$

we obtain the desired expression $\mathbf{P}\big(\gamma,\,\mathbb{C}(|k\rangle)\otimes\mathcal{N}\big)=\sum\limits_j\mathbf{P}(k|j)\,\mathbf{P}(\rho,\mathcal{L}_j)$. □

[1!] **Problem 12.1.** Let $\mathcal{N}=\bigoplus_j\mathcal{L}_j$ be an orthogonal decomposition, $U_j\in\mathcal{K}$ and $\tilde U_j\in\mathcal{K}\otimes\mathcal{B}^{\otimes N}$ some unitary operators. Suppose that for each j the operator $\tilde U_j$ approximates U_j with precision δ using ancillas (namely, the space $\mathcal{B}^{\otimes N}$). Then the measuring operator $\tilde W=\sum_j\Pi_{\mathcal{L}_j}\otimes\tilde U_j$ approximates $W=\sum_j\Pi_{\mathcal{L}_j}\otimes U_j$ with the same precision δ.

12.3. Garbage removal and composition of measurements. Measurement operators are used to obtain some information about the value of the index j in the decomposition $\mathcal{N}=\bigoplus_{j\in\Omega}\mathcal{L}_j$. From a computation perspective, only a part of this information may be useful. In this situation, the measurement operator can be written in the form

$$W=\sum_{j\in\Omega}\Pi_{\mathcal{L}_j}\otimes R_j,\qquad R_j:\,\mathcal{B}^N\to\mathcal{B}^N,\qquad R_j|0\rangle=\sum_{y,z}c_{y,z}(j)|y,z\rangle,\tag{12.3}$$

where $y\in\mathbb{B}^m$ represents the "useful result" and $z\in\mathbb{B}^{N-m}$ is "garbage". Ignoring the garbage, we get the conditional probabilities

$$\mathbf{P}(y|j)\stackrel{\text{def}}{=}\big\langle 0\big|R_j^\dagger\,\Pi_{\mathcal{M}_y}R_j\big|0\big\rangle,\qquad\mathcal{M}_y=\mathbb{C}\big(|y\rangle\big)\otimes\mathcal{B}^{\otimes(N-m)}.\tag{12.4}$$

How can one construct another measuring operator U that would produce y with the same conditional probabilities, but without garbage? It

seems that there is no general solution to this problem, except for the case where the result y is deterministic, namely, $\mathbf{P}(y|j) = \delta_{y,f(j)}$ for some function $f : \Omega \to \mathbb{B}^m$ (so that we can say that W actually *measures the value of* $f(j)$). Then we can use the same trick as in the proof of Lemma 7.2: we measure $f(j)$, copy the result, and "un-measure".

We are going to extend this simple result in three ways. First, we will assume that W measures f with some error probability ε; we will find out with what precision the above procedure corresponds to a garbage-free measurement (the answer is $\sqrt{2\varepsilon}$). Second, the formula for the conditional probabilities (12.4) makes sense for an arbitrary orthogonal decomposition $\mathcal{B}^{\otimes N} = \bigoplus_{y\in\Delta} \mathcal{M}_y$. Third, instead of copying the result, we can apply any operator V that is measuring with respect to the indicated decomposition. The copying corresponds to $V : |y, z, v\rangle \mapsto |y, z, y \oplus v\rangle$.

[2!] **Problem 12.2.** Let $\mathcal{N} = \bigoplus_{j\in\Omega} \mathcal{L}_j$ and $\mathcal{B}^{\otimes N} = \bigoplus_{y\in\Delta} \mathcal{M}_y$ be orthogonal decompositions. Consider two measuring operators on $\mathcal{N} \otimes \mathcal{K} \otimes \mathcal{B}^{\otimes N}$, such that $\mathcal{B}^{\otimes N}$ serves as the "instrument" for one and the "object" for the other:

$$W = \sum_{j\in\Omega} \Pi_{\mathcal{L}_j} \otimes I_{\mathcal{K}} \otimes R_j, \qquad V = \sum_{y\in\Delta} I_{\mathcal{N}} \otimes Q_y \otimes \Pi_{\mathcal{M}_y}.$$

Suppose that W *measures a function* $f : \Omega \to \Delta$ *with error probability* $\leq \varepsilon$, i.e., the conditional probabilities $\mathbf{P}(y|j) = \langle 0^N | R_j^\dagger \Pi_{\mathcal{M}_y} R_j | 0^N \rangle$ satisfy $\mathbf{P}(f(j)|j) \geq 1 - \varepsilon$. Then the operator $\tilde{U} = W^{-1}VW$ approximates the operator

$$U = \sum_{j\in\Omega} \Pi_{\mathcal{L}_j} \otimes Q_{f(j)} \ : \ \mathcal{N} \otimes \mathcal{K} \to \mathcal{N} \otimes \mathcal{K}$$

with precision $2\sqrt{\varepsilon}$, using $\mathcal{B}^{\otimes N}$ as the ancillary space. If V is the copy operator, then the precision is $\sqrt{2\varepsilon}$.

13. Quantum algorithms for Abelian groups

The only nontrivial quantum algorithm we have considered so far is Grover's algorithm for the solution of the universal search problem (see Section 9.2). Unfortunately, there we achieved only a polynomial increase in speed. For this reason Grover's algorithm does not yield any serious consequences (of type BQP $\supset$ BPP) for complexity theory. At present time, there is no proof that quantum computation is super-polynomially faster than classical probabilistic computation. But there are several pieces of indirect evidence in favor of such an assertion. The first of these is an example of a *problem with oracle* (cf. Definition 2.2 on page 26 and the beginning of Section 9.2),

for which there exists a polynomial quantum algorithm, while any classical probabilistic algorithm is exponential.[10] This example, constructed by D. Simon [**66**], is called the hidden subgroup problem for $(\mathbb{Z}_2)^k$. The famous factoring algorithm by P. Shor [**62**] is based on a similar idea. After discussing these examples, we will solve the hidden subgroup problem for the group $\mathbb{Z}^k$, which generalizes both results.

THE HIDDEN SUBGROUP PROBLEM. Let G be a group with a specified representation of its elements by binary words. There is a device (an oracle) that computes some function $f : G \to \mathcal{B}^n$ with the following property:

$$f(x) = f(y) \iff x - y \in D, \tag{13.1}$$

where $D \subseteq G$ is an initially unknown subgroup. It is required to find that subgroup. (The result should be presented in a specified way.)

13.1. The problem of hidden subgroup in $(\mathbb{Z}_2)^k$; Simon's algorithm. We consider the problem formulated above for the group $G = (\mathbb{Z}_2)^k$. The elements of this group can be represented by length k words of zeros and ones; the group operation is bitwise addition modulo 2. We may regard G as the k-dimensional linear space over the field $\mathbb{F}_2$. Any subgroup of G is a linear subspace, so it can be represented by a basis.

It is easy to show that a "hidden subgroup" cannot be found quickly using a classical probabilistic machine. (A classical machine sends words $x_1, \dots, x_l$ to the input of the "black box" and receives answers $y_1, \dots, y_l$. Each subsequent query x_j depends on the previous answers $y_1, \dots, y_{j-1}$ and some random number r that is generated in advance.)

Proposition 13.1. *Let $n \geq k$. For any classical probabilistic algorithm making no more than $2^{k/2}$ queries to the oracle, there exist a subgroup $D \subseteq (\mathbb{Z}_2)^k$ and a corresponding function $f : (\mathbb{Z}_2)^k \to \mathcal{B}^n$ for which the algorithm is wrong with probability $> 1/3$.*

Proof. For the same subgroup D there exist several different oracles f. We assume that one of them is chosen randomly and uniformly. (If the algorithm is wrong with probability $> 1/3$ for the randomized oracle, then it will be wrong with probability $> 1/3$ for some particular oracle.) The randomized oracle works as follows. If the present query is x_j, and $x_j - x_s \in D$ for some $s < j$, the answer y_j coincides with the answer y_s that was given before. Otherwise, y_j is uniformly distributed over the set $\mathcal{B}^n \setminus \{y_1, \dots, y_{j-1}\}$. The randomized oracle is not an oracle in the proper sense, meaning that its answer may depend on the previous queries rather than only on the current

[10] One should keep in mind that the complexity of problems with oracle frequently differs from the complexity of ordinary computational problems. A classical example is the theorem stating that IP = PSPACE [**58, 59**]. The oracle analogue of this assertion is not true [**25**]!

one. In this manner, the randomized oracle is equivalent to a device with memory, which, being asked the question x_j, responds with the smallest number $s_j \leq j$ such that $x_j - x_{s_j} \in D$. Indeed, if we have a classical probabilistic machine making queries to the randomized oracle, we can adapt it for the use with the device just described. For this, the machine should be altered in such a way that it will transform each answer s_j to y_j and proceed as before. (That is, $y_j = y_s$ if $s_j < j$, or y_j is uniformly distributed over $\mathcal{B}^n \setminus \{y_1, \ldots, y_{j-1}\}$ if $s_j = j$.)

Let the total number of queries be $l \leq 2^{k/2}$. Without loss of generality all queries can be assumed different. In the case $D = \{0\}$, all answers are also different, i.e., $s_j = j$ for all j. Now we consider the case $D = \{0, z\}$, where z is chosen randomly, with equal probabilities, from the set of all nonzero elements of the group $(\mathbb{Z}_2)^k$. Then, regardless of the algorithm that is used, $z \notin \{x_j - x_1, \ldots, x_j - x_{j-1}\}$ with probability $\geq 1 - (j-1)/(2^k - 1)$. The condition for z implies that $s_j = j$. This is true for all $j = 1, \ldots, l$ with probability $\geq 1 - l(l-1)/(2(2^{k-1} - 1)) > 1/2$. Recall that we have two random parameters: z and r. We can fix z in such a way that the probability of obtaining the answers $s_j = j$ (for all j) will still be greater than $1/2$. Let us see what a classical machine would do in such a circumstance. If it gives the answer "$D = \{0\}$" with probability $\geq 2/3$, we set $D = \{0, z\}$, and then the resulting answer will be wrong with probability $> (2/3) \cdot (1/2) = 1/3$. If, however, the probability of the answer "$D = \{0\}$" is smaller than $2/3$, we set $D = \{0\}$. □

Let us define a quantum analog of the oracle f. The corresponding quantum oracle is a unitary operator

$$U : |x, y\rangle \to |x, y \oplus f(x)\rangle \tag{13.2}$$

($\oplus$ denotes bitwise addition). We note that the quantum oracle allows linear combinations of the various queries; therefore it is possible to use it more efficiently than the classical oracle.

Now we describe Simon's algorithm for finding the hidden subgroup in $\mathbb{Z}_2^k$. Let $E = G/D$ and let E^* be the group of characters on E. (By definition, a *character* is a homomorphism $E \to \mathbf{U}(1)$.) In the case $G = (\mathbb{Z}_2)^k$ we can characterize the group E^* in the following manner:

$$E^* = \{h \in (\mathbb{Z}_2)^k \text{ such that } h \cdot z = 0 \text{ for all } z \in D\},$$

where $h \cdot z$ denotes the inner product modulo 2. (The character corresponding to h has the form $z \mapsto (-1)^{h \cdot z}$.) Let us show how one can generate a random element $h \in E^*$ using the operator U. After generating sufficiently many random elements, we will find the group E^*, hence the desired subgroup D.

We begin by preparing the state

$$|\xi\rangle = 2^{-k/2} \sum_{x \in G} |x\rangle = H^{\otimes k} |0^k\rangle$$

in the first quantum register. In the second register we place the state $|0^n\rangle$ and apply the operator U. Then we discard the second register, i.e., we will no longer make use of its contents. Thus we obtain the mixed state

$$\rho = \mathrm{Tr}_2\Big(U\big(|\xi\rangle\langle\xi| \otimes |0\rangle\langle 0|\big)U^\dagger\Big) = 2^{-k} \sum_{x,y:\, x-y \in D} |x\rangle\langle y|.$$

Now we apply the operator $H^{\otimes k}$ to the remaining first register. This yields a new mixed state

$$\gamma = H^{\otimes k} \rho H^{\otimes k} = 2^{-2k} \sum_{a,b} \sum_{x,y:\, x-y \in D} (-1)^{a \cdot x - b \cdot y} |a\rangle\langle b|.$$

It is easy to see that $\sum_{x,y:\, x-y \in D} (-1)^{a \cdot x - b \cdot y}$ is different from zero only in the case where $a = b \in E^*$. Hence

$$\gamma = \frac{1}{|E^*|} \sum_{a \in E^*} |a\rangle\langle a|.$$

This is precisely the density matrix corresponding to the uniform probability distribution on the group E^*. It remains to apply the following lemma, which we formulate as a problem.

[2!] **Problem 13.1.** Let $h_1, \ldots, h_l$ be independent uniformly distributed random elements of an Abelian group X. Prove that they generate the entire group X with probability $\geq 1 - |X|/2^l$.

Therefore, $2k$ random elements suffice to generate the entire group E^* with probability of error $\leq 2^{-k}$, where "error" refers to the case where $h_1, \ldots, h_{2k}$ actually generate a proper subgroup of E^*. (Note that such a small — compared to $1/3$ — probability of error is obtained without much additional expense. To make it still smaller, it is most efficient to use the standard procedure: repeat all calculations several times and choose the most frequent outcome.)

Summing up, to find the "hidden subgroup" D, we need $O(k)$ queries to the quantum oracle. The overall complexity of the algorithm is $O(k^3)$.

13.2. Factoring and finding the period for raising to a power. A second piece of evidence in favor of the hypothesis BQP $\supset$ BPP is the fast quantum algorithm for factoring integers into primes and for another number-theoretic problem — finding the discrete logarithm. They were found by P. Shor [**62**]. Let us discuss the first of these two problems.

FACTORING (an integer into primes). Suppose we are given a positive integer y. It is required to find its decomposition into prime factors

$$y = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}.$$

This problem is thought to be so complex that practical cryptographic algorithms are based on the hypothetical difficulty of its solution. From the theoretical viewpoint, the situation is somewhat worse: there is neither a reduction of problems of class NP to the factoring problem, nor any other "direct" evidence in favor of its complexity. (The word "direct" is put in quotation marks because at present the answer to the question $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ is unknown.) Therefore, the conjecture about the complexity of the factoring problem complements the abundant collection of unproved conjectures in the computational complexity theory. It is desirable to decrease the number of such problems. Shor's result is a significant step in this direction: if we commit an "act of faith" and believe in the complexity of the factoring problem, then the need for yet another act of faith (regarding the greater computational power of the quantum computer) disappears.

We will construct a fast quantum algorithm for solving not the factoring problem, but another problem, called PERIOD FINDING, to which the factoring problem is reduced with the aid of a classical probabilistic algorithm.

PERIOD FINDING. Suppose we are given an integer $q > 1$ that can be written using at most n binary digits (i.e., $q < 2^n$) and another integer a such that $1 \le a < q$ and $\gcd(a, q) = 1$ (where $\gcd(a, q)$ denotes the greatest common divisor). It is required to find the period of a with respect to q, i.e., the smallest nonnegative number t such that $a^t \equiv 1 \pmod{q}$.

In other words, the period is the order of the number a in the multiplicative group of residues $(\mathbb{Z}/q\mathbb{Z})^*$. We will denote the period of a with respect to q by $\mathrm{per}_q(a)$.

Below we will examine a quantum algorithm for the solution of the period finding problem. But we will begin by describing the classical probabilistic reduction of the factoring problem to the problem of finding the period. We suggest that the reader reviews the probabilistic test for primality presented in Part 1 (see Section 4.2).

13.3. Reduction of factoring to period finding. Thus, let us assume that we know how to find the period. It is clear that we can factor the number y by running $O(\log y)$ times a subprogram which, for any composite number, finds a nontrivial divisor with probability at least $1/2$. (Of course, it is also necessary to use the standard procedure for amplification of success probability; see formula (4.1) on p. 37 and the paragraph preceding it.)

Procedure for finding a nontrivial divisor.

Input. An integer y $(y > 1)$.

Step 1. Check y for parity. If y is even, then give the answer "2"; otherwise proceed to Step 2.

Step 2. Check whether y is the k-th power of an integer for $k = 2, \dots, \log_2 y$. If $y = m^k$, then give the answer "m"; otherwise proceed to Step 3.

Step 3. Choose an integer a randomly and uniformly between 1 and $y - 1$. Compute $b = \gcd(a, q)$ (say, by Euclid's algorithm). If $b > 1$, then give the answer "b"; otherwise proceed to Step 4.

Step 4. Compute $r = \mathrm{per}_y(a)$ (using the period finding algorithm that we assume we have). If r is odd, then the answer is "y is prime" (which means that we give up finding a nontrivial divisor). Otherwise proceed to Step 5.

Step 5. Compute $d = \gcd(a^{r/2} - 1, y)$. If $d > 1$, then the answer is "d"; otherwise the answer is "y is prime".

Analysis of the divisor finding procedure. If the above procedure yields a number, it is a nontrivial divisor of y. The procedure fails and gives the answer "y is prime" in two cases: 1) when $r = \mathrm{per}_y(a)$ is odd, or 2) when r is even but $\gcd(a^{r/2} - 1, y) = 1$, i.e., $a^{r/2} - 1$ is invertible modulo y. However, $(a^{r/2} + 1)(a^{r/2} - 1) \equiv a^r - 1 \equiv 0 \pmod{y}$, hence $a^{r/2} + 1 \equiv 0 \pmod{y}$ in this case. The converse is also true: if r is even and $a^{r/2} + 1 \equiv 0 \pmod{y}$, then the answer is "$y$ is prime".

Let us prove that our procedure succeeds with probability at least $1 - 1/2^{k-1}$, where k is the number of distinct prime divisors of y. (Note that this probability vanishes for prime y, so that the the procedure also works as a primality test.) In the proof we will need the Chinese Remainder Theorem (Theorem A.5 on page 241) and the fact that the multiplicative group of residues modulo p^α, p prime, is cyclic (see Theorem A.11).

Let $y = \prod_{j=1}^{k} p_j^{\alpha_j}$ be the decomposition of y into prime factors. We introduce the notation

$$a_j \equiv a \pmod{p_j^{\alpha_j}}, \qquad r_j = \mathrm{per}_{(p_j^{\alpha_j})}\, a_j = 2^{s_j} r_j', \quad \text{where } r_j' \text{ is odd.}$$

By the Chinese Remainder Theorem, r is the least common multiple of all the r_j. Hence $r = 2^s r'$, where $s = \max\{s_1, \dots, s_k\}$ and r' is odd.

We now prove that the procedure yields the answer "y is prime" if and only if $s_1 = s_2 = \cdots = s_k$. Indeed, if $s_1 = \cdots = s_k = 0$, then r is odd. If $s_1 = \cdots = s_k \geq 1$, then r is even, but $a_j^{r_j/2} \equiv -1 \pmod{p_j^{\alpha_j}}$ (using the cyclicity of the group $(\mathbb{Z}/p_j^{\alpha_j}\mathbb{Z})^*$), hence $a^{r/2} \equiv -1 \pmod{y}$ (using the

Chinese Remainder Theorem). Thus the procedure yields the answer "y is prime" in both cases. Conversely, if not all the s_j are equal, then r is even and $s_m < s$ for some m, so that $a_m^{r/2} \equiv 1 \pmod{p_m^{\alpha_m}}$. Hence $a^{r/2} \not\equiv -1 \pmod{y}$, i.e., the procedure yields a nontrivial divisor.

To give a lower bound of the success probability, we may assume that the procedure has reached Step 4. Thus a is chosen according to the uniform distribution over the group $(\mathbb{Z}/q\mathbb{Z})^*$. By the Chinese Remainder Theorem, the uniform random choice of a is the same as the independent uniform random choice of $a_j \in (\mathbb{Z}/p_j^{\alpha_j}\mathbb{Z})^*$ for each j. Let us fix j, choose some $s \geq 0$ and estimate the probability of the event $s_j = s$ for the uniform distribution of a_j. Let g_j be a generator of the cyclic group $(\mathbb{Z}/p_j^{\alpha_j}\mathbb{Z})^*$. The order of this group may be represented as $p_j^{\alpha_j} - p_j^{\alpha_j - 1} = 2^{t_j} q_j$, where q_j is odd. Then

$$\begin{aligned} \bigl|\{a_j : s_j = s\}\bigr| &= \Bigl|\bigl\{g_j^l : l = 2^{t_j - s} m, \text{ where } m \text{ is odd}\bigr\}\Bigr| \\ &= \begin{cases} q_j & \text{if } s = 0, \\ (2^s - 2^{s-1}) q_j & \text{if } s = 1, \ldots, t_j. \end{cases} \end{aligned}$$

For any given s, the probability of the event $s_j = s$ does not exceed $1/2$. Now let $s = s_1$ be a random number (depending on a_1); then $\mathbf{Pr}[s_j = s] \leq 1/2$ for $j = 2, \ldots, k$. It follows that

$$\mathbf{Pr}[s_1 = s_2 = \cdots = s_k] \leq (1/2)^{k-1}.$$

This yields the desired estimate of the success probability for the entire procedure: with probability at least $1 - 1/2^{k-1}$ the procedure finds a nontrivial divisor of y.

13.4. Quantum algorithm for finding the period: the basic idea. Thus, the problem is this: given the numbers q and a, construct a polynomial size quantum circuit that computes $\mathrm{per}_q(a)$ with error probability $\epsilon \leq 1/3$. The circuit will operate on a single n-qubit register, as well as on many other qubits, some of which may be considered classical. The n-qubit register is meant to represent residues modulo q (recall that $q < 2^n$).

Let us examine the operator that multiplies the residues by a, acting by the rule

$$U_a : |x\rangle \mapsto |ax \bmod q\rangle.$$

(A more accurate notation would be $U_{q,a}$, indicating the dependence on q. However, q is fixed throughout the computation, so we suppress it from the subscript. We keep a because we will also use the operators U_b for arbitrary b.) This operator permutes the basis vectors for $0 \leq x < q$ (recall that $(a, q) = 1$). However, we represent $|x\rangle$ by n qubits, so x may take any value between 0 and $2^n - 1$. We will assume that the operator U_a acts trivially on such basis vectors, i.e., $U_a : |x\rangle = |x\rangle$ for $q \leq x < 2^n$.

Since for the multiplication of the residues there is a Boolean circuit of polynomial — $O(n^2)$ — size, there is a quantum circuit (with ancillas) of about the same size.

The permutation given by the operator U_a can be decomposed into cycles. The cycle containing 1 is $(1, a, a^2, \ldots, a^{\mathrm{per}_q(a)-1})$; it has length $\mathrm{per}_q(a)$. The algorithm we are discussing begins at the state $|1\rangle$, to which the operator U_a gets applied many times. But such transformations do not take us beyond the orbit of 1 (the set of elements which constitute the cycle described above). Therefore we consider the restriction of the operator U_a to the subspace generated by the orbit of 1.

$$\text{Eigenvalues of } U_a: \quad \lambda_k = e^{2\pi i \cdot k/t}, \text{ where } t \text{ is the period;}$$

$$\text{Eigenvectors of } U_a: \quad |\xi_k\rangle = \frac{1}{\sqrt{t}} \sum_{m=0}^{t-1} e^{-2\pi i \cdot km/t} |a^m\rangle.$$

It is easy to verify that the vectors $|\xi_k\rangle$ are indeed eigenvectors. It suffices to note that the multiplication by a leads to a shift of the indices in the sum. If we change the variable of summation in order to remove this shift, we get the factor $e^{2\pi i \cdot k/t}$.

If we are able to measure the eigenvalues of the operator U_a, then we can obtain the numbers k/t. First let us analyze how this will help us in determining the period.

Suppose we have a machine which in each run gives us the number k/t, where t is the sought-for period and k is a random number uniformly distributed over the set $\{0, \ldots, t-1\}$. We suppose that k/t is represented as an irreducible fraction k'/t' (if the machine were able to give the number in the form k/t, there would be no problem at all).

Having obtained several fractions of the form k'_1/t'_1, $k'_2/t'_2, \ldots, k'_l/t'_l$, we can, with high probability, find the number t by reducing these fractions to a common denominator.

Lemma 13.2. *If $l \geq 2$ fractions are obtained, then the probability that their least common denominator is different from t is less than $3 \cdot 2^{-l}$.*

Proof. The fractions k'_1/t'_1, $k'_2/t'_2, \ldots, k'_l/t'_l$ can be obtained as reductions of fractions $k_1/t, \ldots, k_l/t$ (i.e., $k'_j/t'_j = k_j/t$), where $k_1, \ldots, k_l$ are independently distributed random numbers. The least common multiple of $t'_1, \ldots, t'_l$ equals t if and only if the greatest common divisor of $k_1, \ldots, k_l$ and t is equal to 1.

The probability that $k_1, \ldots, k_l$ have a common prime divisor p does not exceed $1/p^l$. Therefore the probability of not getting t after reducing to a common denominator does not exceed $\sum_{k=2}^{\infty} \frac{1}{k^l} < 3 \cdot 2^{-l}$ (the range of the index k in this sum obviously includes all prime divisors of t). □

Now we construct the machine M that generates the number k/t (in the form of an irreducible fraction) for random uniformly distributed k. This will be a quantum circuit which realizes the measuring operator $W = \sum_{k=0}^{t-1} V_k \otimes \Pi_{\mathcal{L}_k}$, where $\mathcal{L}_k = \mathbb{C}(|\xi_k\rangle)$, the subspace generated by $|\xi_k\rangle$. The operators V_k are the form $|0\rangle \mapsto \sum_{y,z} |y, z\rangle$, where y is an irreducible fraction and z is garbage. In this, the conditional probabilities should satisfy the inequality

$$\mathbf{P}\Big(\boxed{\tfrac{k}{t}}\Big|k\Big) \stackrel{\text{def}}{=} \sum_z \Big|\Big\langle \boxed{\tfrac{k}{t}}, z\Big|V_k\Big|0\Big\rangle\Big|^2 \geq 1-\varepsilon, \tag{13.3}$$

where $\boxed{\frac{k}{t}}$ denotes the irreducible fraction equal to the rational number k/t.

The construction of such a measuring circuit is rather complex, so we first explain how it is used to generate the outcome y with the desired probability $w_y = \sum_{k\in M_y} \frac{1}{t}$, where $M_y = \Big\{k : \boxed{\frac{k}{t}} = y\Big\}$. Let us take the state $|1\rangle$ as the initial state. A direct computation (the reader is advised to carry it through) shows that

$$|1\rangle = \frac{1}{\sqrt{t}} \sum_{k=0}^{t-1} |\xi_k\rangle.$$

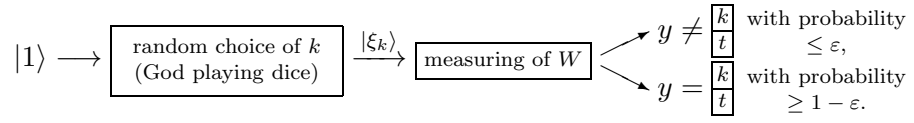
If we perform the measurement on this state, then by the formula for total probability we obtain

$$\mathbf{Pr}[\text{outcome} = y] = \mathbf{P}\big(W(|0\rangle \otimes |1\rangle), y\big) = \sum_k \mathbf{P}(y|k)\, \mathbf{P}(|1\rangle, \mathcal{L}_k).$$

The probabilities of all $|\xi_k\rangle$ are equal: $\mathbf{P}(|1\rangle, \mathcal{L}_k) = |\langle \xi_k|1\rangle|^2 = 1/t$, which corresponds to the uniform distribution of k. The property (13.3) guarantees that we obtain the outcome $\boxed{\frac{k}{t}}$ with probability $\geq 1-\varepsilon$. Well, the reader may find this statement not rigorous because k does not have a certain value. To be completely pedantic, we need to derive an inequality similar to (10.4), namely,

$$\sum_y \Big|\mathbf{Pr}[\text{outcome} = y] - w_y\Big| \leq 2\epsilon.$$

Schematically, the machine M functions as follows:



The random choice of k happens automatically, without applying any operator whatsoever. Indeed, the formula of total probability is arranged in such a way as if: before the measurement begins, a random k was generated,

which then remains constant. (Of course, the formula is only true when the operator W is measuring with respect to the given subspaces $\mathcal{L}_k$.)

13.5. The phase estimation procedure. Now we will construct the operator that measures the eigenvalues of U_a. The eigenvalues have the form

$$\lambda_k = e^{2\pi i \varphi_k}, \quad \text{where } \varphi_k = \frac{k}{t} \bmod 1.$$

The phase φ_k is a real number modulo 1, i.e., $\varphi_k \in \mathbb{R}/\mathbb{Z}$. (The set $\mathbb{R}/\mathbb{Z}$ can be conveniently represented as a circle of unit length.) The procedure for determining φ_k is called *phase estimation*.

As we already mentioned, we can limit ourselves to the study of the action of the operator U_a on the input vector $|\xi_k\rangle$. The construction is divided into four stages.

1. We construct a measuring operator such that the conditional probabilities depend on the value of $\varphi = \varphi_k$. Thus a single use of this operator will give us *some information* about φ (like flipping a biased coin tells something about the bias, though inconclusively) (see 13.5.1).
2. We localize the value of φ *with modest precision*. It is the moment to emphasize that, in all the arguments, there are two parameters: the *probability of error* ε and the *precision* δ. As the result of a measurement, we obtain some number y, for which the condition $|y-\varphi|_{\mathrm{mod}1} < \delta$ must hold with probability at least $1-\varepsilon$. (Here $|\cdot|_{\mathrm{mod}1}$ denotes the distance on the unit length circle, e.g., $|0.1-0.9|_{\mathrm{mod}1} = 0.2$.) For the time being, a modest precision will do, say $\delta = 1/16$ (see 13.5.2).
3. Now we must *increase the precision*. Specifically, we determine φ with precision $1/2^{2n+2}$ (see 13.5.3).
4. We need to pass from the approximate value of φ to the exact one, represented *in the form of an irreducible fraction*. It is essential to be able to distinguish between numbers of the form $\varphi = k/t$, where $0 \le k < t < 2^n$. Notice that if $k_1/t_1 \neq k_2/t_2$, then $|k_1/t_1 - k_2/t_2|_{\mathrm{mod}1} \ge 1/(t_1 t_2) > 1/2^{2n}$. Therefore, knowing $\varphi = k/t$ with precision $1/2^{2n+1}$, one can, in principle, determine its exact value. Moreover, this can be done efficiently by the use of continued fractions (see 13.5.4).

At stage 3, we will use the operator U_b for *arbitrary* b (not just for $b = a$, the number for which we seek the period). To this end, we introduce an operator U that sends $|b, x\rangle$ to $|b,\, bx \bmod q\rangle$ whenever $\gcd(b, q) = 1$. How the operator U acts in the remaining cases is not important; this action can be defined in an arbitrary computationally trivial way, so that U be represented by a quantum circuit of size $O(n^2)$. In fact, all the earlier arguments about the simulation of Boolean circuits by quantum circuits hold true for the simulation of circuits that compute partially defined functions.

[1] **Problem 13.2.** Using the operator U, realize the operator $\Lambda(U_b)$ for arbitrary b relatively prime to q.

13.5.1. ***How to get some information about the phase.*** In Section 12 we introduced the operator $\Xi(U_a) = (H \otimes I)\Lambda(U_a)(H \otimes I)$, which measures the eigenvalues. In our case $\lambda_k = e^{2\pi i\varphi_k}$ and we can write the operator in the form

$$\Xi(U_a) = \sum_k V_k \otimes \Pi_{\mathcal{L}_k}, \quad V_k = \frac{1}{2}\begin{pmatrix} 1+e^{2\pi i\varphi_k} & 1-e^{2\pi i\varphi_k} \\ 1-e^{2\pi i\varphi_k} & 1+e^{2\pi i\varphi_k} \end{pmatrix},$$

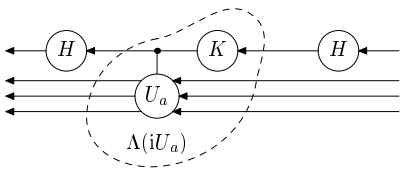
and its action in the form

$$|0\rangle \otimes |\xi_k\rangle \xrightarrow{\Xi(U_a)} \left(\frac{1+e^{2\pi i\varphi_k}}{2}|0\rangle + \frac{1-e^{2\pi i\varphi_k}}{2}|1\rangle\right) \otimes |\xi_k\rangle,$$

so that for the conditional probabilities we get the expressions

$$\mathbf{P}(0|k) = \left|\frac{1+e^{2\pi i\varphi_k}}{2}\right|^2 = \frac{1+\cos(2\pi\varphi_k)}{2}, \qquad \mathbf{P}(1|k) = \frac{1-\cos(2\pi\varphi_k)}{2}.$$

Although the conditional probabilities depend on φ_k, they do not allow one to distinguish between $\varphi_k = \varphi$ and $\varphi_k = -\varphi$. That is why another type of measurement is needed. We will use the operator $\Xi(iU_a)$. Its realization is shown in the diagram below. It uses the operator $K = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$ from



the standard basis. The encircled part of the diagram realizes the operator $\Lambda(iU_a)$. Indeed, K multiplies only $|1\rangle$ by i, but this is just the case where the operator U_a is applied (by the definition of $\Lambda(U_a)$). For the operator $\Xi(iU_a)$ the conditional probabilities are

$$\mathbf{P}(0|k) = \frac{1-\sin(2\pi\varphi_k)}{2}, \qquad \mathbf{P}(1|k) = \frac{1+\sin(2\pi\varphi_k)}{2}.$$

The complexity of the realization of the operators $\Xi(U_a)$ and $\Xi(iU_a)$ depends on the complexity of the operator $\Lambda(U_a)$, which is not much higher than the complexity of the operator U (cf. Problem 13.2). Thus, $\Xi(U_a)$ and $\Xi(iU_a)$ can be realized by quantum circuits of size $O(n^2)$ in the standard basis.

13.5.2. ***Determining the phase with constant precision.*** We want to localize the value of $\varphi = \varphi_k$, i.e., to infer the inequality $|\varphi - y|_{\mathrm{mod}1} < \delta$ for some (initially unknown) y and a given precision δ. To get such an estimate, we apply the operators $\Xi(U_a)$ and $\Xi(iU_a)$ to the same "object of measurement" but different "instruments" (auxiliary qubits). The reasoning is the same for both operators, so we limit ourselves to the case $\Xi(U_a)$.

We have the quantum register A that contains $|\xi_k\rangle$. Actually, this register initially contains $|1\rangle = \frac{1}{\sqrt{t}} \sum_{k=0}^{t-1} |\xi_k\rangle$, but we consider each $|\xi_k\rangle$ separately. (We can do this because we apply only operators that are measuring with respect to the orthogonal decomposition $\bigoplus_k \mathbb{C}(|\xi_k\rangle)$, so that different eigenvectors do not mix.) Let us introduce a large number s of auxiliary qubits. Each of them will be used in applying the operator $\Xi(U_a)$.

As was proved in Section 12 (see p. 114), the conditional probabilities in such a case multiply. For the operators $\prod_{r=1}^{s} \Xi(U_a)[r, A]$, the conditional probabilities are equal to $\mathbf{P}(v_1, \ldots, v_s|k) = \prod_{r=1}^{s} \mathbf{P}(v_r|k)$ (here v_r denotes the value of the r-th auxiliary qubit).

From this point on, the qubits containing the results of the "experiments" will only be operated upon classically. Since the conditional probabilities multiply, we can assume that we are estimating the probability $p_* = \mathbf{P}(1|k)$ of the outcome 1 ("head" in a coin flip) by performing a series of Bernoulli trials.

If the coin is tossed s times (where s is large), then the observed frequency $(\sum v_r)/s$ of the outcome 1 is close to its probability p_*. What is the accuracy of this estimate? The exact question: with what probability does the number $(\sum v_r)/s$ fail to approximate p_* with a given precision δ? The answer is given by *Chernoff's bound*:

$$\mathbf{Pr}\Big[\Big|s^{-1}\sum_{r=1}^{s} v_r - p_*\Big| \geq \delta\Big] \leq 2e^{-2\delta^2 s}. \tag{13.4}$$

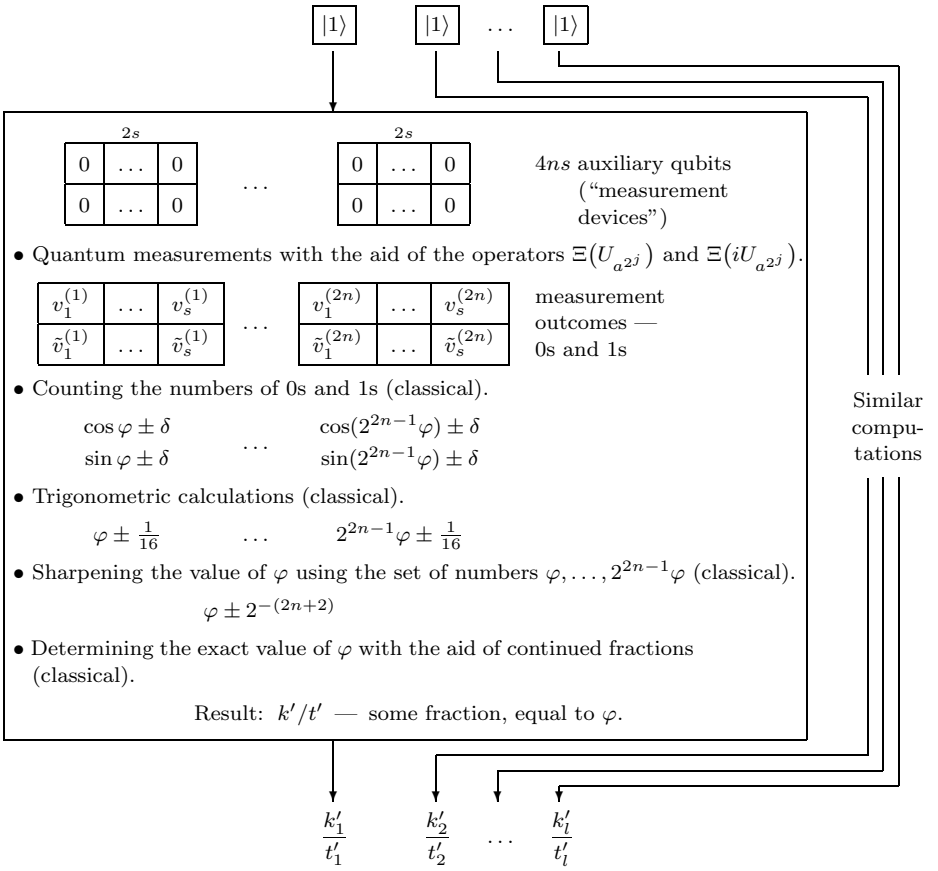
(This inequality is a generalization of the inequality (4.1) which was used to prove the amplification of success probability in the definitions of BPP and BQP.) Thus, for a fixed δ we can find a suitable constant $c = c(\delta)$ such that the error is smaller than ε when $s = \lceil c \log(1/\varepsilon) \rceil = \Theta(\log(1/\varepsilon))$ trials are made.

So, we have learned how to find $\cos(2\pi\varphi)$ and $\sin(2\pi\varphi)$ with any given precision δ. Now we choose δ so that the value φ can be determined from the values of the sine and the cosine with precision $1/16$. This still takes $\Theta(\log(1/\varepsilon))$ trials. The second stage is completed.

[3] **Problem 13.3.** Prove the inequality (13.4).

Input: a and q.

– Computation of the powers $a^{2^j} \bmod q$ for $j = 0, \ldots, 2n-1$ (classical).
– Setting up l quantum registers, containing the base state $|1\rangle$.



– Calculation of the least common denominator (classical).

Answer: t (with probability of error $< 3 \cdot 2^{-l} + nle^{-\Omega(s)}$).

Table 13.1. General scheme of the period finding algorithm. Shown in a box is the phase estimation part.

13.5.3. ***Determining the phase with exponential precision.*** To increase the precision, we will use, along with $\Lambda(U_a)$, the operators $\Lambda\big((U_a)^{2^j}\big)$ for all $j \leq 2n-1$. We can quickly raise numbers to a power, but, in general, computing a power of an operator is difficult. However, the operation U_a of (mod q)-multiplication by a possesses the following remarkable property:

$$(U_a)^p = U_{a^p} = U_{(a^p \bmod q)}.$$

Consequently, $\Lambda\big((U_a)^{2^j}\big) = \Lambda(U_b)$, where $b \equiv a^{2^j} \pmod q$. The required values for the parameter b can be calculated using a circuit of polynomial size; then we can apply the result of Problem 13.2.

Let us return to the circuit described in 13.5.1. We found the eigenvalue $\lambda_k = \lambda = e^{2\pi i\varphi}$ for some eigenvector $|\xi_k\rangle$. This same vector is an eigenvector for any power of the operator U_a, so that in the same quantum register we can look for an eigenvalue of $U_a^2 = U_{a^2}$ (it equals $\lambda^2 = e^{2\pi i\cdot 2\varphi}$), of $U_a^4 = U_{a^4}$ (it equals $\lambda^4 = e^{2\pi i\cdot 4\varphi}$), etc.

In other words, we can determine with precision $1/16$ the values of φ, 2φ, $\dots$, $2^{2n-1}\varphi$ modulo 1. But this allows us to *determine φ with precision $1/2^{2n+2}$ efficiently* (in linear time with constant memory). The idea is based on the following obvious fact: if $|y - 2\varphi|_{\mathrm{mod}1} < \delta < 1/2$, then

$$\text{either } |y_0' - \varphi|_{\mathrm{mod}1} < \delta/2 \quad \text{or } |y_1' - \varphi|_{\mathrm{mod}1} < \delta/2,$$

where y_0', y_1' are the solutions to the equation $2y' \equiv y \pmod 1$. Thus we can start from $2^{2n-1}\varphi$ and increase the precision as we proceed toward φ. The approximate values of $2^{j-1}\varphi$ $(j = 2n, 2n-1, \dots, 1)$ will allow us to make the correct choices.

Let $m = 2n$. For $j = 1, \dots, m$ we replace the known approximate value of $2^{j-1}\varphi$ by β_j, the closest number from the set $\left\{\frac{0}{8}, \frac{1}{8}, \frac{2}{8}, \frac{3}{8}, \frac{4}{8}, \frac{5}{8}, \frac{6}{8}, \frac{7}{8}\right\}$. This guarantees that

$$|2^{j-1}\varphi - \beta_j|_{\mathrm{mod}1} < 1/16 + 1/16 = 1/8.$$

Let us introduce a notation for binary fractions: $\overline{.\alpha_1 \cdots \alpha_p} = \sum_{j=1}^{p} 2^{-j}\alpha_j$ $(\alpha_j \in \{0,1\})$. Our algorithm is as follows.

Algorithm for sharpening the value of φ. Set $\overline{.\alpha_m\alpha_{m+1}\alpha_{m+2}} = \beta_m$ and proceed by iteration:

$$\alpha_j = \begin{cases} 0 & \text{if } \left|\overline{.0\alpha_{j+1}\alpha_{j+2}} - \beta_j\right|_{\mathrm{mod}1} < 1/4, \\ 1 & \text{if } \left|\overline{.1\alpha_{j+1}\alpha_{j+2}} - \beta_j\right|_{\mathrm{mod}1} < 1/4 \end{cases} \qquad \text{for } j = m-1, \dots, 1$$

(each time, exactly one of the two cases holds). The result satisfies the inequality

$$\left|\overline{.\alpha_1\alpha_2 \cdots \alpha_{m+2}} - \varphi\right|_{\mathrm{mod}1} < 2^{-(m+2)}.$$

The proof is a simple induction: $\left|\overline{.\alpha_j \cdots \alpha_{m+2}} - 2^{j-1}\varphi\right|_{\mathrm{mod}1} < 2^{-(m+3-j)}$ at each step.

This procedure is an example of computation by a finite-state automaton (see Problem 2.11). The state of the automaton is the pair $(\alpha_{j+1}, \alpha_{j+2})$, whereas the input symbols are β_j. It follows that the computation can be represented by a Boolean circuit of size $O(m)$ and depth $O(\log m)$.

13.5.4. ***Determining the exact value of the phase.*** We have found a number y satisfying $|y - k/t| < 1/2^{2n+1}$. We represent it as a continued fraction (see Section A.7) and try all convergents of y until we find a fraction k'/t' such that $|y - k'/t'| < 1/2^{2n+1}$. The second part of Theorem A.13 guarantees that the number k/t is contained among the convergents, and therefore will be found unless the algorithm stops earlier. But it cannot stop earlier because there is at most one fraction with denominator $\le 2^n$ that approximates a given number with precision $1/2^{2n+1}$. The running time of this algorithm is $O(n^3)$.

Important observations. 1. It is essential that the vector $|\xi_k\rangle$ does not deteriorate during the computation.

2. The entire period finding procedure depends on the parameters l and s; they should be adjusted so that the error probability be small enough. The error can occur in determining the period t as the least common denominator (see Lemma 13.2) or in estimating the cosine and the sine of φ_k with constant precision δ (see inequality (13.4)). The total probability of error does not exceed $3 \cdot 2^{-l} + nle^{-\Omega(s)}$. If it is required to get the result with probability of error $\le 1/3$, then we must set $l = 4$, $s = \Theta(\log n)$. In this way we get a quantum circuit of size $O(n^3 \log n)$. (In fact, there is some room for optimization; see Corollary 13.3.1 below.)

13.6. Discussion of the algorithm. We discuss two questions that arise naturally with regard to the algorithm that has been set forth.

— **Which eigenvalues do we find?** We find a randomly chosen eigenvalue. The distribution over the set of all eigenvalues can be controlled by appropriately choosing the initial state. In our period finding algorithm, it was $|1\rangle = \frac{1}{\sqrt{t}} \sum_{k=0}^{t-1} |\xi_k\rangle$, which corresponded to the uniform distribution on the set of eigenvalues associated with the orbit of 1.

— **Is it possible to find eigenvalues of other operators in the same way as in the algorithm for determining the period?** Let us be accurate: by *finding an eigenvalue with precision δ and error probability $\le \varepsilon$* we mean constructing a measuring operator with garbage, as in equation (12.3), where $j \in \Omega$ is an index of an eigenvalue $\lambda_j = e^{2\pi i \varphi_j}$, $\mathcal{L}_j$ is the corresponding eigenspace, and $y = \overline{.\alpha_1 \cdots \alpha_n}$ $(n = \lceil \log_2(1/\delta) \rceil)$. The conditional probabilities (12.4) should satisfy

$$\mathbf{Pr}\Big[|y - \varphi_j|_{\mathrm{mod}1} < \delta\Big] = \sum_{y:\, |y-\varphi_j|_{\mathrm{mod}1} < \delta} \mathbf{P}(y|j) \ge 1 - \varepsilon. \tag{13.5}$$

The answer to the question is "yes" — it is only necessary to implement $\Lambda(U)$, which is usually easy. (For example, if $U|0\rangle = |0\rangle$, we can use the result of Problem 8.4.) However, in general, the attainable precision is not

great and depends polynomially on the number of times the operator $\Lambda(U)$ is used. If one can efficiently compute the powers of U, e.g., if one can implement the operator

$$\Upsilon_m(U) \,:\, |p\rangle \otimes |\xi\rangle \mapsto |p\rangle \otimes U^p|\xi\rangle \quad (0 \le p < 2^m), \tag{13.6}$$

then the precision can be made exponential, $\delta = \exp(-\Omega(m))$.

13.7. Parallelized version of phase estimation. Applications. Remarkably, the phase estimation procedure (except for its last part — the continued fraction algorithm) can be realized by a quantum circuit of small depth. This result is due to R. Cleve and J. Watrous [**18**], but our proof is different from theirs.

Theorem 13.3. *Eigenvalues of a unitary operator U can be determined with precision $\delta = 2^{-n}$ and error probability $\le \varepsilon = 2^{-l}$ by an $O(n(l+\log n))$-size, $O(\log n + \log l)$-depth quantum circuit over the standard basis, with the additional gate $\Upsilon_m(U)$, $m = n + \log(l+\log n) + O(1)$. This gate is used in the circuit only once.*

Proof. At the core of the usual phase estimation procedure is a sequence of operators $\Lambda(U^{2^k})$, $k = 1, \dots, n-1$, applied to the same main register A with distinct control qubits $1, \dots, t$. (Here $t = 2ns$, which corresponds to $2n$ series of Bernoulli trials, each consisting of $s = \Theta(l + \log n)$ coin tosses. Each series is made to determine a single number $\cos(2^k\varphi_j)$ or $\sin(2^k\varphi_j)$ with the suitable constant precision and error probability $2^{-l}/(2n)$.) We need to parallelize these sequences. This can be done as follows: instead of applying the circuit

$$\Lambda(U^{p_t})[t, A] \cdots \Lambda(U^{p_1})[1, A],$$

we compute $p = p(u_1, \dots, u_t) = u_1p_1 + \cdots + u_tp_t$ (where $u_1, \dots, u_t \in \mathbb{B}$ are the values of the control qubits), use p as the control parameter for the operator $\Upsilon_m(U)$, and uncompute p.

To optimize the computation of p, we notice that each p_r is of the form $p_r = 2^{k_r}$. The terms in the sum $\sum_{r=1}^{t} u_r p_r$ can be divided into $2s$ groups in such a way that the numbers k_r be distinct within each group. Therefore, each group corresponds to an n-digit integer, and there are $2s = O(l + \log n)$ of such integers. The sum can be computed by a circuit of size $O(n(l+\log n))$ and depth $O(\log n + \log l)$ (see Problem 2.13a).

Let us estimate the complexity of the remaining part of the procedure. Each gate $\Lambda(U^{2^k})$ is accompanied by two H gates and possibly by one K gate, which contribute $O(t)$ to the size and $O(1)$ to the depth. Further, one needs to count the number of "heads" in each of the $2n$ series of coin flips. This is done by circuits of size $O(s)$ and depth $O(\log s)$. The subsequent trigonometric calculations are performed with constant precision, so each

instance of such calculation is done by a circuit of constant size. Finally, sharpening of the value of φ_j is carried out by a circuit of size $O(n)$ and depth $O(\log n)$. All these numbers stay within the required bounds. □

Unfortunately, Theorem 13.3 does not imply that the algorithms for period finding and factoring can be fully parallelized. However, one can derive the following corollary.

Corollary 13.3.1. *Period finding and factoring can be performed by a uniform sequence of $O(n^3)$-size, $O((\log n)^2)$-depth quantum circuits, with some classical pre-processing and post-processing. The pre-processing and post-processing are realized by uniform sequences of $O(n^3)$-size Boolean circuits.*

Note that if we use Definition 9.2, classical pre-processing does not count, since it can be included into the machine that generates the circuit. (However, the post-processing does count.) The division into three stages is clear from Table 13.1. The pre-processing amounts to modular exponentiation, $(q, a, p) \mapsto a^p \bmod q$. No small depth circuit is known for the solution of this problem. Thus we must compute the numbers $a^{2^j} \bmod q$ $(j = 0, \ldots, 2n-1)$ in advance. The post-processing includes finding the exact value of φ (by the continued fraction algorithm) and the calculation of the least common denominator (by Euclid's algorithm).

Proof of Corollary 13.3.1. The operator

$$\Upsilon(U_a) \,:\, \big|p, x\big\rangle \mapsto \big|p,\, (a^p x \bmod q)\big\rangle$$

is realized using the construction from the proof of Theorem 7.3. We need to compute $(a^p x \bmod q)$ and $(a^{-p} x \bmod q)$ by circuits of small depth. With pre-computed values of $(a^{2^j} \bmod q)$ and $(a^{-2^j} \bmod q)$, the computation of $(a^p x \bmod q)$ or $(a^{-p} x \bmod q)$ amounts to multiplying $O(n)$ numbers and calculating the residue mod q, which is done by a circuit of size $O(n^3)$ and depth $O((\log n)^2)$. □

Remark 13.1. R. Cleve and J. Watrous [**18**] also noticed that the depth can be decreased at the cost of increase in size. Indeed, the multiplication of $O(n)$ n-digit numbers can be performed with depth $O(\log n)$ and size $O(n^5(\log n)^2)$ (see [**9**]); therefore the same bound applies to period finding and factoring.

Now we can also prove Theorem 8.3. We will actually consider a more general situation: instead of realizing a single operator, we will try to simulate a circuit (see Theorem 13.5 below). Let us begin with a lemma.

Lemma 13.4. *The operator $\Upsilon_n(e^{2\pi i/2^n}) : |l\rangle \mapsto e^{2\pi i l/2^n}|l\rangle$ $(0 \le l < 2^n)$ can be realized with precision $\delta = 2^{-n}$ by an $O(n^2 \log n)$-size $O((\log n)^2)$-depth*

circuit C_n over the standard basis, using ancillas. The circuit C_n can be constructed algorithmically in time $\mathrm{poly}(n)$.

Proof. Let us assume that we have at our disposal an n-qubit register in the state

$$|\psi_{n,k}\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} \exp\left(-2\pi i \frac{kj}{2^n}\right) |j\rangle, \tag{13.7}$$

where k is odd. We will now see how it helps to achieve our goal of realizing the operator $\Upsilon_n(e^{2\pi i/2^n})$.

The vector $|\psi_{n,k}\rangle$ is an eigenvector of the permutation operator $X : |j\rangle \mapsto |(j+1) \bmod 2^n\rangle$,

$$X|\psi_{n,k}\rangle = e^{2\pi i\varphi_k}|\psi_{n,k}\rangle, \qquad \varphi_k = k/2^n.$$

Application of a power of X to the target state $|\psi_{n,k}\rangle$ results in multiplication by a phase factor,

$$X^p|\psi_{n,k}\rangle = e^{2\pi i(kp/2^n)}|\psi_{n,k}\rangle.$$

If k is odd, we can choose p to satisfy $kp \equiv l \pmod{2^n}$, which will provide the required phase factor $e^{2\pi il/2^n}$. Thus, for the realization of the operator $|l\rangle \mapsto e^{2\pi il/2^n}|l\rangle$ we use the value of l to compute p, apply the operator

$$\Upsilon_n(X) : |p, j\rangle \mapsto \big|p, (j+p) \bmod 2^n\big\rangle, \qquad p, j \in \{0, \dots, 2^n - 1\}, \tag{13.8}$$

controlled by this p, and "uncompute" p. The operator $\Upsilon_n(X)$ can be realized by a circuit of size $O(n)$ and depth $O(\log n)$ over the standard basis.

Ideally, we would want to use the vector $|\psi_{n,k}\rangle$ for some particular k, say, $k = 1$. But constructing such a vector is not easy, so we will start from a superposition of all odd values of k, namely,

$$|\eta\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|2^{n-1}\rangle = \frac{1}{\sqrt{2^{n-1}}} \sum_{s=1}^{2^{n-1}} |\psi_{n,2s-1}\rangle.$$

Then we will measure $k = 2s - 1$ and solve the equation for p. We now describe the required actions.

1. Create the vector $|\eta\rangle = \sigma^z[1]\, H[1]\, |0^n\rangle$.
2. Measure k with error probability $\le \varepsilon = \delta^2/4$. To find k, it suffices to determine the phase $\varphi_k = k/2^n$ with precision $\delta = 2^{-n}$. By Theorem 13.3, such phase estimation is realized by a circuit of size $O(n^2)$ and depth $O(\log n)$. The measured value should be odd, $k = 2s - 1$. (If it has happened to be even, set $k = 1$.)
3. Find $p = p(s, l)$ satisfying the equation $(2s - 1)p \equiv l \pmod{2^n}$ (see below).

4. Apply X^p to the n-bit register (which presumably contains $|\psi_{n,2s-1}\rangle$). This will effect the desired phase shift.
5. Reverse the computation done at Steps 1–3.

Apart from Step 1 and its reverse, the above procedure can be described symbolically as $W^{-1}VW$, where W represents Steps 2 and 3, and V represents Step 4. Hence the result of Problem 12.2 applies — the procedure realizes the operator $U = \Upsilon_n(e^{2\pi i/2^n})$ with precision $2\sqrt{\varepsilon} = \delta$.

Step 3 is the most demanding for resources. The solution to the equation $(2s-1)p \equiv l \pmod{2^n}$ can be obtained as follows:

$$p \equiv -l \sum_{j=0}^{m-1} (2s)^j \equiv -l \prod_{r=1}^{t-1} \bigl(1 + (2s)^{2^r}\bigr) \mod 2^n, \quad m = 2^t, \quad t = \lceil \log_2 n \rceil.$$

This calculation is done by a circuit of size $O(n^2 \log n)$ and depth $O((\log n)^2)$ (cf. solution to Problem 2.14a). □

Theorem 13.5. *Any circuit C of size L and depth d over a fixed finite basis $\mathcal{C}$ can be simulated with precision δ by an $O\bigl(Ln + n^2 \log n\bigr)$-size $O\bigl(d \log n + (\log n)^2\bigr)$-depth circuit $\tilde{C}$ over the standard basis (using ancillas), where $n = O(\log(L/\delta))$.*

Proof. Due to the results of Problems 8.1 and 8.2, each gate of the original basis $\mathcal{C}$ can be replaced by a constant size circuit over the basis $\mathcal{Q} \cup \bigl\{\Lambda(e^{i\varphi}) : \varphi \in \mathbb{R}\bigr\}$. Thus the circuit C is transformed into a circuit C' of size $L' = O(L)$ and depth $d' = O(d)$ over the new basis. Each gate $\Lambda(e^{i\varphi})$ can be approximated with precision $\delta' = \delta/(3L')$ by a gate of the form $\Lambda(e^{2\pi i l/2^n})$, where $n = \lceil \log_2(1/\delta') \rceil$, and l is an integer. The operator $\Lambda(e^{2\pi i l/2^n})$ is a special case of $\Upsilon_n(e^{2\pi i/2^n})$, hence we can use Lemma 13.4. However, the resulting circuit is somewhat larger than required, although it will suffice for the proof of Theorem 8.3 (which corresponds to the case $L = d = 1$).

To optimize the above simulation procedure, let us examine the proof of Lemma 13.4. Most of the resource usage can attributed to solving the equation $kp \equiv l \pmod{2^n}$. But this step is redundant if $k = 1$. In fact, the operator $\Upsilon_n(e^{2\pi i/2^n})$ can be realized by applying $\Upsilon_n(X)$ (see (13.8)) to the target state $|\psi_{n,1}\rangle$; this is done by a circuit of size $O(n)$ and depth $O(\log n)$. Thus we need to create L' copies of the state $|\psi_{n,1}\rangle$ and use one copy per gate in the simulation of the circuit C'. The exact sequence of actions is as follows.

1. Create the state $|\psi_{n,0}\rangle = H^{\otimes n}|0^n\rangle$.
2. Turn it into $|\psi_{n,1}\rangle = \Upsilon_n(e^{-2\pi i/2^n})|\psi_{n,0}\rangle$ by the procedure of Lemma 13.4. This is done with precision $\delta' = 2^{-n} \le \delta/3$. The corresponding circuit has size $O(n^2 \log n)$ and depth $O((\log n)^2)$.

3. Make L' copies of the state $|\psi_{n,1}\rangle$ out of one copy (see below).
4. Simulate the circuit C' with precision $\delta/3$, using one copy of $|\psi_{n,1}\rangle$ per gate.
5. Reverse Steps 1–3.

To produce multiple copies of the state $|\psi_{n,1}\rangle$, we can use the equation

$$|\psi_{n,k}\rangle^{\otimes m} = W^{-1}\Big(|\psi_{n,0}\rangle^{\otimes(m-1)} \otimes |\psi_{n,k}\rangle\Big),$$
$$W \,:\, |x_1,\dots,x_{m-1},x_m\rangle \mapsto |x_1,\dots,x_{m-1},\, x_1+\dots+x_m\rangle.$$

The operators W and W^{-1} are realized using the construction from the proof of Theorem 7.3. This involves the addition of m n-digit numbers, which is done by a Boolean circuit of size $O(nm)$ and depth $O(\log n + \log m)$ (see Problem 2.13a). In our case $m = L' = O(L)$. The overall size and depth of the resulting quantum circuit are as required. □

[3!] **Problem 13.4.** Let $q \geq 1,\ n = \lceil \log_2 q \rceil,\ \delta = 2^{-l}$. Realize the Fourier transform on the group $\mathbb{Z}_q$ with precision δ by a quantum circuit of size $\mathrm{poly}(n,l)$ and depth $\mathrm{poly}(\log n, \log l)$ over the standard basis. Estimate the size and the depth of the circuit more accurately. (For definition of the quantum Fourier transform, see Problem 9.4c.)

13.8. The hidden subgroup problem for $\mathbb{Z}^k$. The algorithms discovered by Simon and Shor can be generalized to a rather broad class of problems connected with Abelian groups. The most general of these is the hidden subgroup problem for $\mathbb{Z}^k$ [**12**], to which the hidden subgroup problem in an arbitrary finitely generated Abelian group G can be reduced. (Indeed, G can be represented as a quotient group of $\mathbb{Z}^k$ for some k.)

A "hidden subgroup" $D \subseteq \mathbb{Z}^k$ (as defined on page 117) has finite index: the order of the group $E = \mathbb{Z}^k/D$ does not exceed 2^n. Therefore $D \cong \mathbb{Z}^k$. From the computational viewpoint, D is given by a basis $(g_1,\dots,g_k)$ whose binary representation has length $\mathrm{poly}(k,n)$. Any such basis gives a solution to the problem. (The equivalence of two bases can be verified by a polynomial algorithm.)

The problem of computing the period is a special case of the hidden subgroup problem in $\mathbb{Z}$. Recall that $\mathrm{per}_q(a) = \min\{t \geq 1 : a^t \equiv 1 \pmod q\}$. The function $f : x \mapsto a^x \pmod q$ satisfies condition (13.1), where $D = \{m\ \mathrm{per}_q(a) : m \in \mathbb{Z}\}$. This function is polynomially computable, hence an arbitrary polynomial algorithm for finding a hidden subgroup can be transformed into a polynomial algorithm for calculating the period.

The well-known problem of calculating the discrete logarithm can be reduced to the hidden subgroup problem for $\mathbb{Z}^2$. The smallest positive integer s such that $\zeta^s = a$, where ζ is a generator of the group $(\mathbb{Z}/q\mathbb{Z})^*$, is

called the *discrete logarithm* of a number a at base ζ. Consider the function $f : (x_1, x_2) \mapsto \zeta^{x_1} a^{x_2} \bmod q$. This function also satisfies condition (13.1), where $D = \{(x_1, x_2) \in \mathbb{Z}^2 : \zeta^{x_1} a^{x_2} \equiv 1 \bmod q\}$. If we know a basis of the subgroup $D \subseteq \mathbb{Z}^2$, it is easy to find an element of the form $(s, -1) \in D$. Then $\zeta^s = a$, i.e., s is the discrete logarithm of a at base ζ.

Let us describe a quantum algorithm that solves the hidden subgroup problem for $G = \mathbb{Z}^k$. It is analogous to the algorithm for the case $G = (\mathbb{Z}_2)^k$, but instead of the operator $H^{\otimes k}$ we use the procedure for measuring the eigenvalues. Instead of a basis for the group D we will look for a system of generators of the *character group* $E^* = \mathrm{Hom}(E, U(1))$ (the transition from E^* to D is realized by a polynomial algorithm; see, for example, [**54**, vol. 1]). The character

$$(g_1, \dots, g_k) \mapsto \exp\Bigl(2\pi i \sum_j \varphi_j g_j\Bigr)$$

is determined by the set $\varphi_1, \dots, \varphi_k$ of numbers modulo 1. These are rational numbers with denominators not exceeding $|E^*| \le 2^n$.

If we produce $l = n + 3$ uniformly distributed random characters $(\varphi_1^{(1)}, \dots, \varphi_k^{(1)}), \dots, (\varphi_1^{(l)}, \dots, \varphi_k^{(l)})$, then they will generate the entire group E^* with probability $\ge 1 - 1/2^{l-n} = 1 - 1/8$ (see Problem 13.1). It suffices to know each $\varphi_j^{(r)}$ with precision δ and the probability of error $\le \varepsilon$, where

$$\delta \le \frac{1}{2^{2n+1}}, \qquad \varepsilon \le \frac{1}{5kl}. \tag{13.9}$$

The last condition guarantees that the total probability of error does not exceed $1/8 + 1/5 < 1/3$.

Let us choose a sufficiently large number $M = 2^m$ (a concrete estimate can be obtained by analyzing the algorithm). We will work with integers between 0 to $M - 1$.

Let us prepare, in one quantum register of length km, the states

$$|\xi\rangle = M^{-k/2} \sum_{g \in \Delta} |g\rangle, \text{ where } \Delta = \{0, \dots, M-1\}^k.$$

In another register we put $|0^n\rangle$. We apply the quantum oracle (13.2) and then discard the second register. We obtain the mixed state

$$\rho = \mathrm{Tr}_{[km+1,\dots,km+n]}\Bigl(U\bigl(|\xi\rangle\langle\xi| \otimes |0^n\rangle\langle 0^n|\bigr)U^\dagger\Bigr) = M^{-k} \sum_{g,h \in \Delta:\, g-h \in D} |g\rangle\langle h|.$$

Now we are going to measure the eigenvalues of the shift (mod M) operators

$$V_j : \bigl(g_1, \dots, g_j, \dots, g_k\bigr) \mapsto \bigl(g_1, \dots, (g_j + 1) \bmod M, \dots, g_k\bigr)$$

(only the j-th component changes). These operators commute, so that they have a common basis of eigenvectors, and therefore we can determine their eigenvalues simultaneously. The eigenvalues have the form $e^{2\pi i s_j/M}$. The corresponding eigenvectors are

$$|\xi_{s_1,\dots,s_k}\rangle \;=\; M^{-k/2} \sum_{(g_1,\dots,g_k)\in\Delta} \exp\Big(-2\pi i \sum_{j=1}^{k} \frac{g_j s_j}{M}\Big)\, |g_1,\dots,g_k\rangle.$$

The probability that a given set $(s_1,\dots,s_k)$ will be realized equals

$$\begin{aligned}\mathbf{P}(\rho,\, \mathcal{L}_{s_1,\dots,s_k}) &= \langle \xi_{s_1,\dots,s_k}|\, \rho\, |\xi_{s_1,\dots,s_k}\rangle \\ &= M^{-2k} \sum_{g,h\in\mathbb{Z}^k} \chi_D(g-h)\, \chi_\Delta(g)\chi_\Delta(h)\, \exp\Big(2\pi i \sum_{j=1}^{k} \frac{(g_j-h_j)s_j}{M}\Big),\end{aligned}$$

where $\chi_A(\cdot)$ denotes the characteristic function of the set A. The Fourier transform of the product is the convolution of the Fourier transforms of the factors. Therefore,

$$\mathbf{P}(\rho,\, \mathcal{L}_{s_1,\dots,s_k}) = \frac{1}{|E^*|} \sum_{(\varphi_1,\dots,\varphi_k)\in E^*} p_{\varphi_1,\dots,\varphi_k}(s_1,\dots,s_k), \tag{13.10}$$

where

$$p_{\varphi_1,\dots,\varphi_k}(s_1,\dots,s_k) = \prod_{j=1}^{k} \left(\frac{\sin(M\pi(s_j/M-\varphi_j))}{M\sin(\pi(s_j/M-\varphi_j))}\right)^2.$$

For a given element $(\varphi_1,\dots,\varphi_k)\in E^*$ the function $p_{\varphi_1,\dots,\varphi_k}$ is a probability distribution. Therefore equation (13.10) can be modeled by the following process: first, a random uniformly distributed element $(\varphi_1,\dots,\varphi_k)\in E^*$ is generated; second, the parameters $s_1,\dots,s_k$ are set according to the conditional probabilities $p_{\varphi_1,\dots,\varphi_k}(s_1,\dots,s_k)$. The conditional probabilities have the following property:

$$\mathbf{Pr}\Big[|s_j/M-\varphi_j|>\beta\Big] \le \frac{1}{M\beta}$$

for any $\beta>0$. If we estimate the quantities s_j/M with precision β and error probability $\le 1/M\beta$, we obtain the values of $\varphi_1,\dots,\varphi_k$ with precision $\delta=2\beta$ and error probability $\le \varepsilon = 2/M\beta$. It remains to choose the numbers M and β so that inequality (13.9) be satisfied.

Complexity of the algorithm. We need $O(n)$ queries to the oracle, each query being of length $O(k(n+\log k))$. The size of the quantum circuit is estimated as $O(kn^3)\,\mathrm{poly}(\log k,\log n)$.

14. The quantum analogue of NP: the class BQNP

It is possible to construct quantum analogues not only for the class P, but also for other classical complexity classes. This is not a routine process, but suitable generalizations often come up naturally. We will consider the class NP as an example. (For another example — the class IP and its quantum analogue QIP — see [**72, 38**]. We also mention that the quantum analogue of PSPACE equals PSPACE [**71**].)

14.1. Modification of classical definitions. Quantum computation, as well as probabilistic computation, is more naturally described using partially defined functions. Earlier we made do without this concept so as not to complicate matters by the inclusion of extraneous detail, but now we need it.

A *partially defined Boolean function* is a function

$$F : \mathbb{B}^n \to \{0,\ 1,\ \text{“undefined”}\}.$$

In this section it will be tacitly understood that by Boolean function we mean partially defined Boolean function.

One more comment regarding notation: we have used the symbol P both for the class of polynomially computable functions and for the class of polynomially decidable predicates; now we act analogously, using the notations P, NP, etc. for classes of partially defined functions.

P, of course, denotes the class of polynomially computable partially defined functions. We introduce a modified definition of the class NP.

Definition 14.1. A function $F : \mathbb{B}^n \to \{0,\ 1,\ \text{“undefined”}\}$ belongs to the class NP if there is a partially defined function $R \in \mathrm{P}$ in two variables such that

$$\begin{aligned} F(x) = 1 &\implies \exists\, y\, \big((|y| < q(|x|)) \wedge (R(x,y) = 1)\big) \\ F(x) = 0 &\implies \forall\, y\, \big((|y| < q(|x|)) \Rightarrow (R(x,y) = 0)\big). \end{aligned}$$

As before, $q(\cdot)$ is a polynomial.

What would change if in Definition 14.1 we replaced the condition $R \in \mathrm{P}$ by the condition $R \in \mathrm{BPP}$? First of all, we would get a different, broader, class, which we could denote by BNP. However, for this class there is another, standard, notation — MA, indicating that it falls into a hierarchy of classes defined by *Arthur-Merlin games*. We have mentioned Arthur and Merlin in connection with the definition of NP. We have also discussed games corresponding to other complexity classes (see Section 5.1). Traditionally, the term “Arthur-Merlin games” is used for probabilistic games in which Arthur is a polynomial Turing machine whereas Merlin is all-powerful;

before each move Arthur flips coins so that both players see them. The order of the letters in the symbol MA indicates the order of the moves: at first Merlin communicates y , then Arthur checks the truth of the predicate $R(x, y)$, by a polynomial probabilistic computation. The message y is sometimes called a "proof"; it may be hard to find but easy to check.

14.2. Quantum definition by analogy.

Definition 14.2. A function $F : \mathbb{B}^n \to \{0, 1, \text{"undefined"}\}$ belongs to the class BQNP if there exists a polynomial classical algorithm that computes a function $x \mapsto Z(x)$, where $Z(x)$ is a description of a quantum circuit, realizing an operator $U_x : \mathcal{B}^{\otimes N_x} \to \mathcal{B}^{\otimes N_x}$ such that

$$F(x) = 1 \implies \exists\, |\xi\rangle \in \mathcal{B}^{\otimes m_x}\ \mathbf{P}\Big(U_x|\xi\rangle \otimes |0^{N_x - m_x}\rangle,\ \mathcal{M}\Big) \geq p_1,$$
$$F(x) = 0 \implies \forall\, |\xi\rangle \in \mathcal{B}^{\otimes m_x}\ \mathbf{P}\Big(U_x|\xi\rangle \otimes |0^{N_x - m_x}\rangle,\ \mathcal{M}\Big) \leq p_0.$$

Here $\mathcal{M} = \mathbb{C}\big(|1\rangle\big) \otimes \mathcal{B}^{\otimes(N_x - 1)}$, and p_0, p_1 satisfy the condition $p_1 - p_0 \geq \Omega(n^{-\alpha})$ for some constant $\alpha \geq 0$. The quantifiers of $|\xi\rangle$ include only vectors of unit length. (We will use an analogous convention further on in this section, pushing numeric factors outside the $|\cdot\rangle$ sign.)

The vector $|\xi\rangle$ plays the role of y in the previous definition. Note that $m_x \leq N_x \leq |Z(x)| = \text{poly}(|x|)$ since the algorithm is polynomial.

In effect, the very same game of Merlin with Arthur is taking place, but now it is governed by the laws of quantum mechanics. Merlin sends a quantum message (the state $|\xi\rangle$) to Arthur, who checks it by applying the operator U_x. A suitable message will convince Arthur that $F(x) = 1$ (if this is actually so) with probability $\geq p_1$. But if $F(x) = 0$, Merlin cannot succeed in convincing Arthur to the contrary with probability higher than p_0, whatever message he sends. Instead of a pure state $|\xi\rangle$, we can allow Merlin to send an arbitrary density matrix — the maximum of the probability is achieved on a pure state anyway.

In Definition 14.2, we have the same flexibility in choosing the threshold probabilities p_0 and p_1 as in the definitions of BPP and BQP.

Lemma 14.1 (amplification of probabilities). *If $F \in$ BQNP, then it likewise satisfies a variant of Definition* 14.2 *in which the numbers p_0, p_1 ($p_1 - p_0 = \Omega(n^{-\alpha})$) are replaced by*

$$p_1' = 1 - \varepsilon, \quad p_0' = \varepsilon, \qquad \varepsilon = \exp(-\Omega(n^\beta)),$$

where β is an arbitrary positive constant.

Proof. The general idea of amplifying the probabilities remains as before: we consider $k = \text{poly}(n)$ copies of the circuit realizing the operator $U = U_x$.

To the results of their work we apply a variant of the majority function, with the threshold value adjusted so as to separate p_0 from p_1:

$$G(z_1,\dots,z_k)=\begin{cases}1 & \text{if } \sum_{j=1}^k z_j \geq pk,\\ 0 & \text{if } \sum_{j=1}^k z_j < pk,\end{cases} \tag{14.1}$$

where $p=(p_0+p_1)/2$. But now there appears an additional difficulty: Merlin may attempt to deceive Arthur by sending him a message that is not factorable into the tensor product.

Let us grant Merlin greater freedom, allowing him to submit any density matrix $\rho\in\mathbf{L}(\mathcal{B}^{\otimes km})$. The probability of obtaining outcomes $z_1,\dots,z_k$ by applying k copies of U to the message ρ is as follows:

$$\mathbf{P}(z_1,\dots,z_k|\,\rho)=\operatorname{Tr}\bigl(X^{(z_1)}\otimes\cdots\otimes X^{(z_k)}\rho\bigr), \tag{14.2}$$

where

$$X^{(a)}=\operatorname{Tr}_{[m+1,\dots,N]}\Bigl(U^\dagger\Pi_1^{(a)}U\,\bigl(I_{\mathcal{B}^{\otimes m}}\otimes|0^{N-m}\rangle\langle 0^{N-m}|\bigr)\Bigr). \tag{14.3}$$

Here $\Pi_1^{(a)}$ is the projection onto the subspace of states having a in the first qubit (i.e., $\mathbb{C}\bigl(|a\rangle\bigr)\otimes\mathcal{B}^{\otimes(N-1)}$).

If $F(x)=1$, Merlin can simply send the state $\rho=\rho_x^{\otimes k}$, where $\rho_x=|\xi_x\rangle\langle\xi_x|$ is the message that would convince Arthur with probability $\geq p_1$ in the original version of the game (with a single copy of U). By the general properties of quantum probability, formula (14.2) takes the form

$$\mathbf{P}(z_1,\dots,z_k|\,\rho)=\prod_{j=1}^k\operatorname{Tr}(X^{(z_j)}\rho_x)=\prod_{j=1}^k\mathbf{P}(z_j|\rho_x).$$

We will derive a lower bound for this quantity from a more general analysis given below.

In the opposite case, $F(x)=0$, we will obtain an upper bound for the probability $\mathbf{P}(z_1,\dots,z_k|\rho)$ over all density matrices ρ.

Let us select an orthonormal basis in the space $\mathcal{B}^{\otimes m}$, in which the operator $X^{(1)}$ is diagonalized (this operator is clearly Hermitian). The operator $X^{(0)}=I-X^{(1)}$ is diagonal in the same basis. We define a set of "conditional probabilities" $p(z|d)=\mathbf{P}\bigl(z\bigl||d\rangle\langle d|\bigr)=\langle d|X^{(z)}|d\rangle$, where $|d\rangle$ is one of the basis vectors. It is obvious that $p(z|d)\geq 0$ and $p(0|d)+p(1|d)=1$. In this notation, the quantity $\mathbf{P}(z_1,\dots,z_k|\rho)$ becomes

$$\mathbf{P}(z_1,\dots,z_k|\,\rho)=\sum_{d_1,\dots,d_k}p_{d_1,\dots,d_k}\,p(z_1|d_1)\cdots p(z_k|d_k),\qquad\sum_{d_1,\dots,d_k}p_{d_1,\dots,d_k}=1,$$

where $p_{d_1,\dots,d_k}=\langle d_1,\dots,d_k|\rho|d_1,\dots,d_k\rangle$.

This formula has the following interpretation. Consider the set of probabilities $\mathbf{P}(z_1,\dots,z_k|\rho)$ for all sequences $(z_1,\dots,z_k)\in\mathbb{B}^k$ as a vector

in a 2^k-dimensional real space; we denote this vector by $\overrightarrow{\mathbf{P}}(\rho) \in \mathbb{R}^{\mathbb{B}^k}$. We have shown that for a general density matrix ρ the vector $\overrightarrow{\mathbf{P}}(\rho)$ belongs to the convex hull of such vectors corresponding to product states, namely, $|d_1\rangle\langle d_1| \otimes \cdots \otimes |d_k\rangle\langle d_k|$. Therefore the probability of the event $G(z_1, \ldots, z_k) = 1$,

$$\mathbf{Pr}\big[G(z_1, \ldots, z_k) = 1 \big| \rho\big] = \sum_{z \in \mathbb{B}^k} G(z)\mathbf{P}(z|\rho) = \Big(\overrightarrow{G}, \overrightarrow{\mathbf{P}}(\rho)\Big),$$

achieves its maximum at a density matrix of this special type.

In the case where G is the threshold function (14.1),

$$p_{\max} \stackrel{\text{def}}{=} \max_{\rho} \mathbf{Pr}\big[G(z_1, \ldots, z_k) = 1 \big| \rho\big] = \sum_{j \geq l} \binom{k}{j} p_*^j (1 - p_*)^{k-j},$$

where $p_* = \max_{|\xi\rangle} \langle \xi | X^{(1)} | \xi \rangle$. The number $p_{\max}$ equals the probability of getting $\geq pk$ "heads" for k coins tossed, $\mathbf{Pr}\Big[k^{-1} \sum_{j=1}^k v_j \geq p\Big]$, where $\mathbf{Pr}\big[v_j = 1\big] = p_*$. This probability can be estimated using Chernoff's inequality (13.4). Thus we obtain[11]

$$\begin{aligned} p_{\max} &\leq \exp(-2(p - p_*)^2 k) && \text{if } p \geq p_*, \\ p_{\max} &\geq 1 - \exp(-2(p - p_*)^2 k) && \text{if } p \leq p_*. \end{aligned}$$

According to the assumptions of the lemma, $p_* \leq p_0$ if $F(x) = 0$, and $p_* \geq p_1$ if $F(x) = 1$. We have chosen p so that $p - p_0 \geq \Omega(n^{-\alpha})$ and $p_1 - p \geq \Omega(n^{-\alpha})$. Choosing $k = cn^{2\alpha+\beta}$ (for a suitable constant c), we get exactly the estimate which is required,

$$\begin{aligned} p_{\max} &\leq \exp(-\Omega(n^\beta)) && \text{if } F(x) = 0, \\ p_{\max} &\geq 1 - \exp(-\Omega(n^\beta)) && \text{if } F(x) = 1. \end{aligned}$$

□

Remark 14.1. An important point in the proof was the fact that $X^{(0)}$ and $X^{(1)}$ are diagonalized over the same basis. In general, the amplification of probability for nontrivial complexity classes (both classical and quantum) is a rather subtle thing.

14.3. Complete problems. Similarly to the class NP, the class BQNP has complete problems. Completeness is understood with respect to the same polynomial reduction that we considered earlier (i.e., Karp reduction; see Definition 3.4). Here is the simplest example.

[11] We have omitted the factor 2 from (13.4) because it includes both cases that are now considered separately (see the proof of Chernoff's inequality in the solution to Problem 13.3). This is an unimportant factor anyway.

PROBLEM 0. Consider a function F which is defined on a subset of the words of this form:

$$z = \big((\text{description of a quantum circuit } U),\, p_0, p_1\big),$$

where by description of a circuit we mean its approximate realization in the standard basis, and p_0, p_1 are such that $p_1 - p_0 \geq \Omega(n^{-\alpha})$ (n is the size of the circuit, $\alpha > 0$ is a constant). The function F is defined as follows:

$F(z) = 1 \iff$ there exists a vector $|\xi\rangle$, on which we get 1 in the first bit with probability greater than p_1;

$F(z) = 0 \iff$ for all $|\xi\rangle$ the probability of getting 1 in the first bit is smaller than p_0.

The completeness of Problem 0 is obvious: by saying that the problem is complete we just rephrase Definition 14.2.

We now consider more interesting examples. To start, we define a quantum analog of 3-CNF — the local Hamiltonian (locality is the analogue of the condition that the number of variables in each clause is bounded).

Definition 14.3. An operator $H : \mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes n}$ is called a *k-local Hamiltonian* if it is expressible in the form

$$H = \sum_j H_j[S_j],$$

where each term $H_j \in \mathbf{L}(\mathcal{B}^{\otimes |S_j|})$ is a Hermitian operator acting on a set of qubits S_j, $|S_j| \leq k$.

In addition, we put a normalization condition, namely, $0 \leq H_j \leq 1$, meaning that both H_j and $I - H_j$ are nonnegative.

PROBLEM 1: THE LOCAL HAMILTONIAN. Let

$$z = \big(\text{description of a } k\text{-local Hamiltonian } H,\, a, b\big),$$

where $k = O(1)$, $0 \leq a < b$, $b - a = \Omega(n^{-\alpha})$ ($\alpha > 0$ is a constant). Then

$$F(x) = 1 \iff H \text{ has an eigenvalue not exceeding } a,$$
$$F(x) = 0 \iff \text{all eigenvalues of } H \text{ are greater than } b.$$

Proposition 14.2. *The problem* LOCAL HAMILTONIAN *belongs to* BQNP.

Proof. At the outset we describe the general idea. We construct a circuit W that can be applied to a state $|\eta\rangle \in \mathcal{B}^{\otimes n}$ so as to produce a result 1 or 0 ("yes" or"no"): it says whether Arthur accepts the submitted state or not. The answer "yes" will occur with probability $p = 1 - r^{-1}\langle\eta|H|\eta\rangle$, where r is the number of terms in the Hamiltonian H. If $|\eta\rangle$ is an eigenvector

corresponding to an eigenvalue $\lambda \le a$, then the probability of the answer "yes" is

$$p = 1 - r^{-1}\langle\eta|H|\eta\rangle = 1 - r^{-1}\lambda \ge 1 - r^{-1}a,$$

and if every eigenvalue of H exceeds b, then

$$p = 1 - r^{-1}\langle\eta|H|\eta\rangle \le 1 - r^{-1}b.$$

At first we construct such a circuit for a single term. This will be just a realization of the POVM measurement corresponding to the decomposition $I = H_j + (I - H_j)$. We could use the general result about POVM measurements (see Problem 11.8), but let us give an explicit construction from scratch.

Let $H_j = \sum_s \lambda_s |\psi_s\rangle\langle\psi_s|$. This operator acts on a bounded number of qubits, $|S_j| \le k$. Therefore we can realize the operator

$$W_j : |\psi_s, 0\rangle \mapsto |\psi_s\rangle \otimes \Big(\sqrt{\lambda_s}|0\rangle + \sqrt{1-\lambda_s}|1\rangle\Big)$$

by a circuit of constant size. It acts on the set of qubits $S_j \cup \{\text{"answer"}\}$, where "answer" denotes the qubit that will contain the measurement outcome.

We compute the probability that the outcome is 1. Let $|\eta\rangle = \sum_s y_s|\psi_s\rangle$ be the expansion of $|\eta\rangle$ in the orthogonal system of eigenvectors of H_j. We have, by definition of the probability,

$$\begin{aligned}
\mathbf{P}_j(1) &= \langle\eta, 0|\, W_j^\dagger \big(I \otimes \underbrace{|1\rangle\langle 1|}_{\text{answer}}\big) W_j\, |\eta, 0\rangle \\
&= \left(\sum_s y_s^* \langle\psi_s, 0|\right) W_j^\dagger \big(I \otimes \underbrace{|1\rangle\langle 1|}_{\text{answer}}\big) W_j \left(\sum_t y_t\, |\psi_t, 0\rangle\right) \\
&= \sum_{s,t} \sqrt{1-\lambda_s}\; y_s^* \sqrt{1-\lambda_t}\; y_t\; \langle\psi_s|\psi_t\rangle \;=\; \sum_s (1-\lambda_s) y_s^* y_s \\
&= 1 - \langle\eta|H_j|\eta\rangle.
\end{aligned}$$

The general circuit W chooses the integer j randomly and uniformly, after which it applies the corresponding operator W_j. This procedure can be realized by the measuring operator $\sum_j |j\rangle\langle j| \otimes W_j$, applied to the initial vector $\Big(\frac{1}{\sqrt{r}} \sum_j |j\rangle\Big) \otimes |\eta, 0\rangle$. (Here $|j\rangle$ denotes a basis vector in an auxiliary r-dimensional space.) The probability of getting the outcome 1 is

$$\mathbf{P}(1) = \sum_j \frac{1}{r} \mathbf{P}_j(1) = \sum_j \frac{1}{r} \big(1 - \langle\eta|H_j|\eta\rangle\big) = 1 - r^{-1}\langle\eta|H|\eta\rangle.$$

□

14.4. Local Hamiltonian is BQNP-complete.

Theorem 14.3. *The problem* LOCAL HAMILTONIAN *is* BQNP*-complete with respect to the Karp reduction.*

The rest of this section constitutes a proof of this theorem. The main idea goes back to Feynman [**24**]: replacing a unitary evolution by a time independent Hamiltonian (i.e., transition from the circuit to a local Hamiltonian).

Thus, suppose we have a circuit $U = U_L \cdots U_1$ of size L. We will assume that U acts on N qubits, the first m of which initially contain Merlin's message $|\xi\rangle$, the rest being initialized by 0. The gates U_j act on pairs of qubits.

14.4.1. ***The Hamiltonian associated with the circuit.*** It acts on the space

$$\mathcal{L} = \mathcal{B}^{\otimes N} \otimes \mathbb{C}^{L+1},$$

where the first factor is the space on which the circuit acts, whereas the second factor is the space of a step counter (clock). The Hamiltonian consists of three terms which will be defined later,

$$H = H_{\text{in}} + H_{\text{prop}} + H_{\text{out}}.$$

We are interested in the minimum eigenvalue of this Hamiltonian, or the minimum of the *cost function* $f\big(|\eta\rangle\big) = \langle\eta|H|\eta\rangle$ over all vectors $|\eta\rangle$ of unit length. We will try to arrange that the Hamiltonian has a small eigenvalue if and only if there exists a quantum state $|\xi\rangle \in \mathcal{B}^{\otimes m}$ causing U to output 1 with high probability. In such a case, the minimizing vector $|\eta\rangle$ will be related to that $|\xi\rangle$ in the following way:

$$|\eta\rangle = \frac{1}{\sqrt{L+1}} \sum_{j=0}^{L} U_j \cdots U_1 |\xi, 0\rangle \otimes |j\rangle.$$

In constructing the terms of the Hamiltonian, we will try to "enforce" this structure of the vector $|\eta\rangle$ by imposing "penalties" that increase the cost function whenever $|\eta\rangle$ deviates from the indicated form.

The term H_{in} corresponds to the condition that, at step 0, all the qubits but m are in state $|0\rangle$. Specifically,

$$H_{\text{in}} = \left(\sum_{s=m+1}^{N} \Pi_s^{(1)} \right) \otimes |0\rangle\langle 0|, \tag{14.4}$$

where $\Pi_s^{(\alpha)}$ is the projection onto the subspace of vectors for which the s-th qubit equals α. The second factor in this formula acts on the space of the counter. (Informally speaking, the term $\Pi_s^{(1)} \otimes |0\rangle\langle 0|$ "collects a penalty" by

adding 1 to the cost function whenever the s-th qubit is in state $|1\rangle$ while the counter being in state $|0\rangle$.)

The term H_{out} corresponds to the final state and equals

$$H_{\text{out}} = \Pi_1^{(0)} \otimes |L\rangle\langle L|. \tag{14.5}$$

Here we assume that the bit of the result has number 1. (That is, at step L the first qubit should be in state $|1\rangle$, or a penalty will be imposed.)

And, finally, the term H_{prop} describes the propagation of a quantum state through the circuit. It consists of L terms, each of which corresponds to the transition from $j-1$ to j:

$$H_{\text{prop}} = \sum_{j=1}^{L} H_j, \tag{14.6}$$

$$H_j = -\frac{1}{2}U_j \otimes |j\rangle\langle j-1| - \frac{1}{2}U_j^\dagger \otimes |j-1\rangle\langle j| + \frac{1}{2}I \otimes \big(|j\rangle\langle j| + |j-1\rangle\langle j-1|\big).$$

Each term H_j acts on two qubits of the space $\mathcal{B}^{\otimes N}$, as well as on the space of the counter (the latter is not represented by qubits yet).

14.4.2. ***Change of basis.*** We effect the change of basis given by the operator

$$W = \sum_{j=0}^{L} U_j \cdots U_1 \otimes |j\rangle\langle j|.$$

(It is worth mentioning that W is a measuring operator with respect to the the value of the counter j.) The change of basis means that we represent the vector $|\eta\rangle$ in the form $|\eta\rangle = W|\widetilde{\eta}\rangle$; from now on, we are dealing with $|\widetilde{\eta}\rangle$ instead of $|\eta\rangle$. Under such a change, the Hamiltonian is transformed into its conjugate, $\widetilde{H} = W^\dagger H W$. We consider how the conjugation by the operator W acts on the terms of H.

On the term H_{in} the conjugation has no effect:

$$\widetilde{H}_{\text{in}} = W^\dagger H_{\text{in}} W = H_{\text{in}}. \tag{14.7}$$

The action on the term H_{out} is:

$$\widetilde{H}_{\text{out}} = W^\dagger H_{\text{out}} W = \Big(U^\dagger \Pi_1^{(0)} U\Big) \otimes |L\rangle\langle L|. \tag{14.8}$$

Each operator H_j in (14.6) is the sum of three terms. Let us write the action of the conjugation on the first of them:

$$\begin{aligned} W^\dagger\big(U_j \otimes |j\rangle\langle j{-}1|\big)W \\ &= \sum_{p,t} \big(U_p \cdots U_1 \otimes |p\rangle\langle p|\big)^\dagger \big(U_j \otimes |j\rangle\langle j{-}1|\big)\big(U_t \cdots U_1 \otimes |t\rangle\langle t|\big) \\ &= \Big((U_j \cdots U_1)^\dagger U_j \,(U_{j-1} \cdots U_1)\Big) \otimes \Big(\big(|j\rangle\langle j|\big)^\dagger |j\rangle\langle j{-}1| \,\big(|j{-}1\rangle\langle j{-}1|\big)\Big) \\ &= I \otimes |j\rangle\langle j{-}1|. \end{aligned}$$

Conjugation of the two other terms proceeds analogously, so that we obtain

$$\begin{aligned} \widetilde{H}_j &= W^\dagger H_j W \\ &= I \otimes \frac{1}{2}\Big(|j{-}1\rangle\langle j{-}1| - |j{-}1\rangle\langle j| - |j\rangle\langle j{-}1| + |j\rangle\langle j|\Big) = I \otimes E_j, \end{aligned}$$

$$\widetilde{H}_{\text{prop}} = W^\dagger H_{\text{prop}} W = I \otimes E, \tag{14.9}$$

where

$$E = \sum_{j=1}^{L} E_j = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & & & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} & & \\ & -\frac{1}{2} & 1 & -\frac{1}{2} & \\ & & -\frac{1}{2} & \ddots & \ddots \\ 0 & & & \ddots & \ddots \end{pmatrix}.$$

14.4.3. ***Existence of a small eigenvalue if the answer is "yes".*** Suppose that the circuit U gives the outcome 1 ("yes") with probability $\geq 1 - \varepsilon$ on some input vector $|\xi\rangle$. This, by definition, means that

$$\mathbf{P}(0) = \big\langle \xi, 0 \big| U^\dagger \Pi_1^{(0)} U \big| \xi, 0 \big\rangle \leq \varepsilon.$$

We want to prove that in this case $\widetilde{H}$ (and so also H) has a small eigenvalue. For this, it is sufficient to find a vector $|\widetilde{\eta}\rangle$ such that $\langle\widetilde{\eta}|\widetilde{H}|\widetilde{\eta}\rangle$ is small enough (the minimum of this expression as a function of $|\widetilde{\eta}\rangle$ is attained at an eigenvector).

In the space of the counter we choose the vector

$$|\psi\rangle = \frac{1}{\sqrt{L+1}} \sum_{j=0}^{L} |j\rangle. \tag{14.10}$$

We set $|\widetilde{\eta}\rangle = |\xi, 0\rangle \otimes |\psi\rangle$ and estimate $\langle\widetilde{\eta}|H|\widetilde{\eta}\rangle$.

It is clear that $E|\psi\rangle = 0$. Therefore

$$\langle\widetilde{\eta}|\, \widetilde{H}_{\text{prop}} \,|\widetilde{\eta}\rangle = 0 = \langle\widetilde{\eta}|\, \widetilde{H}_j \,|\widetilde{\eta}\rangle.$$

Since all auxiliary qubits are initially set to 0, we immediately obtain from the defining formula (14.4) that

$$\langle\widetilde{\eta}|\widetilde{H}_{\text{in}}|\widetilde{\eta}\rangle = 0.$$

It remains to estimate the last term

$$\langle\widetilde{\eta}|\,\widetilde{H}_{\text{out}}\,|\widetilde{\eta}\rangle = \langle\widetilde{\eta}|\Big(U^{\dagger}\Pi_1^{(0)}U\otimes|L\rangle\langle L|\Big)|\widetilde{\eta}\rangle = \mathbf{P}(0)\,\frac{1}{L+1} \le \frac{\varepsilon}{L+1}.$$

Thus we have proved that

$$\langle\widetilde{\eta}|\,\widetilde{H}\,|\widetilde{\eta}\rangle \le \frac{\varepsilon}{L+1},$$

so that H itself has an eigenvalue with the very same upper bound.

14.4.4. ***Lower bound for the eigenvalues if the answer is "no".*** Suppose that for any vector $|\xi\rangle$ the probability of the outcome 1 does not exceed ε, i.e.,

$$\langle\xi,0|U^{\dagger}\Pi_1^{(0)}U|\xi,0\rangle \ge 1-\varepsilon.$$

We will prove that, in this case, all eigenvalues of H are $\ge c\,(1-\sqrt{\varepsilon})\,L^{-3}$, where c is some constant.

The proof is rather long, so we will outline it first. We represent the Hamiltonian in the form $\widetilde{H} = A_1 + A_2$, where $A_1 = \widetilde{H}_{\text{in}} + \widetilde{H}_{\text{out}}$, and $A_2 = \widetilde{H}_{\text{prop}}$. Both terms are nonnegative. It is easy to show that the null subspaces of A_1 and A_2 have trivial intersection (i.e., $\mathcal{L}_1\cup\mathcal{L}_2 = \{0\}$), hence the operator $A_1 + A_2$ is positive definite. But this is not enough for our purpose, so we obtain lower bounds for nonzero eigenvalues of A_1 and A_2, namely, 1 for A_1, and $c'L^{-2}$ for A_2. In order to estimate the smallest eigenvalue of A_1+A_2, we also need to know the angle between the null subspaces. The angle $\vartheta(\mathcal{L}_1,\mathcal{L}_2)$ between subspaces $\mathcal{L}_1$ and $\mathcal{L}_2$ with trivial intersection is given by

$$\cos\vartheta(\mathcal{L}_1,\mathcal{L}_2) = \max_{\substack{|\eta_1\rangle\in\mathcal{L}_1\\ |\eta_2\rangle\in\mathcal{L}_2}} \big|\langle\eta_1|\eta_2\rangle\big|, \qquad 0<\vartheta(\mathcal{L}_1,\mathcal{L}_2)<\frac{\pi}{2}. \tag{14.11}$$

Lemma 14.4. *Let A_1, A_2 be nonnegative operators, and $\mathcal{L}_1$, $\mathcal{L}_2$ their null subspaces, where $\mathcal{L}_1\cap\mathcal{L}_2 = \{0\}$. Suppose further that no nonzero eigenvalue of A_1 or A_2 is smaller than v. Then*

$$A_1 + A_2 \ge v\cdot 2\sin^2\frac{\vartheta}{2},$$

where $\vartheta = \vartheta\,(\mathcal{L}_1,\mathcal{L}_2)$ is the angle between $\mathcal{L}_1$ and $\mathcal{L}_2$.

The notation $A \ge a$ (A an operator, a a number) must be understood as an abbreviation for $A - aI \ge 0$. In other words, if $A \ge a$, then all eigenvalues of A are greater than, or equal to, a.

In our case we will get the estimates 1 and $c'L^{-2}$ for the nonzero eigenvalues of A_1 and A_2 (as already mentioned), and $\sin^2\vartheta \geq (1-\sqrt{\varepsilon})/(L+1)$ for the angle. From this we derive the desired inequality

$$H \geq c\left(1-\sqrt{\varepsilon}\right)L^{-3}.$$

Proof of Lemma 14.4. It is obvious that $A_1 \geq v(I-\Pi_{\mathcal{L}_1})$ and $A_2 \geq v(I-\Pi_{\mathcal{L}_2})$, so it is sufficient to prove the inequality $(I-\Pi_{\mathcal{L}_1})+(I-\Pi_{\mathcal{L}_2}) \geq 2\sin^2(\vartheta/2)$. This, in turn, is equivalent to

$$\Pi_{\mathcal{L}_1}+\Pi_{\mathcal{L}_2} \leq 1+\cos\vartheta. \tag{14.12}$$

Let $|\xi\rangle$ be an eigenvector of the operator $\Pi_{\mathcal{L}_1}+\Pi_{\mathcal{L}_2}$ corresponding to an eigenvalue $\lambda > 0$. Then

$$\Pi_{\mathcal{L}_1}|\xi\rangle = u_1|\eta_1\rangle, \quad \Pi_{\mathcal{L}_2}|\xi\rangle = u_2|\eta_2\rangle, \qquad u_1|\eta_1\rangle + u_2|\eta_2\rangle = \lambda|\xi\rangle,$$

where $|\eta_1\rangle \in \mathcal{L}_1$ and $|\eta_2\rangle \in \mathcal{L}_2$ are unit vectors, and u_1, u_2 are nonnegative real numbers. From this we find

$$\lambda = \langle\xi|\big(\Pi_{\mathcal{L}_1}+\Pi_{\mathcal{L}_2}\big)|\xi\rangle = u_1^2+u_2^2,$$
$$\lambda^2 = \big(u_1\langle\eta_1|+u_2\langle\eta_2|\big)\big(u_1|\eta_1\rangle+u_2|\eta_2\rangle\big) = u_1^2+u_2^2+2u_1u_2\mathrm{Re}\langle\eta_1|\eta_2\rangle.$$

Consequently,

$$(1+x)\lambda-\lambda^2 = x(u_1\pm u_2)^2 \geq 0, \quad \text{where } x = \big|\mathrm{Re}\langle\eta_1|\eta_2\rangle\big|.$$

Thus $\lambda \leq 1+x \leq 1+\cos\vartheta$. □

We will now obtain the above-mentioned estimates. The subspaces A_1 and A_2 can be represented in the form

$$\begin{aligned}\mathcal{L}_1 = \Big(\mathcal{B}^{\otimes m}\otimes|0^{N-m}\rangle\otimes|0\rangle\Big) \oplus \Big(\mathcal{B}^{\otimes N}\otimes\mathbb{C}\big(|1\rangle,\dots,|L-1\rangle\big)\Big)\\ \oplus \Big(U^\dagger\big(|1\rangle\otimes\mathcal{B}^{\otimes(N-1)}\big)\otimes|L\rangle\Big)\end{aligned} \tag{14.13}$$

(the last factor in all three terms pertains to the counter), and

$$\mathcal{L}_2 = B^{\otimes N}\otimes|\psi\rangle, \tag{14.14}$$

where the vector $|\psi\rangle$ was defined by formula (14.10).

For the estimate

$$A_1\big|_{\mathcal{L}_1^\perp} \geq 1 \tag{14.15}$$

it suffices to note that A_1 is the sum of commuting projections, so that all eigenvalues of this operator are nonnegative integers.

For the estimate of $A_2\big|_{\mathcal{L}_2^\perp}$ we need to find the smallest positive eigenvalue of the matrix E. The eigenvectors and eigenvalues of E are

$$|\psi_k\rangle = \alpha_k\sum_{j=0}^{L}\cos\Big(q_k\big(j+\frac{1}{2}\big)\Big)\,|j\rangle, \qquad \lambda_k = 1-\cos q_k,$$

where $q_k = \pi k/(L+1)$ $(k = 0, \dots, L)$. From this it follows that

$$A_2|_{\mathcal{L}_2^\perp} \geq 1 - \cos\left(\frac{\pi}{L+1}\right) \geq c' L^{-2}. \tag{14.16}$$

Finally, we need to estimate the angle between the subspaces $\mathcal{L}_1$ and $\mathcal{L}_2$. We will estimate the square of the cosine of this angle,

$$\cos^2\vartheta = \max_{\substack{|\eta_1\rangle\in\mathcal{L}_1\\ |\eta_2\rangle\in\mathcal{L}_2}} \big|\langle\eta_1|\eta_2\rangle\big|^2 = \max_{|\eta_2\rangle\in\mathcal{L}_2} \langle\eta_2|\Pi_{\mathcal{L}_1}|\eta_2\rangle. \tag{14.17}$$

Since the vector $|\eta_2\rangle$ belongs to $\mathcal{L}_2$, it can be represented in the form $|\eta_2\rangle = |\xi\rangle \otimes |\psi\rangle$ (cf. (14.14)). According to formula (14.13), the projection onto $\mathcal{L}_1$ breaks into the sum of three projections. It is easy to calculate the contribution of the second term; it equals $(L-1)/(L+1)$. The first and third terms add up to

$$\frac{1}{L+1}\big\langle\xi\big|\big(\Pi_{\mathcal{K}_1} + \Pi_{\mathcal{K}_2}\big)\big|\xi\big\rangle \leq \frac{1+\cos\varphi}{L+1},$$

where $\mathcal{K}_1 = \mathcal{B}^{\otimes N}$, $\mathcal{K}_2 = U^\dagger\big(|1\rangle \otimes \mathcal{B}^{\otimes(N-1)}\big)$ and φ is the angle between these two subspaces. (Here we have used inequality (14.12), obtained in the course of the proof of Lemma 14.4.)

The quantity $\cos^2\varphi$ equals the maximum probability for the initial circuit to produce the outcome 1. By hypothesis this probability is not greater than ε. So we can continue the estimate (14.17):

$$\langle\eta_2|\Pi_{\mathcal{L}_1}|\eta_2\rangle \leq \frac{L-1}{L+1} + \frac{1+\sqrt{\varepsilon}}{L+1} = 1 - \frac{1-\sqrt{\varepsilon}}{L+1}.$$

Consequently, $\sin^2\vartheta = 1 - \cos^2\vartheta \geq (1-\sqrt{\varepsilon})/(L+1)$ as asserted above.

14.4.5. *Realization of the counter.* We wrote a nice Hamiltonian almost satisfying the required properties. It has but one shortcoming — the counter is not a qubit. We could, of course, represent it by $O(\log L)$ qubits, but then the Hamiltonian would be only $O(\log L)$-local, not $O(1)$-local.

This shortcoming can be removed if we embed the counter space in a larger space. We take L qubits, enumerated from 1 to L. The suitable embedding $\mathbb{C}^{L+1} \to \mathcal{B}^{\otimes L}$ is

$$|j\rangle \mapsto |\underbrace{1,\dots,1}_{j},\underbrace{0,\dots,0}_{L-j}\rangle.$$

The operators on the space $\mathbb{C}^{L+1}$ used in the construction of the Hamiltonian H are replaced in accordance with the following scheme:

$$\begin{aligned} &|0\rangle\langle 0| \text{ on } \Pi_1^{(0)}, && |0\rangle\langle 1| \text{ on } \big(|0\rangle\langle 1|\big)_1 \Pi_2^{(0)}, \\ &|j\rangle\langle j| \text{ on } \Pi_j^{(1)} \Pi_{j+1}^{(0)}, && |j-1\rangle\langle j| \text{ on } \Pi_{j-1}^{(1)} \big(|0\rangle\langle 1|\big)_j \Pi_{j+1}^{(0)}, \\ &|L\rangle\langle L| \text{ on } \Pi_L^{(1)}, && |L-1\rangle\langle L| \text{ on } \Pi_{L-1}^{(1)} \big(|0\rangle\langle 1|\big)_L. \end{aligned} \tag{13.23}$$

Now they are 3-local (and the Hamiltonian itself, acting also on the qubits of the initial circuit, is 5-local).

To be more precise, we have replaced the Hamiltonian H, acting on the space $\mathcal{L} = \mathcal{B}^{\otimes N} \otimes \mathbb{C}^{L+1}$, by a new Hamiltonian H_{ext}, defined on the larger space $\mathcal{L}_{\text{ext}} = \mathcal{B}^{\otimes N} \otimes \mathcal{B}^{\otimes L}$. The operator H_{ext} maps the subspace $\mathcal{L} \subseteq \mathcal{L}_{\text{ext}}$ into itself and acts on it just as H does.

Now a new problem arises: what to do with the extra states in the extended space of the counter? We will cope with this problem by adding still another term to the Hamiltonian H_{ext}:

$$H_{\text{stab}} = I_{\mathcal{B}^{\otimes N}} \otimes \sum_{j=1}^{L-1} \Pi_j^{(0)} \Pi_{j+1}^{(1)}.$$

The null subspace of the operator H_{stab} coincides with the old working space $\mathcal{L}$, so that the supplementary term does not change the upper bound for the minimum eigenvalue for the answer "yes".

For the answer "no" the required lower bound for the eigenvalues of the operator $H_{\text{ext}} + H_{\text{stab}}$ can be recovered in the following way. Both terms leave the subspace $\mathcal{L}$ invariant, so that we can also examine the action of $H_{\text{ext}} + H_{\text{stab}}$ on $\mathcal{L}$ and on its orthogonal complement $\mathcal{L}^{\perp}$ independently. On $\mathcal{L}$ we have $H_{\text{ext}} \geq c(1 - \sqrt{\varepsilon})L^{-3}$ and $H_{\text{stab}} = 0$, and on $\mathcal{L}^{\perp}$ we have $H_{\text{ext}} \geq 0$ and $H_{\text{stab}} \geq 1$. (Here we use the fact that each of the terms of the Hamiltonian, (14.4), (14.5) and (14.6), remains nonnegative for the change (13.23).) In both cases

$$H_{\text{ext}} + H_{\text{stab}} \geq c(1 - \sqrt{\varepsilon})L^{-3}.$$

This completes the proof of Theorem 14.3.

14.5. The place of BQNP among other complexity classes. It follows directly from the definition that the class BQNP contains the class MA (and so also BPP and NP). Nothing more definitive can be said at present about the strength of "nondeterministic quantum algorithms".[12]

Nor can much more be said about its "weakness".

[12] Caveat: in the literature there is also a different definition of "quantum NP", for which a complete characterization in terms of classical complexity classes can be obtained (cf. [**2, 73**]).

Proposition 14.5. BQNP $\subseteq$ PSPACE.

Proof. The maximum probability that Merlin's message will be accepted by Arthur is equal to the maximum eigenvalue of the operator $X = X^{(1)}$ (cf. formula (14.3)). We will need to compute this quantity with precision $O(n^{-\alpha})$, $\alpha > 0$.

We note that $0 \le X \le 1$. For the estimate of the maximum eigenvalue we will use the following asymptotic equality:

$$\ln \lambda_{\max} = \lim_{d\to\infty} \frac{\ln \operatorname{Tr} X^d}{d}.$$

Let $\lambda_{\max} = \lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_{2^m}$ be the eigenvalues of the operator X (here $m = \operatorname{poly}(n)$ is the length of the message). We have the inequality

$$\ln \lambda_{\max} \le \frac{\ln \operatorname{Tr} X^d}{d} = \frac{\ln \sum_{j=1}^{2^m} \lambda_j^d}{d} \le \ln \lambda_{\max} + \frac{m}{d} \ln 2.$$

Therefore, in order to estimate $\lambda_{\max}$ with polynomial precision, it suffices to compute the trace of the d-th power of X, with d polynomial in m.

The computation of the quantity $\operatorname{Tr} X^d$ is achieved with polynomial memory by the same means that was used to simulate a quantum circuit. □

Remark 14.2. The result obtained can be strengthened: BQNP $\subseteq$ PP. The proof is completely analogous to the solution of Problem 9.5.

Remark 14.3. We have limited ourselves to the case of Arthur-Merlin games in which only one message is communicated. In general, Arthur and Merlin can play several rounds, sending messages back and forth. Recently it has been shown [**72, 38**] that such a quantum game with three messages (i.e., 1.5 rounds) has the same complexity as the game with polynomially many rounds. The corresponding complexity class is called QIP; it contains PSPACE. This contrasts with the properties of classical Arthur-Merlin games. In the classical case, the game with polynomially many rounds yields PSPACE [**58, 59**]. But in wide circles of narrow specialists the opinion prevails that no fixed number of rounds would suffice.[13]

15. Classical and quantum codes

In this section we explain the concept of error-correcting code, in its classical and quantum formulations. Our exposition does not go beyond definitions and basic examples; we do not address the problem of finding codes with optimal parameters. The interested reader is referred to [**47**] (classical codes) and [**17**] (quantum codes).

[13] The game with an arbitrary constant number of rounds corresponds to a complexity class AM $\subseteq \Pi_2$ [**6**]. It is widely believed that "the polynomial hierarchy does not collapse", i.e., co-NP $= \Pi_1 \subset \Pi_2 \subset \Pi_3 \subset \cdots \subset$ PSPACE (the inclusions are strict).

First, a bit of motivation. As discussed earlier, quantum computation is "not too" sensitive to errors in the realization of unitary operators: errors accumulate linearly (see, e.g., the result of Problem 9.2). Therefore, a physical implementation of elementary gates with precision δ will allow one to use circuits of size $L \sim \delta^{-1}$. But this is not enough to make quantum computation practical. Therefore a question arises: is it possible to avoid accumulation of errors by using circuits of some special type? More specifically, is it possible to replace an arbitrary quantum circuit by another circuit that would realize the same unitary operator (or compute the same Boolean function), but in an error-resistant fashion?

The answer to this question is affirmative. In fact, the new circuit will resist not only inaccurate realization of unitary gates but also some interaction with the environment and stochastic errors (provided that they occur with small probability). The rough idea is to *encode* (replace) each qubit used in the computation (*logical qubit*) by several *physical qubits*; see Remark 8.3 on page 74. The essential fact is that errors usually affect only few qubits at a time, so that encoding increases the stability of a quantum state.

Organization of computation in a way that prevents accumulation of errors is called *fault-tolerant* computation. In the classical case, fault-tolerance can be achieved by the use of the *repetition code*: 0 is encoded by $(0, \dots, 0)$ (n times), and 1 is encoded by $(1, \dots, 1)$. Such a simple code does not work in the quantum case, but more complicated codes do. The first method of fault-tolerant quantum computation was invented by P. Shor [**65**] and improved independently by several authors [**42, 36**]. Alternative approaches were suggested by D. Aharonov and M. Ben-Or [**3**] and A. Kitaev [**35**].

Fault-tolerance is a rather difficult subject, but our goal here is more modest. Suppose we have a quantum state of n qubits that is subjected to an error. Under what condition is it possible to recover the original state, assuming that *the execution of the recovery procedure is error-free*? (The fault-tolerant computation deals with the more realistic situation where errors occur constantly, though at a small rate.) Of course, error recovery is not possible for a general state $|\xi\rangle \in \mathcal{B}^{\otimes n}$. However, it can be possible for states $|\xi\rangle \in \mathcal{M}$, where $\mathcal{M} \subseteq \mathcal{B}^{\otimes n}$ is a suitable fixed subspace. Likewise, in the classical case we should consider states that belong to a fixed subset $M \subseteq \mathbb{B}^n$.

Definition 15.1. A classical code of type (n, m) is a subset $M \subseteq \mathbb{B}^n$ which consists of 2^m elements (where m — the number of encoded bits — is not necessarily integral). Elements of M are called *codewords*.

A quantum code of type (n, m) is a subspace $\mathcal{M} \subseteq \mathcal{B}^{\otimes n}$ of dimension 2^m. Elements of $\mathcal{M}$ are called *codevectors*.

Remark 15.1. In the theory of fault-tolerant quantum computation, a slightly different kind of codes is used. Firstly, an encoding must be specified, i.e., the subspace $\mathcal{M}$ must be identified with a fixed space $\mathcal{L}$; usually, $\mathcal{L} = \mathcal{B}$. In other words, an *encoding* is an isometric embedding $V : \mathcal{L} \to \mathcal{B}^{\otimes n}$ such that $\mathcal{M} = \operatorname{Im} V$. Secondly, sometimes one needs to consider one-to-many encodings (because errors happen and get corrected constantly, so at any moment there are some errors that have not been corrected yet). A *one-to-many encoding* is an isometric embedding $V : \mathcal{L} \otimes \mathcal{F} \to \mathcal{B}^{\otimes n}$, where $\mathcal{F}$ is some auxiliary space.

Besides the code, we need to define an *error model*. It is also called *communication channel*: one may think that errors occur when a state (classical or quantum) is transferred from one location to another. Intuitively, this should be something like a multivalued map $\mathbb{B}^n \to \mathbb{B}^{n'}$ or $\mathcal{B}^{\otimes n} \to \mathcal{B}^{\otimes n'}$ (where n' is the number of bits at the output of the channel; usually, $n' = n$). We begin with the classical case and then construct the quantum definition by analogy.

15.1. Classical codes. There are two models of errors: a more realistic probabilistic model and a simplified set-theoretic version. According to the probabilistic model, a communication channel is given by a set of conditional probabilities $p(y|x)$ for receiving the word y upon transmission of the word x. We will consider the case of independently distributed errors, where $n' = n$, and the conditional probabilities are determined by the probability p_1 of an error (bit flip) in the transmission of a single bit:

$$p(y|x) = p_1^{d(x,y)}(1-p_1)^{n-d(x,y)}. \tag{15.1}$$

Here $d(x,y)$ is the *Hamming distance* — the number of distinct bits.

There is a standard method for simplifying a probabilistic error model by classifying errors as "likely" and "unlikely". Let us estimate the probability that in the model defined above, more than k bit flips occur (as is clear from formula (15.1), this probability does not depend on x). Suppose that n and k are fixed, whereas $p_1 \to 0$. Then

$$\mathbf{Pr}\big[\text{number of bit flips} > k\big] = \sum_{j>k} \binom{n}{j} p_1^j (1-p_1)^{n-j} = O(p_1^{k+1}). \tag{15.2}$$

Thus the probability that more than k bit flip is small. So we say that this event is unlikely; we can greatly simplify the model by assuming that such an event never happens. We will suppose that, upon transmission of the word x, some word y is received such that $d(x,y) \le k$. This simplified model only defines a set of possible (or "likely") outcomes but says nothing about their probabilities.

We introduce the notation:

$N = \mathbb{B}^n$ — set of inputs,
$N' = \mathbb{B}^{n'}$ — set of outputs,
$E \subseteq N \times N'$ — set of transitions (i.e., set of errors),
$E(n,k) = \{(x,y) : d(x,y) \le k\}$.

Definition 15.2. A code M *corrects errors* from a set $E \subseteq N \times N'$ if for any $x_1, x_2 \in M$, $(x_1, y_1) \in E$, $(x_2, y_2) \in E$, the condition $x_1 \neq x_2$ implies that $y_1 \neq y_2$.

In the particular case $E = E(n,k)$, we say that the *code corrects k errors*.

Remark 15.2. The term "error correcting code" is imprecise. It would be more accurate to say that the code offers the possibility for correcting errors. An *error-correcting transformation* is a map $P : N' \to N$ such that, if $(x,y) \in E$ and $x \in M$, then $P(y) = x$.

Example 15.1. The repetition code of type $(3,1)$:

$$M_3 = \{(0,0,0),(1,1,1)\} \subseteq \mathbb{B}^3.$$

Such a code will correct a single error.

An obvious generalization of this example leads to classical codes M_n of type $(n,1)$ which correct $k = \lfloor (n-1)/2 \rfloor$ errors (see below). We will also construct more interesting examples of classical codes. To start, we give yet another standard definition.

Definition 15.3. The *code distance* is

$$d(M) = \min\{d(x_1,x_2) : x_1, x_2 \in M,\ x_1 \neq x_2\}.$$

For the code M_3 of Example 15.1 the code distance is 3.

Proposition 15.1. *A code M corrects k errors if and only if $d(M) > 2k$.*

Proof. According to Definition 15.2, the code does not correct k errors if and only if there exist $x_1, x_2 \in M$ $(x_1 \neq x_2)$ and $y \in \mathbb{B}^n$ such that $d(x_1,y) \le k$ and $d(x_1,y) \le k$. For fixed x_1, x_2, such a y exists if and only if $d(x_1,x_2) \le 2k$. □

15.2. Examples of classical codes.

1. The *repetition code* M_n of type $(n,1)$ and distance n:

$$M_n = \{(\underbrace{0,\dots,0}_{n}), (\underbrace{1,\dots,1}_{n})\}.$$

This code can be used with the obvious encoding: we repeat a single bit n times. To restore the codeword after an error, we replace the value of

the bits with the value that occurs most frequently. This series of codes, as will be shown later, does not generalize to the quantum case.

2. *Parity check:* this is a code of type $(n, n-1)$ and distance 2. It consists of all *even* words, i.e., of words containing an even number of 1s.
3. The *Hamming code* H_r. This code is of type $(n, n-r)$, where $n = 2^r - 1$. It is defined as follows.

   Elements of $\mathbb{B}^n$ are sequences of bits $x = (x_\alpha : \alpha = 1, \dots, n)$. In turn, the index of each bit can be represented in binary as $\alpha = (\alpha_1, \dots, \alpha_r)$. We introduce a set of check sums $\mu_j : \mathbb{B}^n \to \mathbb{B}$ $(j = 1, \dots, r)$ and define the Hamming code by the condition that all the check sums are equal to 0:

$$\mu_j(x) = \sum_{\alpha:\, \alpha_j = 1} x_\alpha \mod 2,$$
$$H_r = \bigl\{ x \in \mathbb{B}^{2^r} : \mu_1(x) = \dots = \mu_r(x) = 0 \bigr\}.$$

   For example, the Hamming code H_3 is defined by the system of equations

$$\begin{aligned} x_{100} + x_{101} + x_{110} + x_{111} &= \mu_1(x) = 0, \\ x_{010} + x_{011} + x_{110} + x_{111} &= \mu_2(x) = 0, \\ x_{001} + x_{011} + x_{101} + x_{111} &= \mu_3(x) = 0. \end{aligned}$$

   where (mod 2) arithmetic is assumed.

   We will see that the Hamming code has distance $d(H_r) = 3$ for any $r \geq 2$.

15.3. Linear codes. The set $N = \mathbb{B}^n$ can be regarded as the n-dimensional linear space over the two-element field $\mathbb{F}_2$. A *linear code* is a linear subspace $M \subseteq \mathbb{F}_2^n$. All the examples given above are of this kind. A linear code of type (n, m) can be defined by a dual basis, i.e., a set of $n - m$ linearly independent linear forms (called *check sums*) which vanish on M. The coefficients of the check sums constitute rows of the *check matrix*. For example, the check matrix of the Hamming code H_3 is

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Proposition 15.2. *The code distance of a linear code equals the minimum number of distinct columns of the check matrix that are linearly dependent.*

Proof. A linear dependency between the columns of the check matrix is a nonzero codeword. If a subset $S \subseteq \{1, \dots, n\}$ of columns is dependent, the corresponding word $x \in M$ has nonzero symbols only at positions $\alpha \in S$ (and vice versa). Thus, if k columns are dependent, then there is $x \in M$, $x \neq 0$, such that $d(x, 0) \leq k$. Therefore $d(M) \leq k$. Conversely, if $d(x_1, x_2) \leq k$ for

some $x_1, x_2 \in M$ $(x_1 \neq x_2)$, then $x = x_2 - x_1 \in M$, $x \neq 0$, $d(x,0) \leq k$, hence k (or fewer) columns are linearly dependent. □

The columns of the check matrix of the Hamming code H_r correspond to the nonzero elements of $\mathbb{F}_2^r$. Any two columns are different, hence they are linearly independent. On the other hand, the sum of the first three columns is 0 (for $r \geq 2$). Therefore the code distance is 3.

15.4. Error models for quantum codes. Firstly, we will define a quantum analogue of the transition set $E \subseteq N \times N'$. This is an arbitrary linear subspace $\mathcal{E} \subseteq \mathbf{L}(\mathcal{N}, \mathcal{N}')$, called an *error space*. There is also an analogue of the set $E(n,k)$. Let us assume that $\mathcal{N} = \mathcal{N}' = \mathcal{B}^{\otimes n}$. For each subset of qubits $A \subseteq \{1, \ldots, n\}$, let $\mathcal{E}[A]$ be the set of linear operators that act only on those qubits and do not affect the remaining qubits (such an operator has the form $X[A]$). Then we define the space

$$\mathcal{E}(n,k) = \sum_{A:\,|A|\leq k} \mathcal{E}[A],$$

where we take the sum of linear subspaces: $\sum_j \mathcal{L}_j = \Big\{\sum_j X_j :\ X_j \in \mathcal{L}_j\Big\}$. In the sequel we will be interested in the possibility of correcting errors from the space $\mathcal{E}(n,k)$.

Next, we will introduce a physical model of quantum errors, which has no classical analogue. Consider a system of n qubits which interact with the environment (characterized by a Hilbert space $\mathcal{F}$). We assume that the interaction is described by the Hamiltonian

$$H = H_0 + V, \qquad H_0 = I_{\mathcal{B}^{\otimes n}} \otimes Z, \qquad V = \sum_{j=1}^{n} \sum_{\alpha\in\{x,y,z\}} \sigma_j^\alpha \otimes B_{j\alpha}. \tag{15.3}$$

Here $\sigma_j^\alpha = \sigma^\alpha[j]$ denotes the application of the Pauli matrix σ^α (see (8.4)) to the j-th qubit; $Z, B_{j\alpha} \in \mathbf{L}(\mathcal{F})$ are Hermitian operators acting on the environment: $B_{j\alpha}$ describes interaction of the environment with the j-th qubit, whereas Z is the Hamiltonian of the environment itself. It is an important assumption that qubits interact with the environment by small groups. The Hamiltonian (15.3) contains only one-qubit terms ($\sigma_j^\alpha \otimes B_{j\alpha}$), but we could also include two-qubit ($\sigma_j^\alpha \sigma_l^\beta \otimes B^{(2)}_{j\alpha l\beta}$) and higher-order terms, up to some constant order.

If the interaction lasts for some time τ, it results in the evolution of the quantum state by the unitary operator

$$\begin{aligned} U &= \exp(-i\tau H) = e^{-i\tau(H_0+V)} = \lim_{N\to\infty} \Big(e^{-i\frac{\tau}{N}H_0}\Big(1 - i\frac{\tau}{N}V\Big)\Big)^N \\ &= \lim_{N\to\infty} e^{-i\tau H_0}\Big(1 - i\frac{\tau}{N}V\big(\tfrac{N-1}{N}\tau\big)\Big)\cdots\Big(1 - i\frac{\tau}{N}V\big(\tfrac{1}{N}\tau\big)\Big)\Big(1 - i\frac{\tau}{N}V(0)\Big), \end{aligned}$$

where $V(t) = e^{itH_0} V e^{-itH_0}$. We can expand this expression in powers of V, replacing sums by integrals in the $N \to \infty$ limit. Thus we obtain the following result:

$$U = \exp(-i\tau H) = \sum_{k=0}^{\infty} X_k, \qquad X_k \in \mathcal{E}(n,k) \otimes \mathbf{L}(\mathcal{F}),$$
$$X_k = e^{-i\tau H_0} \left((-i)^k \int\limits_{0<t_1<\cdots<t_k<\tau} \cdots \int V(t_k) \cdots V(t_1)\, dt_1 \cdots dt_k \right), \tag{15.4}$$

where

$$V(t) = e^{itH_0} V e^{-itH_0} = \sum_{j,\alpha} \sigma_j^\alpha \otimes B_{j\alpha}(t), \qquad B_{j\alpha}(t) = e^{itZ} B_{j\alpha} e^{-itZ}.$$

Suppose that the interaction of the qubits with the environment is small,

$$\|B_{j\alpha}\| \le \frac{\delta}{3\tau}. \tag{15.5}$$

Then we can obtain an upper bound for the norm of each term X_k in (15.4), $\|X_k\| \le \frac{n^k}{k!}\delta^k$. Therefore U is approximated by an operator $U^{(k)}$ such that

$$\|U - U^{(k)}\| \le O(\delta^{k+1}), \qquad U^{(k)} \in \mathcal{E}(n,k) \otimes \mathbf{L}(\mathcal{F}). \tag{15.6}$$

Namely, $U^{(k)} = \sum_{l=0}^{k} X_l$ (note that $U^{(k)}$ is not unitary). If errors from the space $\mathcal{E}(n,k)$ are recoverable (assuming that the initial state belongs to $\mathcal{M} \otimes \mathcal{F}$, where $\mathcal{M}$ is a suitable code), then the error-correcting procedure will cancel the effect of U with precision $O(\delta^{k+1})$.

Finally, we will define a quantum version of the model of independent errors. Let us assume that the quantum state of n qubits undergoes the transformation that is described by the physically realizable superoperator $T = T_1^{\otimes n}$, where $\|T_1 - I\|_\diamondsuit \le \delta$ (see Section 11.5 for the definition of the superoperator norm $\|\cdot\|_\diamondsuit$). Let $T_1 - I = R$; then $T = (I+R)^{\otimes n}$. This is essentially a special case of the model described by formulas (15.3)–(15.4): each qubit interacts with its own piece of environment, which is initially not entangled with the rest of the system and is discarded after the action of the operator U. However, the condition $\|T_1 - I\|_\diamondsuit \le \delta$ is different from condition (15.5), so we need to consider this model separately.

One can obtain an estimate that is analogous to (15.2) or (15.6). Let us write the decomposition

$$T = (I+R)^{\otimes n} = \underbrace{\sum_{A:\,|A|\le k} R^{\otimes A} \otimes I^{\otimes(\{1,\ldots,n\}\setminus A)}}_{T^{(k)}} + \underbrace{\sum_{A:\,|A|>k} R^{\otimes A} \otimes I^{\otimes(\{1,\ldots,n\}\setminus A)}}_{P}.$$

The first term $T^{(k)}$ can be represented as $\sum_p X_p \cdot Y_p^\dagger$, where $X_p, Y_p \in \mathcal{E}(n,k)$. So we may write symbolically $T^{(k)} \in \mathcal{E}(n,k) \cdot \mathcal{E}(n,k)^\dagger$. Using the properties of the superoperator norm, we estimate the norm of the remaining term,

$$\|P\|_\diamond \le \sum_{j>k} \binom{n}{j} \|R\|_\diamond^j = O(\delta^{k+1}).$$

The model of independent errors includes two extreme cases. If $T = U \cdot U^\dagger$ (where U is unitary, $\|U - I\| \le \delta/2$), the errors are called *coherent*. In the case where

$$T = (1-p) I \cdot I + \sum_j p_j U_j \cdot U_j^\dagger, \qquad U_j^\dagger U_j = I, \qquad p = \sum_j p_j \le \delta/2,$$

the errors are called *stochastic*, indicating that they can be described in terms of probability rather than operator or superoperator norms.

15.5. Definition of quantum error correction. Following the classical analogy, we would like to say that quantum errors are recoverable if they take distinct codevectors to distinct codevectors. However, the general philosophy of quantum mechanics suggests that we replace "distinct" by "orthogonal".

Definition 15.4. A quantum code (a subspace $\mathcal{M} \subseteq \mathcal{N}$) *corrects errors from* $\mathcal{E} \subseteq \mathbf{L}(\mathcal{N}, \mathcal{N}')$ if

$$\forall |\xi_1\rangle, |\xi_2\rangle \in \mathcal{M}\ \forall X, Y \in \mathcal{E}\ \big(\langle \xi_2 | \xi_1 \rangle = 0\big) \Rightarrow \big(\langle \xi_2 | Y^\dagger X | \xi_1 \rangle = 0\big). \tag{15.7}$$

In the case where $\mathcal{E} = \mathcal{E}(n,k)$, one says that the code *corrects* k *errors*.

Definition 15.5. Let $\mathcal{M} \subseteq \mathcal{N}$ and $\mathcal{E} \subseteq \mathbf{L}(\mathcal{N}, \mathcal{N}')$. A physically realizable superoperator $P : \mathbf{L}(\mathcal{N}') \to \mathbf{L}(\mathcal{M})$ is called an *error-correcting transformation* for the code $\mathcal{M}$ and the error space $\mathcal{E}$ if

$$\forall T \in \mathcal{E} \cdot \mathcal{E}^\dagger\ \exists c = c(T)\ \forall \rho \in \mathbf{L}(\mathcal{M})\ PT\rho = c(T)\rho.$$

Note that if T is trace-preserving, then $c(T) = 1$.

Theorem 15.3. *If the code* $\mathcal{M}$ *corrects errors from* $\mathcal{E}$*, then an error-correcting transformation exists.*

A proof will be given below. The converse assertion is proved in [**36**].

Example 15.2. *Trivial code* of type (n,m): let $\mathcal{M} = \mathcal{B}^{\otimes m} \otimes |0^{n-m}\rangle$ and $\mathcal{E} = \mathcal{E}[m+1, \dots, n]$, i.e., the first m qubits are used for the coding whereas the errors act on the other qubits. Condition (15.7) is clearly satisfied. For the role of error-correcting transformation we can take $P = I_{\mathbf{L}(\mathcal{B}^{\otimes m})} \otimes R$, where $R : X \mapsto (\operatorname{Tr} X)\, |0^{n-m}\rangle\langle 0^{n-m}|$. The transformation P is realized very simply: we discard the last $n-m$ qubits and replace them by new qubits

in the state $|0\rangle$. There is, of course, little practical use for such a code. It is interesting, however, that any error-correcting quantum code has, in a certain sense, the same structure as the trivial one (cf. Lemma 15.5 below).

Example 15.3. We examine a quantum analog of the repetition code: $\mathcal{M}_n^z = C\big(|0,\dots,0\rangle, |1,\dots,1\rangle\big)$. It comes with the standard encoding $V_n^z : |a\rangle \mapsto |a,\dots,a\rangle$. (The index z in the notation indicates that we copy a qubit relative to the basis which consists of the eigenvectors of σ^z, i.e., the standard basis.) Consider two vectors, $|\xi_1\rangle = |0,\dots,0\rangle + |1,\dots,1\rangle$ and $|\xi_2\rangle = |0,\dots,0\rangle - |1,\dots,1\rangle$, and two errors $X, Y \in \mathcal{E}(n,1)$, namely, $X = I$, $Y = \sigma^z[j]$ (where j is arbitrary). It is obvious that $Y|\xi_2\rangle = X|\xi_1\rangle = |\xi_1\rangle \neq 0$, which contradicts condition (15.7). We see that the repetition code of any size does not protect against a one-qubit error.

Thus the existence of nontrivial quantum codes is far from obvious. See formula (15.11) for the simplest example.

In Definition 15.4 the statement was only about pairs of orthogonal states. However, we can infer a consequence regarding an arbitrary pair of states. Let us fix $X, Y \in \mathcal{E}$ and set $Z = Y^\dagger X$. Then

$$\forall |\xi_1\rangle, |\xi_2\rangle \in \mathcal{M}\ \langle\xi_2|Z|\xi_1\rangle = c(Z)\langle\xi_2|\xi_1\rangle, \tag{15.8}$$

where $c(Z)$ is some complex number, independent of $|\xi_1\rangle, |\xi_2\rangle$. Indeed, let $|\eta_1\rangle, \dots, |\eta_m\rangle$ be an orthonormal basis for the subspace $\mathcal{M}$. By Definition 15.4, $\langle\eta_j|Z|\eta_k\rangle = 0$ when $j \neq k$. It also follows that $\langle\eta_j|Z|\eta_j\rangle$ does not depend on j, since

$$\langle\eta_j|Z|\eta_j\rangle - \langle\eta_k|Z|\eta_k\rangle = \langle\eta_j - \eta_k|Z|\eta_j + \eta_k\rangle + \langle\eta_k|Z|\eta_j\rangle - \langle\eta_j|Z|\eta_k\rangle = 0.$$

(All three terms on the right-hand side of the equality are equal to zero, since the vectors that enter into them are orthogonal.)

Remark 15.3. To understand the meaning of condition (15.8), let us put it into this form:

$$\forall |\xi\rangle \in \mathcal{M}\ \Pi_{\mathcal{M}} Z|\xi\rangle = c(Z)|\xi\rangle. \tag{15.9}$$

We may replace the projector $\Pi_{\mathcal{M}}$ by a measurement that distinguishes "good" states ($|\psi\rangle \in \mathcal{M}$) from "bad" states ($|\psi\rangle \perp \mathcal{M}$). Loosely speaking, formula (15.9) says that if the codevector $|\xi\rangle$ is acted upon by Z and still accepted as "good" (which happens with probability $|c(Z)|^2$), it remains intact. Thus any possible damage to the codevector caused by the error Z is being detected.

We summarize the above discussion as follows.

Definition 15.6. A quantum code $\mathcal{M} \subseteq \mathcal{N}$ *detects* an error[14] $Z \in \mathbf{L}(\mathcal{N})$ if there exists some $c = c(Z) \in \mathbb{C}$ such that

$$\forall |\xi_1\rangle, |\xi_2\rangle \in \mathcal{M}\ \langle \xi_2|Z|\xi_1\rangle = c(Z)\langle \xi_2|\xi_1\rangle.$$

The *code distance* is the smallest number $d = d(\mathcal{M})$ for which the code does not detect errors from the space $\mathcal{E}(n, d)$.

Proposition 15.4. *A code $\mathcal{M} \subseteq \mathcal{N}$ corrects errors from $\mathcal{E} \subseteq \mathbf{L}(\mathcal{N}, \mathcal{N}')$ if and only if it detects errors from the space*

$$\mathcal{E}^\dagger\mathcal{E} = \Big\{\sum_p Y_p^\dagger X_p :\ X_p, Y_p \in \mathcal{E}\Big\}.$$

In particular, a code $\mathcal{M} \subseteq \mathcal{B}^{\otimes n}$ corrects k errors if and only if $d(\mathcal{M}) > 2k$.

The first part of the proposition has been already proved. The second part follows from the fact that $\mathcal{E}(n,k)^\dagger\mathcal{E}(n,k) = \mathcal{E}(n,2k)$.

We now proceed to the proof of Theorem 15.3.

Lemma 15.5. *Let a quantum code $\mathcal{M} \subseteq \mathcal{N}$ correct errors from a subspace $\mathcal{E} \subseteq \mathbf{L}(\mathcal{N}, \mathcal{N}')$. Then there exist a Hilbert space $\mathcal{F}$, an isometric embedding $V : \mathcal{M} \otimes \mathcal{F} \to \mathcal{N}'$ and a linear map $f : \mathcal{E} \to \mathcal{F}$ such that*

$$\forall X \in \mathcal{E}\ \forall |\xi\rangle \in \mathcal{M}\ X|\xi\rangle = V\big(|\xi\rangle \otimes |f(X)\rangle\big). \tag{15.10}$$

Proof. Let $\mathcal{E}_0 = \big\{X \in \mathcal{E} :\ \forall |\xi\rangle \in \mathcal{M}\ X|\xi\rangle = 0\big\}$. Consider the quotient space $\mathcal{F} = \mathcal{E}/\mathcal{E}_0$ together with the natural map $f : \mathcal{E} \to \mathcal{F}$. The very definition of $\mathcal{F}$ and f implies the existence of a linear map $V : \mathcal{M} \otimes \mathcal{F} \to \mathcal{N}'$ that satisfies (15.10). It is only necessary to check that V is an isometry.

An inner product on the space $\mathcal{F}$ can be defined with the aid of the function c from property (15.8) of the code: if $|\eta_1\rangle = |f(X)\rangle$ and $|\eta_2\rangle = |f(Y)\rangle$, then $\langle \eta_2|\eta_1\rangle = c(Y^\dagger X)$. It is clear that this quantity depends only on $|\eta_1\rangle$ and $|\eta_2\rangle$ rather than on the particular choice of X and Y. It is also clear that $\langle \eta|\eta\rangle > 0$ if $|\eta\rangle \neq 0$. Formula (15.8) shows at once that the map V is an isometry. □

Proof of Theorem 15.3. We decompose the space $\mathcal{N}'$ into the sum of two orthogonal subspaces: $\mathcal{N}' = (\operatorname{Im} V) \oplus \mathcal{K}$, where V is the map of the preceding lemma. Let $W : \mathcal{K} \to \mathcal{N}'$ be the inclusion map, and $R : \mathbf{L}(\mathcal{K}) \to \mathbf{L}(\mathcal{M})$ an arbitrary physically realizable superoperator. Then we define

$$P :\ \rho \mapsto \operatorname{Tr}_{\mathcal{F}}(V^\dagger \rho V) + R(W^\dagger \rho W), \qquad c :\ X \cdot Y^\dagger \mapsto \langle f(Y)|f(X)\rangle.$$

The function c extends to the space $\mathcal{E} \cdot \mathcal{E}^\dagger$ by linearity. □

[14] Admittedly, this terminology may be confusing. For example, the identity operator is "detected" by any code. As explained above, it is more adequate to say that the code detects the error-induced damage, if any. But this sounds too awkward.

Lemma 15.5 and the proof of Theorem 15.3 can be explained in the following way. An error-correcting code is characterized by the property that the error does not mix with the encoded state, i.e., it remains in the form of a separate tensor factor $|\eta\rangle = f(X) \in \mathcal{F}$. (Using the terminology of Remark 15.1, we may say that the original state $|\xi\rangle \in \mathcal{M}$ gets encoded with the one-to-many encoding V.) The correcting transformation extracts the "built-in" error $|\eta\rangle$ and deposits it in the trash bin.

[1!] **Problem 15.1.** Let the code $\mathcal{M} \subseteq B^{\otimes n}$ detect errors from $\mathcal{E}(A)$. Prove that the state $\rho \in \mathbf{L}(\mathcal{M})$ can be restored without using qubits from the set A.

15.6. Shor's code. Following Shor [**64**], we define a series of quantum codes with arbitrary large distance. The r-th member of this series encodes one logical qubit into r^2 physical qubits; the distance of this code equals r.

The idea is to fix the repetition code $\mathcal{M}_n^z$ defined above. We have seen that it fails to correct a one-qubit error of the form $\sigma_j^z = \sigma^z[j]$. Operators generated by $\sigma_1^z, \dots, \sigma_n^z$ (i.e., ones that preserve the basis vectors up to a phase), are called *phase errors*. Still, the repetition code protects against *classical errors* — operators of the form σ_j^x and their products (as well as linear combinations of such products). Specifically, the n-qubit repetition code $\mathcal{M}_n^z$ corrects classical errors that affect at most $\lfloor (n-1)/2 \rfloor$ qubits. However, phase errors and classical errors are conjugate to each other, since $\sigma^x = H\sigma^z H$. Therefore the "dual repetition code" $\mathcal{M}_n^x$, defined by the encoding

$$V_n^x : \quad |\eta_a\rangle \mapsto |\eta_a\rangle \otimes \dots \otimes |\eta_a\rangle, \qquad |\eta_a\rangle = H|a\rangle,$$
$$|a\rangle \mapsto 2^{-(n-1)/2} \sum_{\substack{y_1,\dots,y_n \\ y_1+\dots+y_n \equiv a \pmod 2}} |y_1, \dots, y_n\rangle \qquad (a = 0, 1),$$

will protect against phase errors (but not classical errors).

We may try to combine the two codes: first we encode the logical qubit with V_r^x; then we encode each of the resulting qubit with V_r^z. (Such composition of two codes is called a *concatenated code*.) Thus we obtain the encoding $V = (V_r^z)^{\otimes r} V_r^x : \mathcal{B} \to \mathcal{B}^{\otimes r^2}$. Inasmuch as the number of physical qubits is an exact square, it is convenient to write basis vectors of the corresponding space as matrices, e.g., $\left| \begin{smallmatrix} x_{11} & \dots & x_{1r} \\ \dots & \dots & \dots \\ x_{r1} & \dots & x_{rr} \end{smallmatrix} \right\rangle$. In this notation the encoding V assumes the form $|a\rangle \mapsto |\xi_a\rangle$, where

$$|\xi_a\rangle \;=\; 2^{-(r-1)/2} \sum_{\substack{y_1,\dots,y_r \in \mathbb{F}_2 \\ y_1+\dots+y_r = a}} \left| \begin{smallmatrix} y_1 & \dots & \dots & y_1 \\ y_2 & \dots & \dots & y_2 \\ \dots & \dots & \dots & \dots \\ y_r & \dots & \dots & y_r \end{smallmatrix} \right\rangle \qquad (a = 0, 1). \tag{15.11}$$

To analyze the Shor code, we will decompose errors over a basis of operators built on Pauli matrices. More precisely, the basis of the space $\mathbf{L}(\mathcal{B})$ consists of the identity operator I and the three Pauli matrices. We introduce nonstandard notation for these matrices:

$$\sigma_{00} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I, \qquad \sigma_{01} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \sigma^z,$$

$$\sigma_{10} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \sigma^x, \qquad \sigma_{11} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = \sigma^y.$$

These operators are remarkable in that they are unitary and Hermitian at the same time. The indexing we have introduced allows one to conveniently express the multiplication rules and commutation relations between the basis operators:

$$\sigma_{\alpha\beta}\sigma_{\alpha'\beta'} = i^{\widetilde{\omega}(\alpha,\beta;\alpha',\beta')}\,\sigma_{\alpha\oplus\alpha',\beta\oplus\beta'}, \qquad \sigma_{\alpha\beta}\sigma_{\alpha'\beta'} = (-1)^{\alpha\beta'-\alpha'\beta}\sigma_{\alpha'\beta'}\sigma_{\alpha\beta},$$

where $\widetilde{\omega}(\alpha,\beta';\alpha',\beta) \in \mathbb{Z}_4$ (see (15.16) below for an explicit formula). The set of indices forms the Abelian group $G = \mathbb{Z}_2 \oplus \mathbb{Z}_2$, which can also be considered as a 2-dimensional linear space over the field $\mathbb{F}_2$.

The basis for $\mathbf{L}(\mathcal{B}^{\otimes n})$ consists of 4^n operators,

(15.12)
$$\sigma(f) = \sigma(\alpha_1,\beta_1,\alpha_2,\beta_2,\dots,\alpha_n,\beta_n) \stackrel{\text{def}}{=} \sigma_{\alpha_1,\beta_1} \otimes \sigma_{\alpha_2,\beta_2} \otimes \cdots \otimes \sigma_{\alpha_n,\beta_n}.$$

Here $f \in G^n = \mathbb{F}_2^{2n}$.

We now examine the error-correcting properties of the Shor code. Our goal is to prove that its distance is at least r, i.e., the code detects $r-1$ errors (in fact, the distance is precisely r). By the linearity of the definition it suffices to study errors of the form $\sigma(f)$. Such an error can be decomposed into a classical component and a phase component,

$$\sigma(f) = c\,\sigma(f^{(x)})\,\sigma(f^{(z)}), \quad f^{(x)} = (\alpha_1,0,\alpha_2,0,\dots), \;\; f^{(z)} = (0,\beta_1,0,\beta_2,\dots),$$

where c is a phase factor. Since we assume that $|f| < r$ (where $|f|$ denotes the number of nonzero pairs (α_j,β_j)), we have $|f^{(x)}|, |f^{(z)}| < r$. It suffices to show that for $Z = \sigma(f)$

(15.13)
$$\langle\xi_1|Z|\xi_0\rangle = 0, \qquad \langle\xi_1|Z|\xi_1\rangle = \langle\xi_0|Z|\xi_0\rangle.$$

Let us consider two cases.

1. $f^{(x)} \neq 0$. The error $Z = c\sigma(f^{(x)})\sigma(f^{(z)})$ takes a basis vector to a basis vector. It flips $s = |f^{(x)}|$ bits; in our case $0 < s < r$. The codevectors of the Shor code are linear combinations of special basis vectors: all bits in each row are equal. Flipping s bits breaks this special form; therefore $\langle\xi_a|Z|\xi_b\rangle = 0$.

2. $f^{(x)} = 0$. The error $Z = \sigma(f^{(z)}) = \prod_{j,k}(\sigma^z_{jk})^{\beta_{jk}}$ multiplies each basis vector by ± 1. The special basis vectors in (15.11) are transformed as follows:

$$Z\left|\begin{matrix} y_1 & \cdots & \cdots & y_1 \\ y_2 & \cdots & \cdots & y_2 \\ \cdots & \cdots & \cdots & \cdots \\ y_r & \cdots & \cdots & y_r \end{matrix}\right\rangle = (-1)^{\sum_j \lambda_j y_j}\left|\begin{matrix} y_1 & \cdots & \cdots & y_1 \\ y_2 & \cdots & \cdots & y_2 \\ \cdots & \cdots & \cdots & \cdots \\ y_r & \cdots & \cdots & y_r \end{matrix}\right\rangle,$$

where $\lambda_j = \sum_k \beta_{jk} \in \mathbb{F}_2$. Let $|\lambda|$ denote the number of nonzero components in the vector $\lambda = (\lambda_1, \dots, \lambda_r) \in \mathbb{F}_2^r$. Then $|\lambda| \le |f^{(z)}| < r$. There are three possibilities:

a) $\lambda = (0, \dots, 0)$. In this case $Z|\xi_b\rangle = |\xi_b\rangle$, i.e., the error does not affect codevectors. Therefore $\langle \xi_a | Z | \xi_b \rangle = \delta_{ab}$.

b) $\lambda = (1, \dots, 1)$. This is actually impossible since $|\lambda| < r$.

c) $\lambda \ne (0, \dots, 0), (1, \dots, 1)$. Then

$$\langle \xi_a | Z | \xi_b \rangle = \left(2^{-(r-1)} \sum_{\substack{y_1, \dots, y_r \in \mathbb{F}_2 \\ y_1 + \cdots + y_r = a}} (-1)^{\sum_j \lambda_j y_j} \right) \delta_{ab} = 0.$$

15.7. The Pauli operators and symplectic transformations. The construction of the Shor code uses the symmetry between σ^x and σ^z. We will now study symmetries between σ-operators in more detail.

As already mentioned, the Pauli matrices are conveniently indexed by elements of the group $G = (\mathbb{Z}_2)^2$. General σ-operators (see (15.12)) are indexed by $\gamma = (\alpha_1, \beta_1, \dots, \alpha_n, \beta_n) \in G^n$. The σ-operators form a basis in $\mathbf{L}(\mathcal{B}^{\otimes n})$. Moreover, $\mathbf{L}(\mathcal{B}^{\otimes n})$ becomes a G^n-graded algebra,

$$\mathbf{L}(\mathcal{B}^{\otimes n}) = \bigoplus_{\gamma \in G^n} \mathbb{C}\big(\sigma(\gamma)\big);$$

the operation $\dagger : X \mapsto X^\dagger$ preserves the grading.

The commutation rules for the σ-operators are as follows:

$$\sigma(\gamma_1)\sigma(\gamma_2) = (-1)^{\omega(\gamma_1, \gamma_2)} \sigma(\gamma_2)\sigma(\gamma_1), \tag{15.14}$$

$$\omega(\alpha_1, \beta_1, \dots, \alpha_n, \beta_n; \alpha'_1, \beta'_1, \dots, \alpha'_n, \beta'_n) = \sum_{j=1}^{n} (\alpha_j \beta'_j - \alpha'_j \beta_j) \bmod 2.$$

The multiplication rules are similar:

$$\sigma(\gamma_1)\sigma(\gamma_2) = i^{\widetilde{\omega}(\gamma_1, \gamma_2)} \sigma(\gamma_1 + \gamma_2), \qquad \widetilde{\omega} : G^n \times G^n \to \mathbb{Z}_4. \tag{15.15}$$

Obviously, $\omega(\gamma_1, \gamma_2) = \widetilde{\omega}(\gamma_1, \gamma_2) \bmod 2$.

To obtain an explicit formula for the one-qubit version of the function $\widetilde{\omega}$, we use the equation $\sigma_{\alpha\beta} = i^{\alpha\beta} \sigma_{\alpha 0} \sigma_{0\beta}$. We express $\sigma_{\alpha\beta}$ and $\sigma_{\alpha'\beta'}$ this

way, and commute $\sigma_{0\beta}$ through $\sigma_{\alpha'0}$. The result is as follows:

$$\widetilde{\omega}(\alpha,\beta;\alpha',\beta') = \alpha\beta + \alpha'\beta' - (\alpha\oplus\alpha')(\beta\oplus\beta') + 2\alpha'\beta \mod 4. \tag{15.16}$$

Note that the inner sums in (15.16) are taken modulo 2, whereas the outer sum is taken modulo 4. (For this reason, one cannot expand the product $(\alpha\oplus\alpha')(\beta\oplus\beta')$ and cancel the terms $\alpha\beta$ and $\alpha'\beta'$.) Such a mixture of $\mathbb{Z}_2$-operations and $\mathbb{Z}_4$-operations is rather confusing, so we prefer to write the formula (15.16) in a different form:

$$\widetilde{\omega}(\alpha,\beta;\alpha',\beta') = \alpha^2\beta^2 + (\alpha')^2(\beta')^2 - (\alpha+\alpha')^2(\beta+\beta')^2 + 2\alpha'\beta.$$

In this case, the value $0\in\mathbb{Z}_2$ can be represented by either $0\in\mathbb{Z}_4$ or $2\in\mathbb{Z}_4$, whereas 1 can be represented by either 1 or 3 — the result will be the same.

Finally, we write down the general formula for the function $\widetilde{\omega}$:

$$\begin{aligned} \widetilde{\omega}(\gamma,\gamma') &= \tau(\gamma)+\tau(\gamma')-\tau(\gamma+\gamma')+2\varkappa(\gamma,\gamma'),\\ \tau(\alpha_1,\beta_1,\dots,\alpha_n,\beta_n) &= \textstyle\sum_{j=1}^n \alpha_j^2\beta_j^2 \in \mathbb{Z}_4,\\ \varkappa(\alpha_1,\beta_1,\dots,\alpha_n,\beta_n;\,\alpha_1',\beta_1',\dots,\alpha_n',\beta_n') &= \textstyle\sum_{j=1}^n \alpha_j'\beta_j \in \mathbb{Z}_2. \end{aligned} \tag{15.17}$$

A *unitary transformation* is a superoperator of the form $T = U\cdot U^\dagger : X\to UXU^\dagger$. It is clear that U can be reconstructed from T up to a phase factor, so the group of unitary transformations on n qubits is $\mathbf{U}(\mathcal{B}^{\otimes n})/\mathbf{U}(1)$. We note that the unitary transformations are precisely the automorphisms of the $*$-algebra $\mathbf{L}(\mathcal{B}^{\otimes n})$ (i.e., the linear maps $\mathbf{L}(\mathcal{B}^{\otimes n})\to\mathbf{L}(\mathcal{B}^{\otimes n})$ that commute with the operator multiplication and the operation $\dagger$).

We are interested in those transformations which preserve the grading of $\mathbf{L}(\mathcal{B}^{\otimes n})$ by the σ-operators, i.e., $U\sigma(\gamma)U^\dagger = c(\gamma)\,\sigma(u(\gamma))$, where $u(\gamma)\in G^n$ and $c(\gamma)$ is a phase factor. Since both $U\sigma(\gamma)U^\dagger$ and $\sigma(u(\gamma))$ are Hermitian, $c(\gamma)=\pm1$. Thus we may write

$$U\sigma(\gamma)U^\dagger = (-1)^{v(\gamma)}\sigma(u(\gamma)),\qquad u: G^n\to G^n,\quad v: G^n\to\mathbb{Z}_2. \tag{15.18}$$

The group of such transformations is called the *extended symplectic group* and is denoted by $\mathrm{ESp}_2(n)$. The operators in this group will be called symplectic. We give some examples.

1. σ-operators: $\sigma(f)\sigma(\gamma)\sigma(f)^\dagger = (-1)^{\omega(f,\gamma)}\sigma(\gamma)$. In the case at hand, $u(\gamma)=\gamma$.
2. The operators H and K from the standard basis. We immediately verify that

$$\begin{aligned} &H\sigma^x H^\dagger = \sigma^z, && H\sigma^y H^\dagger = -\sigma^y, && H\sigma^z H^\dagger = \sigma^x;\\ &K\sigma^x K^\dagger = \sigma^y, && K\sigma^y K^\dagger = -\sigma^x, && K\sigma^z K^\dagger = \sigma^z. \end{aligned}$$

Therefore the transformations $H \cdot H^\dagger$ and $K \cdot K^\dagger$ belong to $\mathrm{ESp}_2(1)$. It is easy to see that $\mathrm{ESp}_2(1) \subseteq \mathbf{U}(2)/\mathbf{U}(1) \cong \mathbf{SO}(3)$ is the group of rotational symmetries of a cube; $|\mathrm{ESp}_2(1)| = 24$. This group is generated by the above transformations. The operators H and K themselves generate the *Clifford group* of $24 \cdot 8 = 192$ elements. Thus $\mathrm{ESp}_2(1)$ is the quotient of the Clifford group by the subgroup $\left\{ i^{k/2} I_{\mathcal{B}} : k = 0, \ldots, 7 \right\} \cong \mathbb{Z}_8$.

3. The controlled NOT — the operator $U = \Lambda(\sigma^x)[1,2]$. By definition we have $U|a,b\rangle = |a, a+b\rangle$. The action of U on generators of the algebra $\mathbf{L}(\mathcal{B}^{\otimes 2})$ is as follows:

$$U\sigma_1^z U^\dagger = \sigma_1^z, \qquad U\sigma_1^x U^\dagger = \sigma_1^x \sigma_2^x,$$
$$U\sigma_2^z U^\dagger = \sigma_1^z \sigma_2^z, \qquad U\sigma_2^x U^\dagger = \sigma_2^x.$$

These equations can be verified without difficulty by direct calculation. However, it is useful to bring an explanation to the fore. The operator σ_j^z is a phase shift that depends on the value of the corresponding qubit. The equations in the left column show how these values change under the action of U. The first equation in the right column indicates that flipping the first qubit before applying U has the same effect as flipping both qubits after the action of U. The last equation indicates that flipping the second qubit commutes with U.

Let $T = U \cdot U^\dagger$ be an arbitrary symplectic transformation. The associated function $u : G^n \to G^n$ (see (15.18)) has the following properties:

1. u is linear.
2. u preserves the form ω, i.e., $\omega(u(f), u(g)) = \omega(f, g)$.

Maps with such properties, as is known, are called symplectic; they form the *symplectic group* $\mathrm{Sp}_2(n)$. It is clear that the correspondence $\theta : T \mapsto u$, $\theta : \mathrm{ESp}_2(n) \to \mathrm{Sp}_2(n)$ is a homomorphism of groups.

Theorem 15.6. $\operatorname{Im}\theta = \mathrm{Sp}_2(n)$, $\operatorname{Ker}\theta = G^n$ (*the kernel is the set of* σ*-operators*). *Therefore,* $\mathrm{ESp}_2(n)/G^n \cong \mathrm{Sp}_2(n)$.

For an understanding of the proof it is desirable to know something about cohomology and extensions of groups [**57**]. For the reader unacquainted with these concepts we have prepared a "roundabout way" (see below).

Proof. The transformation (15.18) must be an automorphism of the $*$-algebra $\mathbf{L}(\mathcal{B}^{\otimes n})$. This is the case if and only if the multiplication rules (15.15) are preserved by the action of $T = U \cdot U^\dagger$ (the operation of taking the Hermitian adjoint commutes with T automatically). This means that the function u has properties indicated, and v satisfies the equation

$$v(x+y) - v(x) - v(y) = w(x,y), \tag{15.19}$$

where $w(x,y) = \frac{\widetilde{\omega}(u(x),u(y))-\widetilde{\omega}(x,y)}{2} \in \mathbb{Z}_2$.

In the case where u is the identity map, the right-hand side of (15.19) equals zero. The solutions are all linear functions; this proves that $\operatorname{Ker}\theta = G^n$.

The assertion that $\operatorname{Im}\theta = \mathrm{Sp}_2(n)$ is equivalent to equation (15.19) having a solution for any $u \in \mathrm{Sp}_2(n)$. To prove the existence of the solution, we note that the function w has the following properties:

$$w(y,z) - w(x+y,z) + w(x,y+z) - w(x,y) = 0, \tag{15.20}$$

$$w(x,y) = w(y,x), \tag{15.21}$$

$$w(x,x) = 0. \tag{15.22}$$

Formula (15.20) is the cocycle equation. It indicates that the function w yields a group structure on the Cartesian product $G^n \times \mathbb{Z}^2$ with the multiplication rule $(x,p)\cdot(y,q) = (x+y,\, p+q+w(x,y))$. The group we obtain (we denote it by E) is an extension of G^n with $\mathbb{Z}_2$, i.e., there is a homomorphism $\lambda : E \to G^n$ with kernel $\mathbb{Z}_2$; the homomorphism is defined by $\lambda : (x,p) \mapsto x$.

Equation (15.21) indicates that the group E is Abelian. Finally, (15.22) means that each element of the group E has order 2 or 1. Consequently, $E \cong (\mathbb{Z}_2)^{2n+1}$.

It follows that the extension $E \to G^n$ is trivial: there exists a homomorphism $\mu : G^n \to E$ such that $\lambda\mu = \mathrm{id}_{G^n}$. Writing this homomorphism in the form $\mu : x \mapsto (x, v(x))$, we obtain a solution to equation (15.19). □

There is another, somewhat *ad hoc* way of proving that $\operatorname{Im}\theta = \mathrm{Sp}_2(n)$. Consider the following symplectic transformations: $\left(H \cdot H^\dagger\right)[j]$, $\left(K \cdot K^\dagger\right)[j]$ and $\left(\Lambda(\sigma^x) \cdot \Lambda(\sigma^x)^\dagger\right)[j,k]$. Their images under the homomorphism θ generate the whole group $\mathrm{Sp}_2(n)$. Idea of the proof: it is possible, using these transformations, to take an arbitrary pair of vectors $\gamma_1, \gamma_2 \in G^n$ such that $\omega(\gamma_1,\gamma_2) = 1$ into $(1,0,0,0,\dots)$ and $(0,1,0,0,\dots)$. (See the proof of Lemma 15.9 for an implementation of a similar argument.)

In fact, in this way we can obtain another interesting result. The specified elements of the group $\mathrm{ESp}_2(n)$ generate all the σ-operators, i.e., the kernel of the homomorphism θ. Consequently, the following statement is true:

Proposition 15.7. *The group* $\mathrm{ESp}_2(n)$ *is generated by the elements*

$$\left(H \cdot H^\dagger\right)[j], \quad \left(K \cdot K^\dagger\right)[j], \quad \left(\Lambda(\sigma^x) \cdot \Lambda(\sigma^x)^\dagger\right)[j,k].$$

15.8. Symplectic (stabilizer) codes. These are analogous to the classical linear codes. The role of check sums is played by the σ-operators $\sigma(\alpha_1, \beta_1, \dots, \alpha_n, \beta_n)$.

For example, the Shor code can be represented in this way. Recall that this is a two-dimensional subspace $\mathcal{M} \subseteq \mathcal{B}^{\otimes r^2}$ spanned by the vectors (15.11).

What equations do vectors of $\mathcal{M}$ satisfy?

1. For each $j = 1, \dots, r$ and $k = 1, \dots, r-1$ we have $\sigma^z_{jk}\sigma^z_{j(k+1)}|\xi\rangle = |\xi\rangle$. That is, $|\xi\rangle$ is a linear combination of special basis vectors: each row consists of the repetition of a single bit.
2. For each $j = 1, \dots, r-1$ we have $\left(\prod_{k=1}^r \left(\sigma^x_{jk}\sigma^x_{j(k+1)}\right)\right)|\xi\rangle = |\xi\rangle$. What does this rule mean? The operator $\prod_{k=1}^r \left(\sigma^x_{jk}\sigma^x_{j(k+1)}\right)$ flips all the bits in the j-th and $j+1$-th rows,

$$\left|\begin{array}{c}\cdots\cdots\cdots\\ y_j \quad \dots\, y_j\\ y_{j+1}\, \dots\, y_{j+1}\\ \cdots\cdots\cdots\end{array}\right\rangle \longmapsto \left|\begin{array}{c}\cdots\cdots\cdots\cdots\\ y_j{+}1 \quad \dots\, y_j{+}1\\ y_{j+1}{+}1\, \dots\, y_{j+1}{+}1\\ \cdots\cdots\cdots\cdots\end{array}\right\rangle$$

(the bits in the other rows do not change). These two basis vectors must enter $|\xi\rangle$ with the same coefficients.

It is clear that if $|\xi\rangle$ satisfies conditions 1 and 2, then $|\xi\rangle = c_0|\xi_0\rangle + c_1|\xi_1\rangle$, where $|\xi_a\rangle$ $(a = 0, 1)$ are defined by (15.11).

We now give the general definition of *symplectic codes*, also called *stabilizer codes*. They were introduced (without name) in [**16**]. We will first give a noninvariant definition, which is easier to understand.

A *symplectic quantum code* is a subspace of the form

$$\mathcal{M} = \Big\{|\xi\rangle \in \mathcal{B}^{\otimes n} \,:\, \forall j\ X_j|\xi\rangle = |\xi\rangle\Big\}, \qquad \text{where}$$

$$X_j = (-1)^{\mu_j}\sigma(f_j), \qquad f_j \in G^n, \quad \mu_j \in \mathbb{Z}_2. \tag{15.23}$$

The operators X_j must commute with each other. They are called *check operators*.

The requirement that the check operators commute is equivalent to the condition $\omega(f_j, f_k) = 0$. Indeed, the general commutation relation for operators of the form (15.23) is $X_jX_k = (-1)^{\omega(f_j,f_k)}X_kX_j$. (Note that if $\omega(f_j, f_k) \neq 0$ for some j and k, the subspace $\mathcal{M}$ is empty; this is why we have excluded this case from consideration.) Without loss of generality we may assume that the f_j are linearly independent.

Note that different choices of check operators may correspond to the same code. In fact, the code depends only on the subspace $F \subseteq G^n$ spanned

by the vectors f_j, and on the function $\mu : F \to \mathbb{Z}_2$, interpolating the values $\mu(f_j) = \mu_j$ so that the operators $X_\mu(f) = (-1)^{\mu(f)}\sigma(f)$ satisfy the condition

$$X_\mu(f+g) = X_\mu(f)X_\mu(g) = X_\mu(g)X_\mu(f).$$

Therefore it is preferable to use the following invariant definition.

Definition 15.7. Let $F \subseteq G^n$ be an *isotropic subspace*, i.e., $\omega(f,g) = 0$ for any $f, g \in F$. Also let a function $\mu : F \to \mathbb{Z}_2$ satisfy the equation

$$\mu(f+g) - \mu(f) - \mu(g) = \nu(f,g), \quad \text{where } \nu(f,g) = \frac{\widetilde{\omega}(f,g)}{2} \in \mathbb{Z}_2. \tag{15.24}$$

(The function ν is defined on pairs (f,g) for which $\omega(f,g) = 0$.) Then the corresponding *symplectic code* is

$$\operatorname{SympCode}(F,\mu) \stackrel{\text{def}}{=} \Big\{ |\xi\rangle \in \mathcal{B}^{\otimes n} \,:\, \forall f \in F \;\; \sigma(f)|\xi\rangle = (-1)^{\mu(f)}|\xi\rangle \Big\}. \tag{15.25}$$

Note that the restriction of ν to the subspace F satisfies equations analogous to (15.20)–(15.22); therefore equation (15.24) has a solution. In fact, there are $2^{\dim F}$ solutions; any two solutions differ by a linear function. We call the corresponding codes *congruent*.

Theorem 15.8. $\dim(\operatorname{SympCode}(F,\mu)) = 2^{n-\dim F}$. *The congruent codes form an orthogonal decomposition of* $\mathcal{B}^{\otimes n}$.

Lemma 15.9. *By symplectic transformations, an arbitrary symplectic code* $\operatorname{SympCode}(F,\mu)$ *can be reduced to a trivial one, for which the check operators are* $\sigma^z[1], \dots, \sigma^z[s]$ $(s = \dim F)$.

Proof. Let $f_1 \in F$ be a nonzero vector. The group $\operatorname{Sp}_2(n)$ acts transitively on nonzero vectors, so there is an element $S_1 \in \operatorname{Sp}_2(n)$ such that $S_1 f_1 = (0,1,0,0,\dots) = g_1$. The isotropic space S_1F consists of vectors of the form $(0,\beta_1,\alpha_2,\beta_2,\dots)$. Let $F_1 \subseteq S_1F$ consist of those vectors for which $\beta_1 = 0$. Then $S_1F = \mathbb{F}_2(g_1) \oplus F_1$ and $F_1 \subseteq G^{n-1}$.

Iterating this argument, we find an $S \in \operatorname{Sp}_2(n)$ such that

$$SF = \big\{(0,\beta_1,\,0,\beta_2,\dots,0,\beta_s,\dots,0,0,\dots) \,:\, \beta_1,\dots,\beta s \in \mathbb{F}_2\big\}.$$

By Theorem 15.6, S corresponds to some symplectic transformation $U \cdot U^\dagger$. It takes the code F to a symplectic code given by the check operators $\pm\sigma^z[j]$, $(j = 1,\dots,s)$. All the signs can be changed to "+" by applying a supplementary transformation of the form $\sigma(f)\cdot\sigma(f)^\dagger$. □

We now examine whether a symplectic code $\operatorname{SympCode}(F,\mu)$ is capable of detecting k-qubit errors. Recall that the property of a code to detect an error Z is given by formula (15.9). By linearity, it is sufficient to consider errors of the form $Z = \sigma(g)$, $|g| \le k$.

Let $|\xi\rangle \in \mathrm{SympCode}(F,\mu)$, i.e., $\sigma(f)|\xi\rangle = (-1)^{\mu(f)}|\xi\rangle$ for any $f \in F$. We denote $|\psi\rangle = \sigma(g)|\xi\rangle$. As we will now show, the vector $|\psi\rangle$ belongs to one of the congruent codes $\mathrm{SympCode}(F,\mu')$. Indeed,

$$\begin{aligned}\sigma(f)|\psi\rangle &= \sigma(f)\sigma(g)|\xi\rangle = (-1)^{\omega(f,g)}\sigma(g)\sigma(f)|\xi\rangle = (-1)^{\omega(f,g)+\mu(f)}\sigma(g)|\xi\rangle \\ &= (-1)^{\mu'(f)}|\psi\rangle,\end{aligned}$$

where $\mu'(f) = \mu(f) + \omega(f,g)$. We note that $\mu' = \mu$ if and only if $g \in F_+$, where

$$F_+ = \big\{g \in G^n \,:\, \forall f \in F\ \omega(f,g) = 0\big\}.$$

Obviously, $F \subseteq F_+$.

For the error $Z = \sigma(g)$ there are three possibilities:

1. $g \notin F_+$. Then $\mu' \neq \mu$, hence $|\psi\rangle \perp \mathrm{SympCode}(F,\mu)$. The code detects such an error.
2. $g \in F$. In this case $|\psi\rangle = \sigma(g)|\xi\rangle = (-1)^{\mu(g)}|\xi\rangle$. Such an error is indistinguishable from the identity operator, since it does not alter the codevector (up to the constant phase factor $(-1)^{\mu(g)}$). Condition (15.9) is fulfilled.
3. $g \in F_+ \setminus F$. As in the previous case, $\mu' = \mu$, hence $\mathcal{M} = \mathrm{SympCode}(F,\mu)$ is an invariant subspace for the operator $Z = \sigma(g)$. However, the action of Z on $\mathcal{M}$ is not multiplication by a constant. (Otherwise Z or $-Z$ could be added to the set of check operators without reducing the code subspace, which is impossible.) The code does not detect such an error.

These considerations prove the following theorem.

Theorem 15.10. *The code $\mathcal{M} = \mathrm{SympCode}(F,\mu)$ has distance*

$$d(\mathcal{M}) = \min\big\{|f| \,:\, f \in F_+ \setminus F\big\}.$$

We observe a difference from classical linear codes. There the minimum is taken over a subspace with 0 excluded, whereas for symplectic codes 0 is replaced by the nontrivial subspace F.

Example 15.4. A symplectic code of type $(5,1)$ that corrects 1 error: the subspace F is generated by the rows of the matrix

$$\begin{bmatrix} 1&0&1&0&0&1&0&1&0&0\\ 1&0&0&1&1&0&0&0&0&1\\ 0&1&0&0&0&1&1&0&0&1\\ 0&1&0&1&0&0&0&1&1&0 \end{bmatrix}.$$

It can be verified that $\omega(f_j,f_k) = 0$ for any two rows f_j, f_k. We note that the columns of the matrix come in pairs. If we take any two pairs, then the corresponding four columns are linearly independent. Consequently, for any

$g \neq 0$ supported by these columns, there is a row f_j such that $\omega(f_j, g) \neq 0$. Thus the code distance is greater than 2. (In fact, the distance is 3 since the first 6 columns are linearly dependent.)

[3!] **Problem 15.2.** Prove that no quantum code of type $(4, 1)$ is capable of correcting a single error.

15.9. Toric code. We introduce an important example of a symplectic code. It is constructed as follows. Consider an $r \times r$ lattice on the torus. We put a qubit on each of its edges. In this way we have $2r^2$ qubits. The check operators will be of two types.

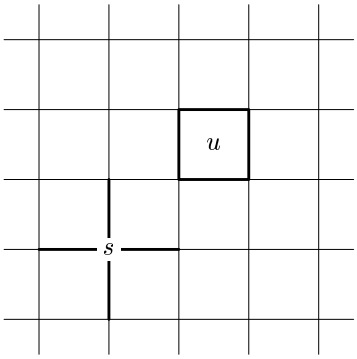


Fig. 15.1. Stabilizer operators for the toric code.

Type I operators are given by vertices. We choose some vertex s and associate with it the check operator

$$A_s^x = \sigma(f_s^x) = \prod_{j \in star(s)} \sigma_j^x.$$

Type II operators are given by faces. We choose some face u and associate with it the check operator

$$A_u^z = \sigma(f_u^z) = \prod_{j \in boundary(u)} \sigma_j^z.$$

The operators A_s^x and A_u^z commute, since the number of common edges between a vertex star and a face boundary is either 0 or 2. (The interchangeability of operators within each type is obvious.)

Although we have indicated $r^2 + r^2 = 2r^2$ check operators (one per a face or a vertex), there are relations between them:

$$\prod_s A_s^x = \prod_u A_u^z = I.$$

(It can be shown that these are the only relations.) Therefore $\dim F = 2r^2 - 2$ and $\dim \mathcal{M} = 2^2$, so two logical qubits can be encoded.

We will now determine the code distance.

For the toric code we have the natural decompositions

$$F = F^{(x)} \oplus F^{(z)}, \qquad F_+ = F_+^{(x)} \oplus F_+^{(z)}$$

into subspaces of operators that consist either only of σ_j^x or only of σ_j^z. Such codes are called *CSS codes* (by the names of their inventors, Calderbank, Shor [**17**] and Steane [**68**]).

In the case of the toric code, the subspaces $F^{(z)}$, $F^{(x)}$, $F_+^{(z)}$, $F_+^{(x)}$ have a topological interpretation. Vectors of the form $(0, \beta_1, \dots, 0, \beta_n) \in G^n$ can be regarded as 1-chains, i.e., formal linear combinations of edges with coefficients $\beta_1, \dots, \beta_n \in \mathbb{Z}_2$. The basis elements of $F^{(z)}$ are the boundaries of 2-cells. Therefore $F^{(z)}$ is the space of 1-boundaries. Likewise, vectors of the form $(\alpha_1, 0, \dots, \alpha_n, 0)$ are regarded as 1-cochains; $F^{(x)}$ is the space of 1-coboundaries.

Let us take an arbitrary element $g \in F_+$, $g = g^{(x)} + g^{(z)}$. The commutativity between g and F can be written as follows:

$$\omega(f_s^x, g^{(z)}) = 0, \quad \omega(f_u^z, g^{(x)}) = 0, \quad \text{for each vertex } s \text{ and face } p.$$

To satisfy $\omega(f_s^x, g^{(z)}) = 0$ it is necessary that each star contain an even number of edges from $g^{(z)}$. In other words, $g^{(z)}$ is a 1-cycle. Analogously, $g^{(x)}$ must be a 1-cocycle.

Thus, the spaces $F_+^{(z)}$, $F_+^{(x)}$ consist of 1-cycles and 1-cocycles (with $\mathbb{Z}_2$ coefficients), and the spaces $F^{(z)}$, $F^{(x)}$ consist of 1-boundaries and 1-coboundaries. The sets $F_+^{(z)} \setminus F^{(z)}$ and $F_+^{(x)} \setminus F^{(x)}$ are formed by cycles and cocycles that are not homologous to 0. Consequently the code distance is the minimum size (the number of nonzero coefficients) of such a cycle or cocycle. It is easy to see that this minimum equals r. This shows that the toric code corrects $\lfloor (r-1)/2 \rfloor$ errors.

Remark 15.4. The family of toric codes (with $r = 1, 2, \dots$) provides an example of *local check codes*. Specifically, the following conditions are satisfied:

- each check operator acts on a uniformly bounded number of qubits;
- each qubit enters a uniformly bounded number of check operators;
- the family contains codes with arbitrarily large distance.

Such codes are interesting in that syndrome measurement (an important part of error correction; see below) can be realized by a constant depth circuit. Therefore an error in the execution of this circuit will affect only a

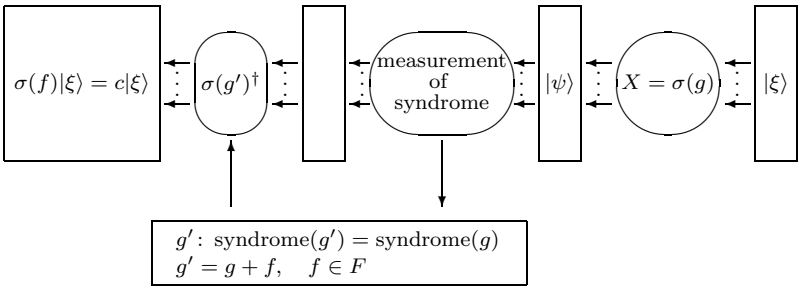


Fig. 15.2. Error correction procedure for symplectic codes.

bounded number of qubits — a useful property for fault-tolerant computation.

15.10. Error correction for symplectic codes. Definition 15.4 and Theorem 15.3 indicate only an abstract possibility for restoring the initial codevector. We will show how to realize an error correction procedure for a symplectic code $\mathcal{M} = \mathrm{SympCode}(F, \mu)$.

We examine a special case where the error is a σ-operator, $W = \sigma(g)$. Let $X_j = (-1)^{\mu_j}\sigma(f_j)$ be the check operators, and F the corresponding isotropic subspace. The sequence of bits $\lambda(g) = (\omega(f_1, g), \ldots, \omega(f_s, g))$ is called the *syndrome* of g. Each of these bits can be measured by measuring the eigenvalue of X_j on the quantum state $|\psi\rangle = W|\xi\rangle$ (in fact, $X_j|\psi\rangle = (-1)^{\omega(f_j,g)}|\psi\rangle$). The measurement of one bit does not change the values of the other, because the check operators commute.

Suppose that $|g| \le k$ and the code corrects k errors (i.e., $d(\mathcal{M}) > 2k$). Is it possible to reconstruct the error from its syndrome? Two errors $g, g' \in G^n$ have the same syndrome if and only if $g' - g \in F_+$. The error correction property of the code implies that

$$\forall\, g, g' \ \big(|g| \le k,\ |g'| \le k\big) \Longrightarrow \big((g' - g \in F) \vee (g' - g \notin F_+)\big).$$

Therefore we may take a different error g' for g, but only if $g' - g \in F$.

It is now clear how we should correct errors. After the syndrome is determined, we reconstruct the error (up to an element $f = g' - g \in F$) and apply the operator which is inverse to the operator of the supposed error. Thus we obtain a state that differ from the initial one by a phase factor, $\sigma(g')^{-1}\sigma(g)|\xi\rangle = c|\xi\rangle$.

We have considered the case of an error of type $\sigma(g)$. But, actually, it is required that an error-correcting transformation protect against all

superoperators of the form

$$T = \sum_{|h|\le k, |h'|\le k} b_{h,h'} \sigma(h) \cdot \sigma(h')^\dagger.$$

As an exercise the reader is encouraged to verify how the above procedure works in this general case.

[3] **Problem 15.3.** Construct a polynomial algorithm for reconstructing an error from its syndrome for the toric code.

15.11. Anyons (an example based on the toric code). Using the construction of the toric code, we will try to give a better idea of Abelian anyons mentioned in the Introduction (non-Abelian anyons are considerably more complicated).

Once again, we consider a square lattice on the torus (or on the plane — now we are only interested in a region with trivial topology). As earlier, associated to each vertex s and each face u are the operators

$$A_s^x = \prod_{j \in star(s)} \sigma_j^x, \qquad A_u^z = \prod_{j \in boundary(u)} \sigma_j^z.$$

The codevectors are characterized by the conditions $A_s^x|\xi\rangle = |\xi\rangle$, $A_u^z|\xi\rangle = |\xi\rangle$. There is a different way to impose the same conditions. Consider the following *Hamiltonian* — the Hermitian operator

$$H = \sum_s (I - A_s^x) + \sum_u (I - A_u^z). \tag{15.26}$$

This operator is nonnegative, and its null space coincides with the code subspace of the toric code. Thus, the vectors of the code subspace are the ones that possess the minimal energy (i.e., they are the eigenvectors corresponding to the smallest eigenvalue of the Hamiltonian). In physics, such states are called *ground states*, and vectors in the orthogonal complement are called *excited states*.

Excited states can be classified by the set of conditions they violate. Specifically, the states violating a particular set of conditions form a subspace; such subspaces form an orthogonal decomposition of the total state space. Note that the number of violated conditions of each type is even since $\prod_s A_s^x = \prod_u A_u^z = I$.

Consider an excited state $|\eta\rangle$ with the smallest nonzero energy. Such a state violates precisely two conditions, for instance, at two vertices, s and p. (In fact, the states violating different pairs of conditions may form linear combinations, but we will assume that s and p are fixed.) Then for these particular vertices

$$A_s^x|\eta\rangle = -|\eta\rangle, \qquad A_p^x|\eta\rangle = -|\eta\rangle,$$

whereas the conditions with the "+" sign for the other vertices remain in force. We say that in the state $|\eta\rangle$ there are two *quasiparticles* (elementary excitations) located at the vertices s and p. Thus quasiparticle is a mental device for classifying excited states. It is a special property of Hamiltonian (15.26) that states with certain quasiparticle positions are also eigenstates. However, the classification of low energy excited states by quasiparticle positions, though approximate, works amazingly well for most physical media.[15]

How can we get the state $|\eta\rangle$ from the ground state $|\xi\rangle$? We join p and s by a lattice path C_1 (see Figure 15.3a) and act on $|\xi\rangle$ by the operator $W = \prod_{j\in C_1} \sigma_j^z$. This operator commutes with A_k^x for all internal vertices of the path C_1, but at the ends they anti-commute: $WA_s^x = -A_s^x W$, $WA_p^x = -A_p^x W$. We set $|\eta\rangle = W|\xi\rangle$ and show that $|\eta\rangle$ satisfies the required properties. For the vertex s (and analogously for p) we have

$$A_s^x|\eta\rangle = A_s^x W|\xi\rangle = -WA_s^x|\xi\rangle = -W|\xi\rangle = -|\eta\rangle.$$

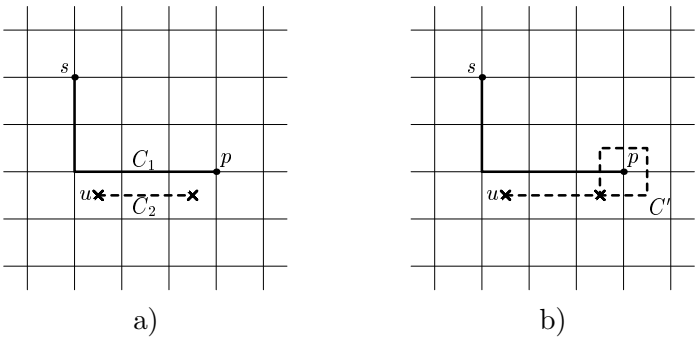


Fig. 15.3. Creating pairs of quasiparticles (a) and moving them around (b).

An arbitrary state of the system can be described as a set of quasiparticles of two types, one of which "lives" on vertices, the other on faces. Mathematically, a quasiparticle is simply a violated code condition, but now we think of it as a physical object. Particles-excitations can move, be created and annihilated. A pair of vertex quasiparticles is created by the action of the operator W; a pair of face quasiparticles is created by the operator $V = \prod_{j\in C_2} \sigma_j^x$, where C_2 is a path on the dual lattice.

What will happen if we move a face quasiparticle around a vertex quasiparticle (see Figure 15.3b)? The initial state $|\psi\rangle$ contains two quasiparticles of each type. The movement of the face quasiparticle around a closed path

[15] The position uncertainty is much larger than the lattice spacing, but much smaller than the distance between the particles. We cannot go into more details here. Thus, reader's acquaintance with conventional quasiparticles (such as electrons and holes in a semiconductor, or spin waves) would be very helpful.

C' is expressed by the operator $U = \prod_{j\in C'} \sigma_j^x = \prod_s A_s^x$, where s runs over all faces inside C'. It is obvious that $A_k^x|\psi\rangle = |\psi\rangle$ for all $k \neq p$. As a result we get

$$U|\psi\rangle = \prod_{j\in C'} \sigma_j^x|\psi\rangle = A_p^x|\psi\rangle = -|\psi\rangle.$$

Thus the state vector gets multiplied by -1. This indicates some sort of long range interaction between the particles: the moving particle somehow "knows" about the second particle without ever touching it! However, the interaction is purely topological: the state evolution depends only on the isotopy class of the braid the particle world lines form in space-time. In the case at hand, the evolution is just the multiplication by a phase factor; such particles are called Abelian anyons.

On the torus we can move particles over two different cycles that form a basis of the homology group. For instance, create a pair of particles from the ground state, move one of them around a cycle, and annihilate with the second one. Now it becomes important that the ground state is not unique. Recall that there is a 4-dimensional space of ground states — the code subspace. The process we have just described affects an operator acting on this subspace. We can think of four different operators of this kind: Z_1, Z_2 are products of σ_j^z over the basis cycles (they correspond to moving a vertex quasiparticle), whereas X_1, X_2 are products of σ_j^x over the homologous cycles on the dual lattice. The commutation relations between these operators are as follows:

$$\begin{array}{llll} Z_jZ_k = Z_kZ_j, & X_jX_k = X_kX_j & (j,k=1,2), \\ X_1Z_1 = Z_1X_1, & X_2Z_2 = Z_2X_2, & \\ X_1Z_2 = -Z_2X_1, & X_2Z_1 = -Z_1X_2. & \end{array} \tag{15.27}$$

Thus we can identify the operators Z_1 and X_2 with σ^z and σ^x acting on one encoded qubit. Correspondingly, Z_2 and X_1 act on the second encoded qubit.

Solutions

In this part we offer either complete solutions to problems or hints which the interested reader can use to work out a rigorous solution.

S1. Problems of Section 1

1.1. The idea is simple: the machine moves symbols alternately from left to right and from right to left until it reaches the center of the input string, at which point it stops.

Now we give a formal description of this machine.

We assume that the external alphabet A is $\{0, 1\}$. The alphabet $S = \{\sqcup, 0, 1, *, 0', 1'\}$ consists of the symbols of the external alphabet, the empty symbol $\sqcup$, and three auxiliary marks used to indicate the positions from which a symbol is taken and a new one should be dropped.

The set of states is

$$Q = \{q_0, q_f, r_0, r_1, l_0, l_1, l_{0'}, l_{1'}\}.$$

The letters r and l indicate the direction of motion, and the subscripts at these letters refer to symbols being transferred.

Now we describe the transition function.

Beginning of work:

$$(q_0, 0) \mapsto (r_0, *, +1), \qquad (q_0, 1) \mapsto (r_1, *, +1),$$
$$(q_0, \sqcup) \mapsto (q_0, \sqcup, -1).$$

The first line indicates that the machine places a mark in the first position and moves the symbol that was there to the right. The second line indicates that the machine stops immediately at the empty symbol.

Transfer to the right:

$$(r_0,0)\mapsto(r_0,0,+1),\qquad (r_1,0)\mapsto(r_1,0,+1),$$
$$(r_0,1)\mapsto(r_0,1,+1),\qquad (r_1,1)\mapsto(r_1,1,+1).$$

The machine moves to the right until it encounters the end of the input string or a mark.

A change in the direction of motion from right to left consists of two actions: remove the mark (provided this is not the empty symbol)

$$(r_0,0')\mapsto(l_{0'},0,-1),\qquad (r_1,0')\mapsto(l_1,0,-1),$$
$$(r_0,1')\mapsto(l_{0'},1,-1),\qquad (r_1,1')\mapsto(l_{1'},1,-1),$$
$$(r_0,\sqcup)\mapsto(l_{0'},\sqcup,-1),\qquad (r_1,\sqcup)\mapsto(l_{1'},\sqcup,-1)$$

and place it in the left adjacent position

$$(l_{0'},0)\mapsto(l_0,0',-1),\qquad (l_{1'},0)\mapsto(l_0,1',-1),$$
$$(l_{0'},1)\mapsto(l_1,0',-1),\qquad (l_{1'},1)\mapsto(l_1,1',-1).$$

Transfer to the left:

$$(l_0,0)\mapsto(l_0,0,-1),\qquad (l_1,0)\mapsto(l_1,0,-1),$$
$$(l_0,1)\mapsto(l_0,1,-1),\qquad (l_1,1)\mapsto(l_1,1,-1).$$

Change of direction from left to right:

$$(l_0,*)\mapsto(q_0,0,+1),\qquad (l_1,*)\mapsto(q_0,1,+1).$$

The completion of work depends on the parity of the word length: for even length, the machine stops at the beginning of the motion to the right

$$(q_0,0')\mapsto(q_f,0,-1),\qquad (q_0,1')\mapsto(q_f,1,-1),$$

and for odd length, — at the beginning of the motion to the left

$$(l_{0'},*)\mapsto(q_f,0,-1),\qquad (l_{1'},*)\mapsto(q_f,1,-1).$$

The transition function is undefined for the state q_f; therefore the machine stops after switching to this state.

1.2. Schematically, this is done as follows: to the second summand we add one by one the bits of the first summand, the added bit being erased. Adding one bit takes time that does not exceed the binary length of the second summand, so that the total working time of the machine depends quadratically on the length of the input.

1.3. The proof is by contradiction. Suppose that there is such an algorithm, i.e., that there exists a machine B which, for the input $([M],x)$, gives the answer "yes" if the machine M stops at input x and gives the answer "no" otherwise. (Recall that $[M]$ denotes a description of the machine M.)

Let us define another machine B' that, given an input y, simulates the work of B for the input (y, y). If the answer of the machine B is "yes", then B' begins moving the head to the right and does not stop. If the answer of B is "no", then B' stops.

Does B' stop for the input $[B']$?

If we suppose that it stops, then B gives the answer "yes" for the input $([B'], [B'])$. Then, by definition of the machine B', it does not stop for the input $[B']$. This is exactly the opposite of our assumption.

If B' does not stop for the input $[B']$, then B gives the answer "no" for the input $([B'], [B'])$. But this implies that B' stops for the input $[B']$, a contradiction.

Remark S1.1. This kind of proof is fairly common in mathematical logic; it is often called *diagonalization*. The idea was first used by Cantor to show that the set of real numbers (or infinite 0-1 sequences) is uncountable. We remind the reader Cantor's argument to explain the name "diagonalization". Suppose that all 0-1 sequences are counted (i.e., assigned numbers $0, 1, 2, \dots$), so that we can think of them as rows of an infinite table,

$$\begin{array}{cccc} x_{00} & x_{01} & x_{02} & \dots \\ x_{10} & x_{11} & x_{12} & \dots \\ x_{20} & x_{21} & x_{22} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

Let us look at the diagonal of this table and build the sequence

$$y = (1 - x_{00},\, 1 - x_{11},\, 1 - x_{22}, \dots).$$

It is clear that this sequence cannot be a row of the original table, which proves that all 0-1 sequences cannot be counted.

Note, however, that the unsolvability of the halting problem is a bit more subtle: the proof is based on the existence of a universal Turing machine (we used it implicitly when constructing B').

1.4. First of all, we show that the elements of an enumerable set can actually be produced one by one by an algorithmic process. Suppose that X is the set of all possible outputs of a Turing machine E. Let us try all pairs of the form (x, n) (where x is a string, and n is a natural number) and simulate the first n steps of E for the input x. If E terminates during the first n steps, we include its output in the list; otherwise we proceed to the next pair (x, n). This way, all elements of X are included (possibly, with repetitions).

Theorem S1.1. *A partial function $F : A^* \to \{0,1\}$ is computable if and only if the sets $X_0 = \{x : F(x) = 0\}$ and $X_1 = \{x : F(x) = 1\}$ are both enumerable.*

Proof. Suppose that X_0 and X_1 are enumerable. Given an input string $y \in X_0 \cup X_1$, we run the enumerating processes for X_0 and X_1 in parallel. Sooner or later, y will be produced by one of the processes. If it is the process for X_0, we announce 0 as the result, otherwise the result is 1.

Conversely, if F is computable, then there is a TM that presents its input x as the output if $F(x) = 0$, and runs forever if $F(x)$ is 1 or undefined. Therefore X_0 is enumerable. (Similarly, X_1 is enumerable.) □

Now, let us turn to the original problem. We are interested in the first set of these two:

$$X_0 = \big\{[M] : M \text{ does not halt for the empty input}\big\},$$
$$X_1 = \big\{[M] : M \text{ halts for the empty input}\big\}.$$

Note that the second set, X_1, is enumerable. Indeed, we can construct a machine E that, given an input $x = [M]$, simulates M and outputs the same x when the simulation ends. (If M does not stop, or if the input string is not a valid description of a TM, then E runs forever.) Therefore, if X_0 were also enumerable, there would exist an algorithm for determining whether a given Turing machine M halts for the empty input.

But then the halting problem (for a Turing machine T and an arbitrary input x) would also be solvable. Indeed, for each pair $([T], x)$ a machine M that first writes x on the tape and then simulates the work of T is easily constructed. We have arrived at a contradiction: according to Problem 1.3, there is no algorithm for the solution of the halting problem.

1.5. The idea of a solution is as follows. Let b be an arbitrary computable function. For any n there exists a machine M_n that writes n on the tape, then computes $nb(n)$ and counts down from $nb(n)$ to zero. This machine has $O(\log n)$ states and a constant number of symbols. (It is easy to see that $O(\log n)$ states are enough to write n in binary form on the tape.)

1.6. We describe briefly a single-tape machine M_1 that simulates a two-tape machine M_2. The alphabet of M_1 is rather large: one symbol encodes four symbols of M_2, as well as four additional bits. The symbol in the k-th cell of M_1 represents the symbols in the k-th cells on the input tape, the output tape and both work tapes of M_2; the additional bits are used to mark the places where the heads of M_2 are located. The control device of M_1 keeps the state of the control device of M_2 and some additional information.

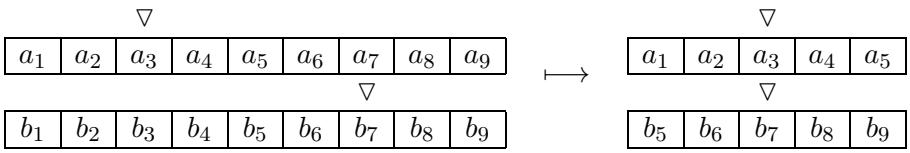
The machine M_1 works in cycles. Each cycle imitates a single step of M_2. At the beginning of each cycle the head of M_1 is located above the

leftmost cell. Each cycle consists of two passes. First M_1 moves from left to right until it finds all of the four marks (i.e., the heads of M_2); the contents of the corresponding cells are stored in the control device. On the way back actions imitating one step of M_2 are carried out. Each such action requires $O(1)$ steps (and finite amount of memory in the control device).

Each cycle takes $O(s)$ steps of the machine M_1, where s is the length of the used portion of the tape. Since $s \le T(n) + 1$, the machine M_1 works in time $O(sT(n)) = O(T^2(n))$.

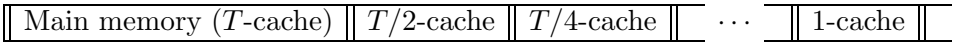
1.7. The main problem in efficient simulation of a multitape TM on an ordinary TM is that the heads of the simulated machine may be far from each other. Therefore the simulating head must move back and forth between them to imitate a single step of the multitape TM.

However, if we have a second work tape, it can be used to move blocks of size n by distance m along the first tape in $O(n + m)$ steps. Indeed, we can copy the block onto the second tape, then move the head on the first tape and then copy the block back. Therefore we can build a "cache" that contains neighborhoods of the heads aligned according to head positions:



After simulating several computation steps in the cache, we can copy the result (changed cache contents) back. Specifically, to simulate t steps, we need to copy the t-neighborhoods of the heads (of size $2t + 1$). We call it *t-cache*.

To get a bound $T(n) \log T(n)$, we need to use recursion and multilevel caching (by powers of 2). Suppose we have already simulated $T = 2^k$ steps of the 3-tape machine M_3 on a 2-tape machine M_2 and want to continue the simulation for another T steps. The current state of M_3 is represented by the first $T + 1$ cells on the first tape of M_2. We extend this "main memory" to size $2T+1$ and perform the simulation in two portions, using a $T/2$-cache. To carry out the computation in this cache, we use a $T/4$ cache, and so on. (All caches are allocated on the first tape.)



Thus, each level of recursion consists in simulating t steps of M_3 in the t-cache. This is done by a procedure F that consists of the the following operations:

1. copy the $t/2$-neighborhoods of the heads into the $t/2$-cache;

2. simulate $t/2$ steps of M_3 recursively (by applying the same procedure F to the $t/2$-cache);
3. copy the result back to the t-cache;
4. copy the $t/2$-neighborhoods of the new head positions into the $t/2$-cache;
5. simulate the remaining $t/2$ steps;
6. write the result back.

To implement the recursion, we augment each cache by a special cell that indicates the operation being done at the current level. Caches are allocated as they are needed, and freed (and filled with blanks) when returning to the previous recursion level. (This is a standard implementation of recursion using a stack.)

The recurrence relation $T(t) \le 2T(t/2) + O(t)$ implies $T(t) = O(t \log t)$.

1.8. Loosely speaking, to copy a string of length n we need to move n bits by distance n. Since a TM has a finite number of states, it carries only $O(1)$ bits by distance 1 at each step; therefore $\Omega(n^2)$ steps are needed.

Here is a more formal argument. For each k we consider a *crossing sequence at boundary k*. It is the sequence of states of the TM during its moves from the k-th cell to the $(k+1)$-th cell. Note that the behavior of the TM in the zone to the right of the boundary (cells $k+1$, $k+2$, etc.) is determined by the initial contents of that zone and the crossing sequence. (The only information that flows into the zone is carried by the head and is recorded in the crossing sequence. Note also that we do not worry how long the head stays outside the zone and do not include this information in the crossing sequence.)

Different input strings should generate different crossing sequences. Therefore, most crossing sequences are long (of size $\Omega(n)$). Since there are $\Omega(n)$ possible values of k, we have $\Omega(n)$ crossing sequences of length $\Omega(n)$. Thus the sum of their lengths is $\Omega(n^2)$, which is a lower bound for the computation time.

Here are the details. For simplicity we assume that n is even ($n = 2m$) and consider inputs of the form $x = v0^m$, where v is a binary string of length m. Let k be a number between m and $2m$, and let $Q(v, k)$ be the crossing sequence for the computation on $v0^m$ at boundary k. As we have said, different strings v lead to different crossing sequences (otherwise different strings would have identical copies); therefore there are at least 2^m crossing sequences. Since the number of states is $O(1)$, some crossing sequences have length $\Omega(m)$. Moreover, it is easy to see that the average length of the crossing sequence $Q(v, k)$ (taken over all strings $v \in \{0,1\}^m$ for a fixed k) is

$\Omega(m)$. Therefore, the average value of

$$\sum_{m\le k\le 2m} \big(\text{length of } Q(v,k)\big)$$

is $\Omega(m^2)$. But for each v this sum does not exceed the computation time; therefore the average is also a lower bound for the computation time.

On the other hand, it is easy to construct a Turing machine M which will duplicate *some* strings (e.g., 0^n) in time $T'(n) = O(n\log n)$. First M checks whether the input consists only of zeros. If this is the case, M counts them and then produces the same number of zeros. (To count zeros, we need a portable counter of size $O(\log n)$, which is carried along the tape.) If the input has nonzero symbols, then M just copies it in the usual way (in $O(n^2)$ steps). But the minimal time is still $O(n\log n)$. One can check that this bound is tight (i.e., $\Omega(n\log n)$ is a lower bound).

1.9. *Hint.* We need to simulate an arbitrary TM. Since the values of the variables can be arbitrarily large, we can store the entire tape of the machine in one variable (tape content is the $|S|$-ary representation of some number, where S is the machine alphabet). The head position is another integer variable, and the state of the control device is yet another.

Changes in these variables after one computation step are described in terms of simple arithmetic operations (addition, multiplication, exponentiation, division, remainder) and comparison of numbers. All these operations can be reduced to the increment and decrement statements and the comparison with 0 (using several auxiliary variables).

The transition table becomes a nested `if-then-else` construct.

S2. Problems of Section 2

2.1. Let us find all functions in two variables that can be expressed by formulas in the basis $\mathcal{A}$. We begin with two *projections*, namely, the functions $p_1(x,y) = x$ and $p_2(x,y) = y$. Then the following procedure is applied to the set $\mathcal{F}$ of already constructed functions. We add to the set $\mathcal{F}$ all functions of the form $f\big(g_1(x_1,x_2),\, g_2(x_3,x_4),\ldots,g_k(x_{2k-1},x_{2k})\big)$, where $x_j\in\{x,y\},\ g_j\in\mathcal{F},\ f\in\mathcal{F}$. If the set $\mathcal{F}$ increases, we repeat the procedure. Otherwise there are two possibilities: either we have obtained all functions in two variables (then the basis is complete), or not (then the basis is not complete).

We estimate the working time of this algorithm. Only 16 Boolean functions in two variables exist; therefore the set $\mathcal{F}$ can be enlarged at most 14 times. At each step we must check at most $16^m\cdot|\mathcal{A}|$ possibilities, where m is the maximum number of arguments of a basis function. Indeed, for each basis function f each of (at most) m positions can be occupied by any

function from $\mathcal{F}$, and $|\mathcal{F}| \le 16$. The length of the input (encoded basis) is at least 2^m (because the table for a function in m Boolean variables has 2^m entries). So, the working time of the algorithm is polynomially bounded in the length of the input.

2.2. An upper bound $O(n2^n) < 2.01^n$ (for large n) follows immediately from the representation of the function in disjunctive normal form (see formula (2.1) on page 19).

To obtain a lower bound we compare the number of Boolean functions in n variables (i.e., 2^{2^n}) and the number of all circuits of a given size. Assume that the standard complete basis is used. For the k-th assignment of the circuit there are at most $O((n+k)^2)$ possibilities (two arguments can be chosen among n input and $k-1$ auxiliary variables). Therefore, the number N_s of different circuits of size s does not exceed

$$O\big(\big((n+s)^2\big)^s\big) = 2^{2s(\log(n+s)+O(1))}.$$

But the number of Boolean functions in n variables equals 2^{2^n}. If

$$2^n > 2s\big(\log(n+s) + O(1)\big), \tag{S2.1}$$

there are more functions than circuits, so that $c_n > s$. If $s = 1.99^n$, then inequality (S2.1) is satisfied for sufficiently large n.

2.3. We recall the construction of the undecidable predicate belonging to P/poly (see Remark 2.1 on page 22).

For any function $\varphi : \mathbb{N} \to \{0,1\}$ the predicate $f_\varphi(x) = \varphi(\text{length}(x))$ belongs to P/poly. Now let φ be a computable function that is difficult to compute: no TM can produce output $\varphi(n)$ in polynomial (in n) time. More precisely, we use a computable function φ such that for any TM M and any polynomial p with integer coefficients there exists n such that $M(1^n)$ does not produce $\varphi(n)$ after $p(n)$ steps.

It remains to construct such a function φ. This can be done by "diagonalization" (cf. Remark S1.1): we consider pairs (M,p) one by one; for each pair we select some n for which $\varphi(n)$ is not defined yet and define $\varphi(n)$ to be different from the result of $p(n)$ computation steps of M on input 1^n. (If computation does not halt after $p(n)$ steps, the value $\varphi(n)$ can be arbitrary.)

2.4. Each output depends on $O(1)$ wires; each of them depends on $O(1)$ other wires, etc. Therefore, the total number of used wires and gates is $O(1)^{\text{depth}} = 2^{O(\log(m+n))} = \text{poly}(m+n)$.

2.5. A circuit consists of assignments of the form $y_j := f_j(u_1, \ldots, u_r)$. We can perform this assignments symbolically, by substituting formulas for $u_1, \ldots, u_r$. If these formulas have depth $\le h$, then y_j becomes a formula of depth $\le h+1$.

2.6. A formula can be represented by a tree (input variables are leaves, internal vertices correspond to subformulas, whereas the root is the formula itself.)

It is easy to see that any formula X of size L has a subformula Z of size between M and $2M$ (inclusive), where $M = \lfloor L/3 \rfloor$. Indeed, we start looking for such a subformula at the root of the tree and each time choose the largest branch (of one or two). When the size of the subformula becomes $\le 2M$, we stop.

Replacing the subformula Z by a new variable z, we obtain a formula $Y(z)$ (of size from $L-2M$ to $L-M$) such that $X = Y(Z)$. Note that both Z and Y have size $\le \lceil 2L/3 \rceil$.

Suppose that Z and Y can be converted into equivalent formulas Z' and Y' of depth $\le h$. Then the following formula of depth $\le h+3$ will compute the same function as X does:

$$X' = (Y(0) \wedge \neg Z) \vee (Y(1) \wedge Z).$$

Thus, we have the recurrence relation

$$h(L) \le h\big(\lceil 2L/3 \rceil\big) + 3,$$

where $h(L)$ is the maximum (over all formulas X of size L) of the minimal depth of a formula X' that is equivalent to X. It follows that $h(L) = O(\log L)$.

2.7. Let us use the disjunctive normal form. If we construct a circuit by formula (2.1) using *AND* gates and *OR* gates with arbitrary fan-in (the number of inputs), only three layers are needed: one layer for negations, one for conjunctions, and one for the disjunction.

2.8. Let L be the size of a circuit computing the function $f = \mathrm{PARITY}$. First, we convert the circuit into a formula of size $L' \le L^3$ (see Problem 2.5). Using De Morgan's identities

$$\neg \bigvee x_j = \bigwedge \neg x_j, \qquad \neg \bigwedge x_j = \bigvee \neg x_j,$$

we can ensure that negations are applied only to the input variables.

Without loss of generality, we may assume that the output is produced by an *OR* gate. (Otherwise, we apply De Morgan's identities again and obtain a circuit for the function $\neg\,\mathrm{PARITY} = \mathrm{PARITY} \oplus 1$; the following arguments work for this function as well.) We may also assume that the inputs to the final *OR* gate are produced by *AND* gates. (If some input is actually produced by an *OR* gate, this gate can be merged with the final one. If it is produced by a *NOT* gate, we can insert a dummy *AND* gate and still have depth 3.)

Now we have

$$f(x_1, \dots, x_n) = t_1 \vee \dots \vee t_m,$$

where each t_i has the form $t_i = u_1 \wedge \dots \wedge u_k$, and each u_k is either a disjunction (a single variable is a special case of that) or the negation of a variable. Note that a variable cannot appear in a negation and a disjunction at the same time, or else the formula can be simplified. For example, if

$$t_i = (x_1 \vee \dots \vee x_k) \wedge \dots \wedge \neg x_1 \wedge \dots ,$$

then x_1 can be deleted from the disjunction $(x_1 \vee \dots \vee x_k)$. Therefore, each t_i has the form

$$t_i = \neg x_{j_1} \wedge \dots \wedge \neg x_{j_p} \wedge \text{ (monotone function in the other variables).}$$

Now we use a special property of the function $f = \text{PARITY}$: if $f(x) = 1$, and x' differs from x in just one bit (say, x_j), then $f(x') = 0$. This condition should be true for all subformulas t_i. It follows that each t_i is the conjunction of n literals (i.e., input variables or their negations). Therefore, $t_i(x) = 1$ for exactly one value of x. Hence the number of the subformulas t_i is not less than the number of points x where $f(x) = 1$, i.e., 2^{n-1}.

Remark. It can be shown that circuits of fixed depth computing the function PARITY and made of gates *NOT*, *OR* and *AND* with arbitrary fan-in, always have exponential size. The proof (quite nontrivial!) is by induction, starting with circuits of depth 2 and 3 (the case discussed above).

The proof of this assertion can be found in [**14**]. We give a short exposition of the main idea. Let us note that *OR*-gates and *AND*-gates in a circuit of minimal size must alternate. One can try to switch two layers by replacing each disjunction of conjunctions by a conjunction of disjunctions; this will reduce the circuit depth by 2. However, this transformation increases the size of the circuit, and we do not get any satisfactory bound for the new size. Still, a reasonable bound can be obtained if we allow further transformation of the resulting circuit. We first assign random values to some of the input variables. Thus we obtain a function of a smaller number of variables, which is either PARITY or its negation. As far as the circuit is concerned, some of its auxiliary variables can be evaluated using only the values of the input variables we have just fixed. And with nonnegligible probability our circuit becomes simpler, so that the transposition of conjunctions and disjunctions does not lead to a large increase in the circuit size.

2.9. There are three possible results of the comparison of two numbers x and y: $x > y$, $x = y$, or $x < y$. We construct a circuit which yields a two-bit answer encoding these three possibilities.

We may assume without loss of generality that n is a power of two. (It is possible to add several zeros on the left so that the total number of bits becomes a power of two. This at most doubles the number of inputs bits.)

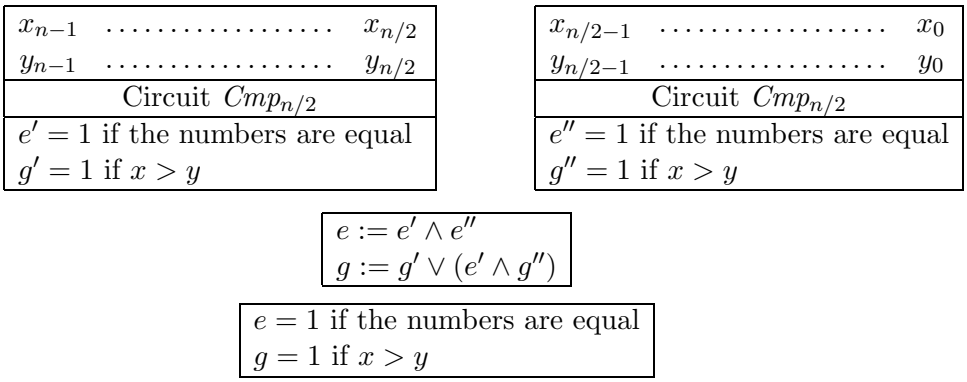


Fig. S2.1. Circuit Cmp_n for comparison of n-bit numbers (size of circuit = $O(n)$, depth = $O(\log n)$).

A circuit for the comparison of n-bit numbers is constructed recursively. We compare the first $n/2$ (high) bits and the last $n/2$ (low) bits separately, and then combine the results, see Figure S2.1. (This "divide and conquer" method will be used for the solution of many other problems.)

We estimate the size L_n and depth d_n of this circuit. It is easy to see that

$$L_n = 2L_{n/2} + 3, \qquad d_n = d_{n/2} + 2.$$

Therefore, $L_n = O(n)$ and $d_n = O(\log n)$.

Remark. The inequality $x > y$ holds if and only if the number $x+(2^n-1-y)$ is greater than $2^n - 1$, which can be checked by looking at the n-th bit of this number. Note that $2^n - 1 - y$ is computed very easily (by negating all bits of y), so the comparison can be reduced to addition (see Problem 2.12). However, the solution we gave is simpler.

2.10. a) Let $j = \overline{j_{l-1} \cdots j_0}$. We will gradually narrow the table by taking into account the values of j_{l-1}, j_{l-2}, and so on. For example, if $j_{l-1} = 0$, we select the first half of the table; otherwise we select the second half. It is clear that the choice is made between $x_{\overline{0j_{l-2}\cdots j_0}}$ and $x_{\overline{1j_{l-2}\cdots j_0}}$ for each combination of $j_{l-2}, \ldots, j_0$. Such choices are described by the function

$$f(a,b,c) = \begin{cases} b & \text{if } a = 0, \\ c & \text{if } a = 1, \end{cases}$$

which is applied simultaneously to all pairs of table entries. The operation is then repeated with the resulting table, so f is used $2^{l-1}+2^{l-2}+\cdots+1 = O(2^l)$ times in l parallel steps. Note that the function f can be computed by a circuit of size $O(1)$.

However, before we can actually apply f multiple times, we need to prepare 2^p copies of j_p for each p (recall the bounded fan-out condition). This requires $O(2^l)$ trivial gates arranged in $O(l)$ layers.

b) The solution is very similar to that of Problem 2.9. Let us construct a circuit Search_n that outputs $l = \log_2 n$ copies of $y = x_0 \vee \cdots \vee x_{n-1}$, as well as the smallest j such that $x_j = 1$ (if such j exists). The circuit Search_n can be obtained from two copies of $\mathit{Search}_{n/2}$ applied to the first and the second half of the string x, respectively. Let the results of these application be $y', \ldots, y', j'$ and $y'', \ldots, y'', j''$. Then we make one additional copy of y' and y'' and compute

$$y = y' \vee y'', \qquad j = \begin{cases} j' & \text{if } y' = 1, \\ n/2 + j'' & \text{if } y' = 0 \end{cases}$$

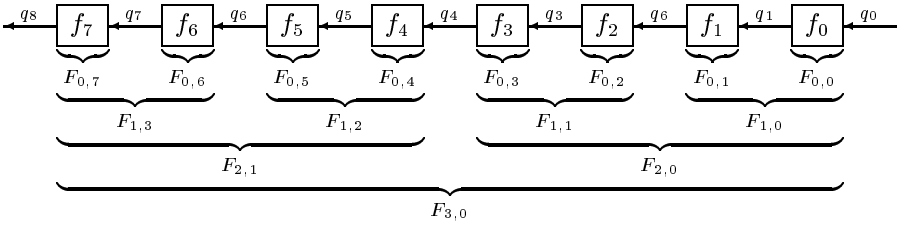
by a circuit of size $O(l)$ and depth $O(1)$ (each copy of y' controls a single bit of j).

2.11. Let $\big(f_j(q),\, g_j(q)\big) \stackrel{\text{def}}{=} D(q, x_j)$. Then the intermediate states of the automaton and its output symbols are

$$q_{j+1} = f_j f_{j-1} \cdots f_0\,(q_0) \quad \text{(composition of functions)}, \qquad y_j = g_j(q_j).$$

The solution of the problem is divided into 4 stages.

1. We tabulate the functions f_j and g_j (by making m copies of the table of D and narrowing the j-th copy according to x_j; see the solution to Problem 2.10a). This is done by a circuit of size $\exp(O(k))\, m$ and depth $O(k + \log m)$.
2. We compute a certain composition of the functions $f_0, \ldots, f_{m-1}$ (see diagram and text below).
3. We compute q_j in a parallel fashion (see below).
4. We apply the functions g_j; this is done by a circuit of size $\exp(O(k))\, m$ and depth $O(k)$.

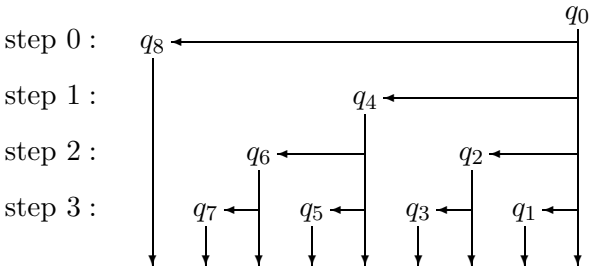


At stages 2 and 3 we assume that $m = 2^l$ (since we can always augment the sequence $f_0, \dots, f_{m-1}$ by identity functions). We organize the functions f_p into a binary tree and compute their compositions $F_{r,p}$ (the case $l = 3$ is shown in the diagram). First we define $F_{0,p} = f_p$. At step 1 we compute the compositions $F_{1,0} = f_1 f_0$ through $F_{1,\,m/2-1} = f_{m-1} f_{m-2}$; we continue this process for l steps until we get the function $F_{l,0} = f_{m-1} \cdots f_0$. The general formula for step r is as follows:

$$F_{r,p} = F_{r-1,\,2p+1} F_{r-1,\,2p}, \qquad r = 1, \dots, l, \quad 0 \le p < 2^{l-r}.$$

In this computation functions are represented by value tables. Composing two functions, say u and v, amounts to computing $u(v(q))$ for all values of q; this is done by a circuit of size $\exp(O(k))$ and depth $O(k)$.

With the functions $F_{r,p}$, the transition from q_0 to q_j becomes much quicker. For example, if j lies between 2^{l-1} and 2^l, we can get to $q_{2^{l-1}}$ in one leap; then we make smaller jumps until we stop at the right place. Doing the same thing for all j, we compute q_j in the following order (for $l = 3$):



In general, the computation consists of $l + 1$ steps. At step s we obtain the values of q_j for every j of the form $j = 2^{l-s}(2p + 1)$, using the recurrence relation

$$q_{2^{l-s}(2p+1)} = F_{l-s,\,2p}\big(q_{2^{l-s+1}p}\big), \qquad s = 0, \dots, l, \quad 0 \le 2p + 1 \le 2^s.$$

The computation of $F_{r,p}$ and q_j is performed by a $2^l \exp(O(k))$-size, $O(lk)$-depth circuit. (Note that each q_j is used only once at each of the following steps, $s+1, \dots, l$, so that we can make copies as we need them, while keeping the fan-out bounded. Therefore, the bounded fan-out condition can be satisfied without increase in depth.)

2.12. Suppose we want to compute $z = x + y$. Let $x = \overline{x_{n-1} \cdots x_0}$, $y = \overline{y_{n-1} \cdots y_0}$, $z = \overline{z_n \cdots x_0}$. The standard addition algorithm is described by the formulas

$$q_0 := 0, \qquad q_{j+1} := \begin{cases} 0 & \text{if } x_j + y_j + q_j < 2 \\ 1 & \text{if } x_j + y_j + q_j \ge 2 \end{cases} \qquad (j = 0, \dots, n-1),$$

$$z_j := x_j \oplus y_j \oplus q_j, \qquad z_n := q_n,$$

where $q_0, \ldots, q_n$ are the carry bits. This sequence of assignments is a circuit of size and depth $O(n)$. Note that it also corresponds to a finite-state automaton with the input alphabet $A' = \mathbb{B}^2$ (pairs of input bits), the output alphabet $A'' = \mathbb{B}$ (bits of the result) and the state set $\mathbb{B}$ (the value of the carry bit). Hence the result of Problem 2.11 applies.

2.13. Part b) follows from Part a), so it is enough to solve a). The first idea is obvious — we need to organize the input numbers into a binary tree of depth $\lceil \log_2 m \rceil$. Addition of two numbers can be done with depth $O(\log n)$, hence we get a circuit of depth $O(\log m \log n)$. However, this is not exactly what we want.

Fortunately, addition of two numbers can be done with depth $O(1)$ if we use a different encoding for the numbers. Specifically, let us represent a number *by the sum of two numbers* (obviously, such a representation is not unique).

Lemma. *There exists a circuit of size $O(n)$ and depth $O(1)$ that converts the sum of three n-digit numbers into the sum of two numbers, i.e., computes a function*

$$F : (x, y, z) \mapsto (u, v) \quad \textit{such that } u + v = x + y + z \textit{ for any } x, y, z.$$

(We will find a particular F such that u has $n+1$ digits, whereas v has n digits. If x has $n+1$ digits instead of n, then both u and v will be $n+1$-digit numbers.)

Proof. Let $x = \overline{x_n x_{n-1} \cdots x_0}$, $y = \overline{y_{n-1} \cdots y_0}$, $z = \overline{z_{n-1} \cdots z_0}$. We can perform the addition bitwise, *without carrying bits between binary places.* Thus at each place j we get the number $w_j = x_j + y_j + z_j \in \{0, 1, 2, 3\}$ (for $j = 0, \ldots, n-1$). This number can be represented by two digits: $w = \overline{u_j v_j}$. Then we put

$$u = \overline{u_{n-1} \cdots u_0 0}, \qquad v = \overline{x_n v_{n-1} \cdots v_0}.$$

□

Now let (x_1, x_2) and (y_1, y_2) be pairs of$n$-digit numbers that represent $x = x_1 + x_2$ and $y = y_1 + y_2$. Applying the lemma twice, we get a pair of $n+1$-digit numbers, (z_1, z_2), such that $z_1 + z_2 = x + y$. Therefore we can build a circuit of size $O(nm)$ and depth $O(\log m)$ that adds m n-digit numbers represented by pairs. This way, we obtain two n'-digit numbers, where $n' = n + \lceil \log_2 m \rceil$. At the end, we need to actually add these numbers so that the result appear in the usual form. This can be done by a circuit of size $O(n')$ and depth $O(\log n')$ (see Problem 2.12).

2.14. The standard division algorithm can be described as a sequence on n subtractions, each of which is skipped under certain conditions. This

corresponds to a circuit of size $O(n^2)$ and depth $O(n \log n)$ (assuming that each subtraction is performed in a parallel fashion, as in Problem 2.12). Unfortunately, this algorithm cannot be parallelized further, not even with tricks like in the previous problem. So, we will use a completely different method which allows parallelization at the cost of some increase in circuit size (by a factor of order $O(\log n)$).

a) Let $y = 1-x/2$ (it can be represented as $\overline{0.0y_1y_2\cdots}$, where $y_j = 1-x_j$). Note that $0 \le y \le 1/2$, so we can express x^{-1} by a rapidly convergent series:

$$x^{-1} = \frac{1}{2}(1-y)^{-1} = \frac{1}{2}\left(\sum_{k=0}^{m-1} y^k + r_m\right), \quad \text{where } 0 \le r_m = \frac{y^m}{1-y} \le 2^{-(m-1)}.$$

We may set $m = 2^l$, $l = \lceil \log_2(n+2) \rceil$, so that $r_m \le 2^{-(n+1)}$ and

$$\sum_{k=0}^{m-1} y^k = (1+y)(1+y^2)(1+y^4)\cdots(1+y^{2^{l-1}});$$

let us denote the last expression by u_m.

We need to compute $x^{-1} = \frac{1}{2}(u_m + r_m)$ with precision 2^{-n}. By neglecting r_m, we introduce an error $\le 2^{-(n+2)}$ in the result. An additional error, bounded by $2^{-(n+1)}$, comes from the inaccurate knowledge of x. Therefore, it suffices to compute u_m with precision $2^{-(n+1)}$. This calculation involves $O(l) = O(\log n)$ multiplications and additions. In doing it, we must take into account that round-off errors may accumulate; therefore we need to keep $n + \Theta(\log n)$ digits. Each multiplication or addition can be done by a circuit of size $O(n^2)$ and depth $O(\log n)$, hence the total size and depth are $O(n^2 \log n)$ and $O((\log n)^2)$, respectively.

b) First, we find an integer s such that $2^s \le b < 2^{s+1}$ (see Problem 2.10b).

The next step is to find an approximate value of $\lfloor a/b \rfloor$, which is done by a circuit of size $O(k^2 \log k)$ and depth $O((\log k)^2)$. We set $x = 2^{-s}b$ and compute x^{-1} with precision 2^{-k-3} by the circuit described above. (The computed value, as well as the exact value, does not exceed 1.) Similarly, we approximate the number $y = 2^{-s}a < 2^{k+1}$ with precision 2^{-2}. Now we can calculate $a/b = yx^{-1}$ with the overall error $< 1/2$ and replace the result by the closest integer q.

Let $r = a - qb$. It is clear that either $q = \lfloor a/b \rfloor$ and $r = (a \bmod b)$, or $q = \lfloor a/b \rfloor + 1$ and $r = (a \bmod b) - b$. We can determine which possibility has realized by checking if r is negative. If it is, we replace q by $q-1$, and r by $r+b$.

2.15. To compute the function MAJ we count 1s among the inputs, i.e., compute the sum of the inputs. This can be done by the circuit from Problem 2.13a. Then we compare the result with $\lceil n/2 \rceil$ (see Problem 2.9).

2.16. We begin with some elementary graph theory. Graphs can be described by their *adjacency matrices*. Rows and columns of the adjacency matrix $A(G)$ of a graph G are indexed by the vertices of the graph. If (j,k) is an edge of G, then $a_{jk} = 1$; otherwise $a_{jk} = 0$. We regard the matrix elements as Boolean variables.

We can define the operations $\vee$ and $\wedge$ on Boolean matrices by analogy with the usual matrix addition and multiplication:

$$(P \vee Q)_{uv} = P_{uv} \vee Q_{uv},$$
$$(P \wedge Q)_{uv} = \bigvee_w (P_{uw} \wedge Q_{wv}).$$

Then, P^k is a short notation for $P \wedge \cdots \wedge P$ (k times).

What is the meaning of the matrix A^k, where $A = A(G)$ is the adjacency matrix of a graph G? Each element of this matrix, $(A^k)_{uv}$, says whether there is a path of length k (i.e., a chain of k edges) between the vertices u and v. Similarly, each element of the matrix $(A \vee I)^k$, where I is the "identity matrix" ($I_{uv} = \delta_{uv}$), corresponds to the existence of a path of length *at most* k. Note that if there is a path between u and v, then there is a path of length $\leq n$, where n is the number of vertices. Therefore, to solve the problem, it suffice to compute B^k, where $B = A \vee I$ and $k \geq n$.

Multiplication of $(n \times n)$-matrices (i.e., the operation $\wedge$) can be performed by a circuit of depth $O(\log n)$ and size $O(n^3)$. Let $l = \lceil \log_2 n \rceil$, $k = 2^l$. All we need is to compute $B^2, B^4, \ldots, B^{2^l}$ by repeated squaring. This can be done by a circuit of size $O(n^3 \log n)$ and depth $O((\log n)^2)$.

2.17. As with many problems, a detailed solution of this one is tedious (we would have to choose a particular encoding for circuits in the first place), but the idea is rather simple. To begin with, we describe an algorithm for evaluating a formula of depth d. Then we will extend that algorithm to circuits of depth d with one output variable (the size of such a circuit is $\exp(O(d))$). The generalization to several output variables is trivial.

The algorithm is simply an implementation of recursion on a Turing machine. Suppose we need to compute a subformula $A = f(A_0, A_1)$, where f denotes an arbitrary gate from the basis. (The specific choice of f is determined by an oracle query.) Let us assume that the computation of A_0 and A_1 can be done with space s. We compute A_0 first and keep the result (which is a single bit), while freeing the rest of the space used in the computation. Then we compute A_1. Finally, we find A and free the

space occupied by the value of A_0. Thus the computation of A requires only $s + O(1)$ bits. Likewise, each level of recursion requires a constant number of bits, hence $O(d)$ bits will suffice for the d levels.

Now let C be a circuit of depth d with one output variable. According to Problem 2.5, such a circuit can be "expanded" into a formula F of the same depth. Specifically, subformulas of F are in one-to-one correspondence with paths from the output variable to nodes (i.e., input and auxiliary variables) of the circuit C. Note that we do not have enough space to hold the expansion. Instead, we need to look up for a path in the circuit C each time we would access a subformula in F according to the original algorithm. Thus the algorithm for circuits involves traversing all paths. Note that space $O(d)$ is sufficient to hold a description of a path. We also need to allocate additional $O(d)$ bits to build an oracle query on this description.

2.18. The idea is to consider the machine M running in space s as an automaton with $\exp(O(s))$ states. This is not quite straightforward, because the machine actively reads bits from random places rather than just receiving a sequence of symbols. However, the difference is not so dramatic. We can arrange that the automaton repeatedly receives the same input string, $xxx\cdots$, each time waiting for a needed bit to come and skipping the others.

Let V be the set of configurations of the machine M with space s. Denote by v_k the initial configuration when the machine is asked to compute the k-th bit of $f_{n,m,s}(x)$. (Note that v_k does not depend on x.) We will construct an automaton with the input and output alphabet $A' = A'' = \mathbb{B}$ and the state set

$$Q = \{0, \dots, m-1\} \times V \times \{0, \dots, |V|-1\} \times \{0, \dots, n-1\},$$

elements of which are denoted by (k, v, t, j). The initial state is $q_0 = (0, v_0, 0, 0)$. What follows is a description of the automaton transition function.

The variable j counts bits of the input string; each time it is incremented by one. Whenever j matches the contents of the machine supplementary tape (which is a part of v), the current input bit is accepted as the answer from the oracle; otherwise it is ignored. When j "turns over", i.e., changes from $n-1$ to 0, the Turing machine clock ticks, meaning that t gets incremented and the machine configuration v changes according to its transition function. Finally, whenever t turns over, the output bit is set to the computation result (contained in v), k gets incremented, and the machine configuration is set to v_{k+1}.

Let x be a binary string of length n. If we feed the string $x^{m|V|} = xx\cdots x$ ($m|V|$ times) to the automaton, and select every l-th bit of the output ($l = |V|n$), then we obtain the value of the desired function, $y = f_{n,m,s}(x)$.

Therefore we can apply the result of Problem 2.11. Our automaton has $\exp(O(s))$ states and receives $m|V|n = \exp(O(s))$ symbols, hence it can be simulated by a circuit of size $\exp(O(s))$ and depth $O(s^2)$.

2.19. The solution consists of the following steps.

1. We transform C into an equivalent formula Γ which operates with elements of a fixed finite group G rather than 0's and 1's. The basis of Γ consists of the group operations, i.e., MULTIPLICATION and INVERSION. The Boolean value 0 is represented by the unit element e of the group, whereas 1 is represented by an element $u \neq e$. More formally, let $\varphi : \mathbb{B} \to G$ be the function defined by $\varphi(0) = e,\ \varphi(1) = u$. Then Γ computes a function $F(g_1, \ldots, g_N)$ $(N = \exp(O(d)))$ such that
$$\varphi\big(f(x_1, \ldots, x_n)\big) = F(g_1, \ldots, g_N), \qquad \text{where } g_j = \varphi(x_{p_j}) \text{ or } g_j = \text{const}.$$
Each variable g_j is used in the formula Γ only once.
2. Using the identity $(ab)^{-1} = b^{-1}a^{-1}$, we transform Γ to a form in which the inversion is applied only to input variables. Due to the associativity of multiplication, the tree structure of the formula does not matter. Thus we arrive at the equation
$$\varphi\big(f(x_1, \ldots, x_n)\big) = h_1 \cdots h_n,$$
$$\text{where } h_j = \varphi(x_{t_j}) \text{ or } h_j = \varphi(x_{t_j})^{-1} \text{ or } h_j = \text{const}.$$
3. The product of group elements $h_1 \cdots h_N$ is computed by a finite-state automaton with $|G| = O(1)$ states.
4. The work of this automaton is simulated by a circuit of width $O(1)$ and size $O(N) = \exp(O(d))$.

Steps 2, 3 and 4 are rather straightforward, so we only explain the first step.

Let $G = A_5$ be the group of even permutations on 5 elements; it consists of $5!/2 = 60$ elements. (A smaller group will not suffice: our construction works only for unsolvable groups.) We will use the standard cycle notation for permutations, e.g., $(245) :\ 2 \mapsto 4,\ 4 \mapsto 5,\ 5 \mapsto 2,\ 1 \mapsto 1,\ 3 \mapsto 3$.

Lemma. *There are elements $u, v, w \in A_5$ that are conjugate to each other (i.e., $v = aua^{-1},\ w = bub^{-1}$) and $w = uvu^{-1}v^{-1}$.*

Proof. The conditions of the lemma are satisfied by
$$u = (12345), \quad v = (13542), \quad w = (14352), \quad a = (235), \quad b = (245).$$
□

We will assume that the original formula C is written in the basis $\{\neg, \wedge\}$. Let us set $\varphi(0) = e,\ \varphi(1) = u$ and use the following relations:
$$\varphi(\neg x) = u\,\varphi(x)^{-1}, \qquad \varphi(x \wedge y) = b^{-1}\varphi(x)\,a\,\varphi(y)\,a^{-1}\varphi(x)^{-1}a\,\varphi(y)^{-1}a^{-1}b.$$

Thus any Boolean formula of depth d is transformed into a formula over A_5 with $N \le N(d)$ once-only variables, where $N(d)$ satisfies the following recurrence relation:

$$N(d+1) = \max\bigl\{N(d)+1,\ 4N(d)+6\bigr\}.$$

From this we get $N(d) = \exp(O(d))$.

S3. Problems of Section 3

3.1. The key to the solution is the following property of Boolean functions: $A \vee \neg B$ and $B \vee C$ imply $A \vee C$ (i.e., if the first two expressions are true for given values of A, B, C, then the third one is also true). Applying this property, called the *resolution rule*, we can derive new disjunctions from old ones. This process terminates at some set $\overline{F}$ of disjunctions that is not extendible any more; we call it the *closure* of the original CNF F. By construction, F and $\overline{F}$ represent the same Boolean function. The presence of the empty disjunction in $\overline{F}$ indicates that this function is identical to 0, i.e., F is not satisfiable. This way of checking the satisfiability is called the *resolution method*. It has polynomial running time for 2-CNFs and exponential running time in general.

Let us describe the resolution method in more detail. Recall that a CNF is a formula of the form $F = D_1 \wedge \cdots \wedge D_m$, where $D_1, \ldots, D_m$ are clauses, i.e., disjunctions of literals. A literal is either a variable or its negation. To be precise, each clause is represented in the standard form, which is a set of literals not containing x_j and $\neg x_j$ at the same time. The idea is that a disjunction like $x_1 \vee x_1$ is reduced to x_1, whereas $x_1 \vee \neg x_1 \vee x_2$ (which is equal to 1) is removed from the list of clauses. The empty set is identified with the logical constant 0. We will regard F as a set of clauses (i.e., each clause enters F at most once, and the order does not matter).

Suppose F contains clauses $A \vee \neg x_j$ and $C \vee x_j$ for some variable x_j. Let $A \vee C \ne 1$. Then we can reduce $D = A \vee C$ to the standard form described above. The resolution rule takes F to $F' = F \cup \{D\}$. The repeated application of this rule takes F to its closure $\overline{F}$. Note that applying the resolution rule to 2-clauses and 1-clauses, one can only get 2-clauses, 1-clauses, or the empty clause. (But if some clauses contain 3 or more literals, the size can grow even further.)

Theorem. *F is not satisfiable if and only if $\overline{F}$ contains the empty clause.*

Proof. We will prove a more general statement. Let $Y = l_1 \vee \cdots \vee l_k$ be an arbitrary clause. Then *F implies Y if and only if $\overline{F}$ contains some clause D that implies Y (i.e., $D \subseteq Y$)*. The theorem corresponds to the $Y = 0$ case of this assertion.

The "if" part is obvious since F implies every clause in $\overline{F}$.

To prove the "only if" part, we will use induction on k, going from $k = n$ (the total number of variables) down to 0. If $k = n$, then the function $Y(x)$ takes value 0 at exactly one point $x = x_*$. The condition "F implies Y" means that $F(x_*) = 0$. Therefore $D(x_*) = 0$ for some $D \in F \subseteq \overline{F}$.

If $k < n$, let x_j be a variable that does not enter Y. By the induction hypothesis, the condition "F implies Y" means that there are some $D_1, D_2 \in \overline{F}$ such that D_1 implies $Y_1 = Y \vee x_j$, and D_2 implies $Y_2 = Y \vee \neg x_j$. If we regard clauses as sets of literals, this condition becomes $D_1 \subseteq Y_1,\ D_2 \subseteq Y_2$. If $x_j \notin D_1$ or $\neg x_j \notin D_2$, then $D_1 \subseteq Y$ or $D_2 \subseteq Y$, so we are done. Otherwise we apply the resolution rule to D_1 and D_2 to obtain a new clause $D \subseteq Y$. □

The resolution method can be optimized for 2-CNFs. To any 2-CNF F we associate a directed graph $\Gamma(F)$. The vertices of $\Gamma(F)$ are literals. Each 2-clause $a \vee b$ is represented by two edges, $(\neg a, b)$ and $(\neg b, a)$. Each 1-clause a is represented by the edge $(\neg a, a)$. Let $\widetilde{F}$ be $\overline{F}$ without the empty clause. It is easy to see that $\Gamma(\widetilde{F})$ consists of all pairs of vertices that are connected by paths in $\Gamma(F)$. According to the above theorem, F is not satisfiable if and only if $\widetilde{F}$ contains clauses x_j and $\neg x_j$ for some j. This is equivalent to the condition that $\Gamma(F)$ contains paths from x_j to $\neg x_j$ and from $\neg x_j$ to x_j. Therefore, we can use the algorithm from Problem 2.16.

3.2. A necessary condition for the existence of an Euler cycle is the *connectivity* of the graph: any two vertices are connected by a path. Another necessary condition: each vertex has even degree. (The *degree* of a vertex in a graph is the number of edges incident to that vertex.) Indeed, if an Euler cycle visits some vertex k times, then degree of this vertex is $2k$.

Together these two conditions are sufficient: if a graph is connected and all vertices have even degrees, it has an Euler cycle. To prove this, let us start at any vertex and extend the path by adding edges that have not been used before. Since all vertices have even degrees, this extension process terminates only when we come to the starting point (i.e., get a cycle). If not all edges are used, we repeat the process and find another cycle (note the unused edges form a graph where each vertex has even degree), etc.

After that our connected graph is covered by several edge-disjoint cycles, and these cycles can be combined into an Euler cycle.

It remains to note that it is easy to compute the degrees of all vertices in polynomial time. One can also find out in polynomial time whether the graph is connected or not.

3.3. Let $F(x_1, \ldots, x_n)$ be a propositional formula. How do we find a satisfying assignment? First, we ask the oracle whether such an assignment exists. If the answer is "yes", we next learn whether there is a satisfying

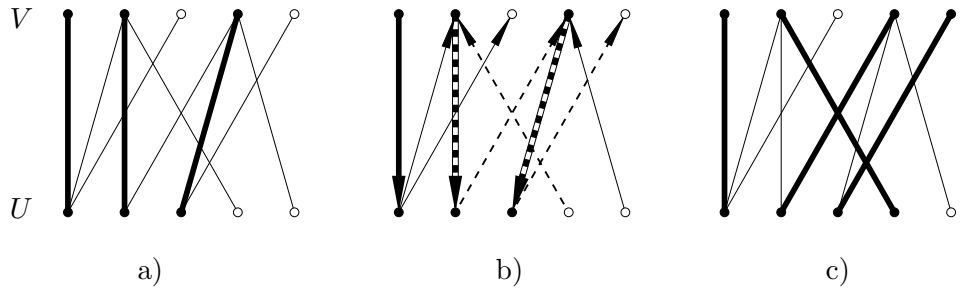


Fig. S3.1. Extending a partial matching by an alternating path: a) old matching; b) alternating path; c) new matching.

assignment with $x_1 = 0$. To this end, we submit the query $F' = F \wedge \neg x_1$. If the answer is "no" (but F is satisfiable), then there is a satisfying assignment with $x_1 = 1$. Then we try to fix the value of x_2 and so forth.

Similar arguments can be used for the HAMILTONIAN CYCLE problem (and other NP-problems).

3.4. We will describe a polynomial algorithm for a more general problem: given a bipartite graph Γ, find a maximal *partial matching*, i.e., a set of graph edges that do not share any vertex.

(A *bipartite graph* is a graph whose vertex set is divided into two parts, U and V, so that the edges connect only vertices from different parts. Thus the edge set is $E \subseteq U \times V$.)

We construct a maximal matching stepwise. At each step we have a set of edges $C \subseteq E$ that provides a one-to-one correspondence between $A \subset U$ and $B \subset V$. To extend C by one edge, i.e., to find a matching C' of size $|C| + 1$, we try to find a so-called alternating path (see Figure S3.1).

Definition. An *alternating path* is a sequence of consecutively connected vertices $x_0, \ldots, x_l$ such that

(1) all vertices in the sequence are distinct;

(2) edges from C and from $E \setminus C$ alternate;

(3) $x_0 \in U \setminus A, \quad x_l \in V \setminus B$.

Thus (x_0, x_1) and (x_{l-1}, x_l) belong to $E \setminus C$, and the path length l is odd.

Lemma. *The matching C can be extended if and only if an alternating path exists.*

Proof. If an alternating path exists, we can extend C by replacing the edges $(x_{2j-1}, x_{2j}) \in C$ with the edges $(x_{2j}, x_{2j+1}) \in E \setminus C$. There is one more edge of the second type than of the first type.

Conversely, suppose a larger matching C' exists. Let us superimpose C and C' to form the symmetric difference $X = C \oplus C' = (C \setminus C') \cup (C' \setminus C)$. Each vertex is incident to at most one edge from $C \setminus C'$ and at most one edge from $C' \setminus C$. Therefore the connected components of X are paths and cycles in which edges from $C \setminus C'$ and $C' \setminus C$ alternate. Since C' is larger than C, at least one of the components contains more edges from $C' \setminus C$ than from $C \setminus C'$. Such a component is an alternating path. □

Therefore, if there is no alternating path, then C is maximal, and the algorithm stops. It remains to explain how to find an alternating path when it exists.

Let us introduce direction on edges of the graph according to the following rule: the edges in C go from V to U, and all other edges go from U to V (as is shown in Figure S3.1b). Then the existence of an alternating path is equivalent to the existence of a pair of vertices $u \in U \setminus A$ and $v \in V \setminus B$ such that there is a directed path from u to v. (Indeed, we may assume that the path from u to v is simple, i.e., it does not visit the same vertex twice, and therefore is an alternating path.) The existence of a directed path between two vertex subsets can be checked by a slightly modified algorithm of Problem 2.16 (now the graph is directed, but this does not matter). The algorithm can also be extended to find the path.

Thus we have proved that the perfect matching problem belongs to P. (The algorithm described above is not optimal.)

3.5. It is sufficient to solve (b); however, we provide a (somewhat simpler) solution for (a) first.

The Clique problem becomes an Independent set problem if we replace the graph by its complement. (Definition of the terms: For each graph G the *complementary graph* $\overline{G}$ has the same vertices, whereas its edges are nonedges of G. An *independent set* is a set I of vertices such that no two vertices in I are joined by an edge.) Clearly, cliques in G are independent sets in $\overline{G}$, and vice versa.

For any 3-CNF F with n variables and m clauses we construct a graph H such that H has an independent set of cardinality m if and only if F is satisfiable. The vertices of H are the literals in C. (We assume that each clause is a disjunction of three literals; therefore H has $3m$ vertices.) Literals in one clause are joined by edges. (Therefore an independent set of cardinality m should include one literal from each clause.) Two literals from different clauses are joined by an edge if they are contradictory (i.e., the edge goes between x_j and $\neg x_j$).

If an independent set I of size m exists, it contains one literal from each clause, and no two literals are contradictory. Therefore we can assign

values to the variables so that all literals in I be true. This will be a satisfying assignment for the 3-CNF. Conversely, if the 3-CNF has a satisfying assignment, we choose a true literal from each clause to be a member of I.

This construction, however, does not solve (b), because several independent sets may correspond to the same satisfying assignment.

(b) To construct a reduction that preserves cardinality we must be more cautious. Assume that we have a 3-CNF with n variables and m clauses. For each variable x_i there are two vertices (called V-vertices) labeled x_i and $\neg x_i$ (see Figure S3.2a); exactly one of them will be in the independent set I of the required size. Informally, $x_i \in I$ [$\neg x_i \in I$] means that $x_i = 1$ [resp. $x_i = 0$] in the assignment.

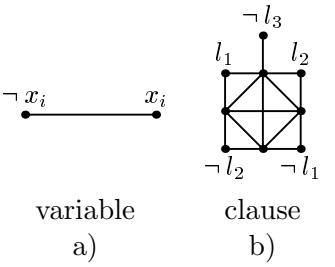


Fig. S3.2

In addition to these $2n$ vertices, we have $4m$ vertices called C-vertices (m is the number of clauses). Specifically, for each clause we have a group of 4 vertices connected by edges (so that an independent set can include only one vertex of the four). Figure S3.2b shows the C-vertices (unlabeled vertices that form a small square in the middle) for a generic clause $l_1 \vee l_2 \vee l_3$. Here l_1, l_2, l_3 are literals, i.e., variables or their negations. (If l_s is $\neg x_i$, then $\neg l_s$ denotes the vertex x_i.) The figure also shows edges between C-vertices and V-vertices (labeled by $l_1, l_2, \neg l_1, \neg l_2, \neg l_3$). Note that in the graph each V-vertex may be connected with several groups of C-vertices, as each variable may enter several clauses.

We will be looking for an independent set of size $n+m$. Such a set must include exactly one V-vertex (labeled x_i or $\neg x_i$) for each variable x_i and exactly one of the four C-vertices for each clause. Depending on whether x_i or $\neg x_i$ is included, we set $x_i = 1$ or $x_i = 0$. The choice of C-vertices is determined uniquely by this assignment. Indeed, let us look at Figure S3.2b, not paying attention to the V-vertex $\neg l_3$. It is easy to check that for each pair of values of the literals l_1 and l_2 there is exactly one C-vertex in the picture that can be included in the independent set. For example, if l_1 is true and l_2 is false, then the vertices l_1 and $\neg l_2$ are in the independent set I; therefore only the rightmost C-vertex can be in I.

The vertex $\neg l_3$ taken into account, the constructed set I may turn not to be independent. This happens when both $\neg l_3$ and the top C-vertex (the one between l_1 and l_2) belong to I. Then we have $l_1 = l_2 = l_3 = 0$. But this is exactly the case where the clause $l_1 \vee l_2 \vee l_3$ is false.

Therefore, an independent set of size $m+n$ in the graph exists if and only if there is a satisfying assignment for the given 3-CNF. The preceding argument shows that the correspondence between independent sets and satisfying assignments is one-to-one.

3.6. (a) See solution for (b) (though in fact (a) has a simpler solution that can be found, e.g., in [**67**]).

(b) Note that the number of 3-colorings is a multiple of 6 (we can permute colors).

Consider a 3-CNF C in n variables with m clauses. We construct a graph G that has $7m+2n+3$ vertices and admits 3-coloring if and only if C is satisfiable. Moreover, the number of 3-colorings (up to permutations, i.e., divided by 6) equals the number of satisfying assignments.

Three vertices of G (called $0, 1, 2$ in the sequel) are connected to each other. In any 3-coloring they have different colors; we assume that they have colors $0, 1, 2$ (see Figure S3.3a); this requirement reduces the number of colorings by a factor $1/6$.

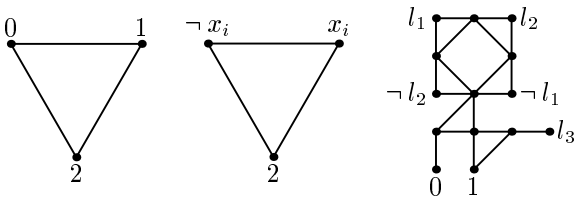


Fig. S3.3

For each of the n variables x_i we have two vertices, which are connected to each other and to the vertex 2 (see Figure S3.3b, where the vertex 2 is shown but the vertices 0 and 1 are not). These two vertices are labeled x_i and $\neg x_i$. In a 3-coloring they will have either colors 1 and 0 ("x_i is true") or 0 and 1 ("x_i is false").

If no other vertices and edges are added, the graph has 2^n 3-colorings that correspond to all possible assignments. We now add some "gadgets" that correspond to clauses of C. For a given assignment each gadget either has no colorings (if the clause is false for that assignment) or has exactly one coloring.

Figure S3.3c shows such a gadget that corresponds to the clause $l_1 \vee l_2 \vee l_3$, where l_1, l_2, l_3 are literals (i.e., variables or their negations).

One can check that the required property (no colorings or unique coloring, depending on $l_1 \vee l_2 \vee l_3$) is indeed true.

3.7. The tiling problem belongs to NP (if Merlin shows a correct tiling, then Arthur can check its correctness).

Let us show that 3-SAT can be reduced to tiling. For any 3-CNF C we need to construct an instance of the tiling problem (i.e., select tile types and boundary conditions) that has a solution if and only if C is satisfiable.

Each tile type will be usable in only one place of the square (this restriction can be enforced by appropriate labeling); for each position in the square we will have several types that are usable in this position. Each position can be regarded as a device that receives information from neighboring devices, processes it, and sends it further.

Tiles in the bottom row correspond to the variables of C. For each of them we have two possibilities that correspond to values 0 and 1; these values are propagated to the top unchanged. The other rows are used to compute the values of clauses, passing intermediate results from left to right. One additional column on the right side is needed to compute the value of C (the result appears in the upper right corner). Finally, we add a condition asserting that $C = 1$.

Another possibility is to reduce an arbitrary NP-problem to tiling directly: the computation table of a TM is a two-dimensional table that satisfies some local rules that can be interpreted as tiling rules.

3.8. Merlin tells Arthur the prime factors of x (each factor may repeat several times). Since the multiplication of integers can be performed in polynomial time, Arthur can check whether or not the factorization provided by Merlin is correct.

3.9. We prove that PRIMALITY $\in$ NP by showing how Merlin constructs a polynomial size "certificate" of primality that Arthur can verify in polynomial time.

Let $p > 2$ be a prime number. It is enough to convince Arthur that $(\mathbb{Z}/p\mathbb{Z})^*$ is a cyclic group of order $p-1$ (see Appendix A, especially Theorem A.10). Merlin can show a generator g of this group, and Arthur can verify whether $g^{p-1} \equiv 1 \pmod{p}$ (this requires $O(\log p)$ multiplications; see Section 4.2.2).

This is still not enough since the order of g may be a nontrivial divisor of $p-1$. If for some reason Arthur knows the factorization of $p-1$ (which includes prime factors $q_1, q_2, \dots$), he can check whether $g^{(p-1)/q_j} \not\equiv 1 \pmod{p}$. But factoring is a difficult problem, so Arthur cannot compute the numbers q_j himself.

However, Merlin may communicate the factorization to Arthur. The only problem is that Merlin has to convince Arthur that the factors are

indeed prime. Therefore Merlin should recursively provide certificates of primality for all factors.

Thus the complete certificate of primality is a tree. Each node of this tree is meant to certify that some number q is prime (the root corresponding to $q = p$). All leaves are labeled by $q = 2$. Each of the remaining nodes is labeled by a prime number $q > 2$ and also carries a generator h of the group $(\mathbb{Z}/q\mathbb{Z})^*$; the children of this node are labeled by prime factors of $q - 1$.

Let us estimate the total size of the certificate. The tree we have just described has at most $n = \lceil \log_2 p \rceil$ leaves (since the product of all factors of $q - 1$ is less than q). The total number of nodes is at most twice the number of leaves (this is true for any tree). Each node carries a pair of n-bit numbers (q and h). Therefore the total number of bits in the certificate is $O(n^2)$.

Now we estimate the certificate verification complexity. For each of $O(n)$ nodes Arthur checks whether $h^{p-1} \equiv 1 \pmod q$. This requires $O(\log q) = O(n)$ multiplications, which is done by a circuit of size $O(n^3)$. Similar checks are performed for each parent-child pair, but the number of such pairs is also $O(n)$. Therefore the certificate check is done by a circuit of size $O(n^4)$, which can be constructed and simulated on a TM in time poly(n).

S5. Problems of Section 5

5.1. If there is an accepting (i.e., ending with "yes") computational path, then there is such a path of length $\exp(O(s))$. Indeed, we may think of machine configurations as points, and possible transitions as edges. Thus an NTM with space s is described by a directed graph with $N = \exp(O(s))$ vertices, and the an accepting path is just a path between two given vertices. If there is such a path, we can eliminate loops from it and get a path of length $\leq N$. Therefore the proof of Theorem 5.2 is still valid for nondeterministic Turing machines.

Thus we reduce the NTM to a polynomial game; then we simulate this game on a deterministic machine (see the first part of Theorem 5.2).

5.2. Negations of predicates from Σ_k belong to Π_k and vice versa, so that $P^{\Sigma_k} = P^{\Pi_k}$.

Similarly to P, the class P^{Σ_k} is closed under negations. Therefore it remains to prove the inclusion $P^{\Sigma_k} \subseteq \Sigma_{k+1}$.

Consider a predicate $F \in P^{\Sigma_k}$ and a polynomial algorithm A that computes it using an oracle $G \in \Sigma_k$. Then $F(x)$ is true if and only if there exists a (polynomial size) sequence σ of pairs (query to an oracle, answer bit) such that (1) it forces A to produce the answer 1; (2) for each pair $(x, 1) \in \sigma$ the

string x is indeed in G; (3) for each pair $(x,0)\in\sigma$ the string x does not belong to G.

Condition (1) has the Σ_1-form, condition (2) belongs to Σ_k, and condition (3) belongs to Π_k. Standard rules for quantifiers say that any predicate of the type

$$\exists x[\Sigma_1(x)\wedge\Sigma_k(x)\wedge\Pi_k(x)]$$

belongs to Σ_{k+1}.

Note that we have used the following property of classes Σ_k and Π_k: if a predicate G belongs to Σ_k/Π_k, then "any element of a finite sequence $z=\langle z_1,\dots,z_n\rangle$ belongs to G" is a Σ_k/Π_k-property of z. (Indeed, polynomially many games can be played in parallel.)

S6. Problems of Section 6

6.1. Let $\mathcal{F}$ be an arbitrary space, and $F:\mathcal{L}\times\mathcal{M}\to\mathcal{F}$ a bilinear function. The bilinearity implies that $F(u,v)=\sum_{j,k}u_jv_k\,F(e_j,f_k)$ for any $u=\sum_j u_je_j$ and $v=\sum_j v_jf_j$. If we set

$$G\left(\sum_{j,k}w_{jk}\,e_j\otimes f_k\right)=\sum_{j,k}w_{jk}\,F(e_j,f_k),$$

then the equation $G(u\otimes v)=F(u,v)$ will hold. Conversely, if $G':\mathcal{L}\otimes\mathcal{M}\to\mathcal{F}$ is a linear function satisfying $G'(u\otimes v)=F(u,v)$, then $G'(e_j\otimes f_k)=F(e_j,f_k)$, hence $G'=G$ by linearity.

6.2. Let $\mathcal{F}=\mathcal{L}'\otimes\mathcal{M}'$ and

$$F:\mathcal{L}\times\mathcal{M}\to\mathcal{F},\qquad F(u,v)=A(u)\otimes B(v).$$

Then the required map C is exactly the function G from the universality property.

6.3. The abstract tensor product is unique in the following sense. Suppose that two pairs, $(\mathcal{N}_1,H_1)$ and $(\mathcal{N}_2,H_2)$, satisfy the universality property. Then there is a unique linear map $G_{21}:\mathcal{N}_1\to\mathcal{N}_2$ such that $G_{21}(H_1(u,v))=H_2(u,v)$ for any u and v. This map is an isomorphism of linear spaces.

The existence and uniqueness of G_{21} follows from the universality of the pair $(\mathcal{N}_1,H_1)$: we set $\mathcal{F}=\mathcal{N}_2$, $F=H_2$, and get $G_{21}=G$.

Note that if $(\mathcal{N}_3,H_3)$ also satisfies the universality property, then $G_{31}=G_{32}G_{21}$ (the composition of maps). Therefore $G_{12}G_{21}=G_{11}$ and $G_{21}G_{12}=G_{22}$. But G_{11} and G_{22} are the identity maps on $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. Thus G_{12} and G_{21} are mutually inverse isomorphisms.

6.4. Let λ_j^2 ($\lambda_j > 0$) and $|\nu_j\rangle$ be the nonzero eigenvalues and the corresponding (orthonormal) eigenvectors of $X^\dagger X$. Then the vectors $|\xi_j\rangle = \lambda_j^{-1} X|\nu_j\rangle$ also form an orthonormal system. Thus we have

$$X|\nu_j\rangle = \lambda_j|\xi_j\rangle, \qquad X|\psi\rangle = 0 \text{ if } \langle\psi|\nu_j\rangle = 0 \text{ for all } j.$$

This implies (6.2). Finally, we check that $(XX^\dagger)|\xi_j\rangle = \lambda_j|\xi_j\rangle$.

6.5.

$$H[2] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix},$$

$$U[3,1] = \begin{pmatrix} u_{00,00} & u_{00,10} & 0 & 0 & u_{00,01} & u_{00,11} & 0 & 0 \\ u_{10,00} & u_{10,10} & 0 & 0 & u_{10,01} & u_{10,11} & 0 & 0 \\ 0 & 0 & u_{00,00} & u_{00,10} & 0 & 0 & u_{00,01} & u_{00,11} \\ 0 & 0 & u_{10,00} & u_{10,10} & 0 & 0 & u_{10,01} & u_{10,11} \\ u_{01,00} & u_{01,10} & 0 & 0 & u_{01,01} & u_{01,11} & 0 & 0 \\ u_{11,00} & u_{11,10} & 0 & 0 & u_{11,01} & u_{11,11} & 0 & 0 \\ 0 & 0 & u_{01,00} & u_{01,10} & 0 & 0 & u_{01,01} & u_{01,11} \\ 0 & 0 & u_{11,00} & u_{11,10} & 0 & 0 & u_{11,01} & u_{11,11} \end{pmatrix}.$$

S7. Problems of Section 7

7.1. Since the conjunction $\wedge$ and the negation $\neg$ form a complete basis for Boolean circuits, Lemmas 7.1, 7.2 show that it is sufficient to realize the functions $\wedge_\oplus$ (i.e., the Toffoli gate), $\neg_\oplus : (x, y) \mapsto (x,\, x \oplus y \oplus 1)$ and $\bigoplus$. But the Toffoli gate is already in the basis, $\neg_\oplus[1,2] = \neg[2] \bigoplus [1,2]$, so it suffices to realize $\bigoplus$. Let us introduce an auxiliary bit u initialized by 0. Then the action of $\bigoplus[1,2]$ can be represented as $\neg[u] \wedge_\oplus [u,1,2]\, \neg[u]$.

S8. Problems of Section 8

8.1. Any rotation in three-dimensional space can be represented as a composition of three rotations: through an angle α about the z axis, then through an angle β about the x axis, and then through an angle γ about

the z axis. Therefore any operator acting on one qubit can be represented in the form

$$U = e^{i\varphi} e^{i(\gamma/2)\sigma^z} e^{i(\beta/2)\sigma^x} e^{i(\alpha/2)\sigma^z}. \tag{S8.1}$$

Each of the operators on the right-hand side of (S8.1) can be expressed in terms of H and a controlled phase shift:

$$e^{i\varphi} = \Lambda(e^{i\varphi})\sigma^x \Lambda(e^{i\varphi})\sigma^x, \qquad e^{i\varphi\sigma^z} = \Lambda(e^{-i\varphi})\sigma^x \Lambda(e^{i\varphi})\sigma^x,$$
$$\sigma^x = H\Lambda(e^{i\pi})H, \qquad e^{i\varphi\sigma^x} = He^{i\varphi\sigma^z}H.$$

8.2. Let $U = e^{i\varphi}Z$, where $Z \in \mathbf{SU}(2)$. Then $\Lambda(U) = \Lambda(e^{i\varphi})\Lambda(Z)$. The operator $\Lambda(e^{i\varphi})$ acts only on the control qubit, so it remains to realize $\Lambda(Z)$.

Any operator $Z \in \mathbf{SU}(2)$ can be represented in the form

$$Z = A\sigma^x A^{-1} B\sigma^x B^{-1}, \qquad A, B \in \mathbf{SU}(2). \tag{S8.2}$$

Therefore $\Lambda(Z)$ is realized by the circuit shown in Figure S8.1a.

Geometrically, equation (S8.2) is equivalent to the assertion that any rotation of the three-dimensional space is the composition of two rotations through 180°. The proof of this assertion is shown in Figure S8.1b.

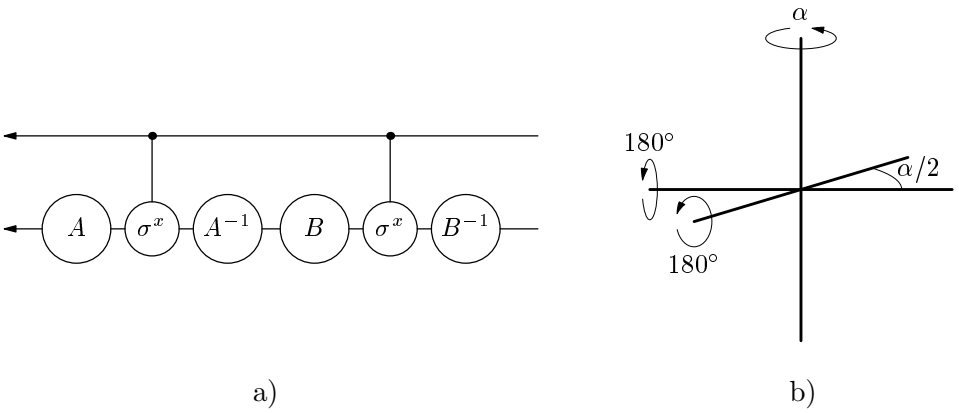


Fig. S8.1. Realization of the operator $\Lambda(Z)$, $Z \in \mathbf{SU}(2)$.

8.3. Let $e^{i\theta}$ be the required phase shift. The operators $X = e^{i\varphi_1}\Lambda(e^{i\theta})$ and $Y = e^{i\varphi_2}\sigma^x$ can be realized over the basis $\mathcal{A}$. Although the phases φ_1, φ_2 are arbitrary, we can realize $X^{-1} = e^{-i\varphi_1}\Lambda(e^{-i\theta})$ and $Y^{-1} = e^{-i\varphi_2}\sigma^x$ with exactly the same phases by inverting the circuits for X and Y. Therefore we can realize the operator

$$e^{i\theta\sigma^z} = X^{-1}Y^{-1}XY;$$

the unknown phases cancel out. It remains to apply this operator to an ancilla in the state $|0\rangle$.

8.4. We can construct a circuit for the operator $\Lambda(U)$ using the *Fredkin gate* $F = \Lambda(\leftrightarrow)$ — a controlled bit exchange. It is defined and realized as follows:

$$F|a,b,c\rangle \stackrel{\text{def}}{=} \begin{cases} |0,b,c\rangle & \text{if } a=0, \\ |1,c,b\rangle & \text{if } a=1, \end{cases}$$

$$F[1,2,3] = \Lambda^2(\sigma^x)[1,2,3]\ \Lambda^2(\sigma^x)[1,3,2]\ \Lambda^2(\sigma^x)[1,2,3].$$

Figure S8.2 shows how, given an arbitrary gate U preserving $|0\rangle$, one can construct a circuit for $\Lambda(U)$. The controlled exchange (shown in rectangles) is performed with two n-qubit registers by applying n copies of the Fredkin gate. If the controlling qubit contains 1, the state $|\xi\rangle$ will be submitted to U, otherwise the input to U is $|0\rangle$.

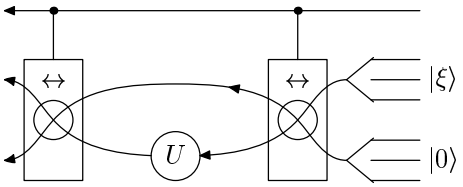


Fig. S8.2. Realization of the operator $\Lambda(U)$, assuming that $U|0\rangle = |0\rangle$.

8.5 (cf. [**7**]). We are going to give a solution with $r = 1$, but this will require some preparation. Our final circuit will be built of subcircuits realizing $\Lambda^n(\sigma^x)$ for $n < k$ with $r = O(n)$ ancillas. The idea is that if a subcircuit operates only on some of the qubits, it can borrow the remaining qubits and use them as indicated, i.e., realizing $U \otimes I_{\mathcal{B}^r}$ instead of U. Ancillas used in this special way will be called *dummy qubits*.

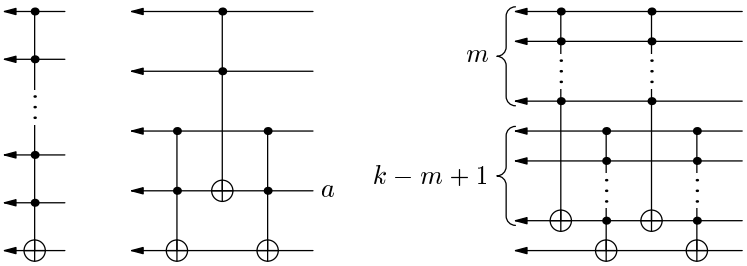


Fig. S8.3

Let us introduce a graphic notation for the operator $\Lambda^k(\sigma^x)$; see Figure S8.3a. The key construction is shown in Figure S8.3b. This circuit performs the following actions:

1. The value a of the fourth (second to bottom) bit is changed if and only if the first two bits are 1. (This change can be compensated by applying the Toffoli gate one more time.)
2. The bottom bit is altered if and only if the first three bits are 1.

The most obvious use of this primitive is shown in Figure S8.3c, where the operator $\Lambda^k(\sigma^x)$ is realized by two copies of $\Lambda^m(\sigma^x)$ and two copies of $\Lambda^{k-m+1}(\sigma^x)$, using one dummy qubit (the second to the bottom one).

Now recall the idea we mentioned at the beginning. In implementing the operators $\Lambda^n(\sigma^x)$ for $n = m$ and $n = k - m + 1$ we can use $k - m + 1$ and m dummy qubits, respectively. If we set $m = \lceil k/2 \rceil$, then in both cases we will have at least $n - 1$ dummy qubits available. Therefore it remains to realize $\Lambda^n(\sigma^x)$ using at most $n - 1$ dummy qubits.

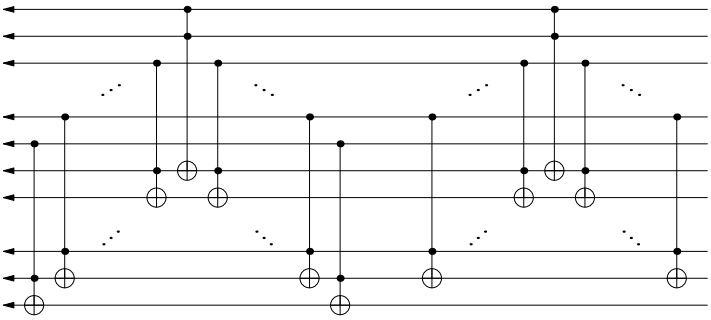


Fig. S8.4. Realization of $\Lambda^n(\sigma^x)$ with $n - 2$ dummy qubits.

The circuit in Figure S8.4 realizes $\Lambda^n(\sigma^x)$ with $n - 2$ dummy qubits (number $n + 1$ through $2n - 2$). The part of the circuit on the right (which is applied first) changes the values of the bits as follows:

$$x_{n+j} \mapsto x_{n+j} \oplus x_1 x_2 \cdots x_{j+1}, \qquad j = 1, \ldots, n-2.$$

The left part acts similarly, but j runs from 1 to $n - 1$. These two actions cancel each other, except for the change in the last bit,

$$x_{2n-1} \mapsto x_{2n-1} \oplus x_1 x_2 \cdots x_n.$$

8.6. The operator $\Lambda^k(U)$ has been realized by the circuit shown in Figure 8.5. Each of the operators $\Lambda^j(Z_{k-j})$ in that circuit is represented by two copies of $\Lambda^j(\sigma^x)$ (or one copy of $\Lambda^j(i\sigma^x)$ and one copy of $\Lambda^j(-i\sigma^x)$) and four applications of one-qubit gates (cf. Figure S8.1a). Let us examine this construction once again. We note that for the realization of the

operator $\Lambda^j(\sigma^x)$ one can use up to $k-j$ dummy qubits (see the solution to Problem 8.5). Thus for $j=1,\ldots,k-1$ each of these operators can be realized by a circuit of size $O(k)$. The remaining constant number of operators $\Lambda^j(\pm i\sigma^x)$ are realized by circuits of size $O(k^2)$ as suggested in the main text (see Figure 8.4). The total size of the resulting circuit is $O(k^2)$.

8.7. Property (8.10) follows from this chain of inequalities:

$$\big\|XY|\xi\rangle\big\| \le \|X\|\,\big\|Y|\xi\rangle\big\| \le \|X\|\,\|Y\|\,\big\||\xi\rangle\big\|.$$

To prove (8.11), we note that the the operators $XX^\dagger$ and $X^\dagger X$ have the same nonzero eigenvalues (see Problem 6.4).

Equation (8.12) follows from the fact that the eigenvalues of the operator $(X\otimes Y)^\dagger(X\otimes Y) = (X^\dagger X)\otimes(Y^\dagger Y)$ have the form $x_j y_k$, where x_j and y_k are the eigenvalues of $X^\dagger X$ and $Y^\dagger Y$, respectively.

Equation (8.13) is obvious.

8.8. a) The condition that $\tilde U$ approximates U with precision δ is expressed by inequality (8.16). Multiplying the expression under the norm by $\tilde U^{-1}$ on the left and by U^{-1} on the right, we get $\|VU^{-1} - \tilde U^{-1}V\| \le \delta$.

b) It is sufficient to verify the statement for $L=2$:

$$\begin{aligned}\big\|\tilde U_2\tilde U_1 V - VU_2U_1\big\| &= \big\|\tilde U_2(\tilde U_1 V - VU_1) + (\tilde U_2 V - VU_2)U_1\big\| \\ &\le \big\|\tilde U_2(\tilde U_1 V - VU_1)\big\| + \big\|(\tilde U_2 V - VU_2)U_1\big\| \\ &\le \big\|\tilde U_2\big\|\,\big\|\tilde U_1 V - VU_1\big\| + \big\|\tilde U_2 V - VU_2\big\|\,\big\|U_1\big\| \\ &= \big\|\tilde U_1 V - VU_1\big\| + \big\|\tilde U_2 V - VU_2\big\|.\end{aligned}$$

8.9. We rephrase the problem as follows. Let $Q = U\otimes I_{\mathcal{B}^{\otimes(N-n)}}$ and $\mathcal{M} = \mathcal{B}^{\otimes n}\otimes|0^{N-n}\rangle$. We have

$$\|\tilde U\Pi_{\mathcal{M}} - Q\Pi_{\mathcal{M}}\| \le \delta, \qquad Q\mathcal{M} = \mathcal{M},$$

and we are looking for a unitary operator W such that $W\Pi_{\mathcal{M}} = Q\Pi_{\mathcal{M}}$ and $\|W-\tilde U\| \le O(\delta)$. Let $\mathcal{L} = \tilde U\mathcal{M}$. We will try to find a unitary operator X such that

$$X\mathcal{L} = \mathcal{M}, \qquad \|X - I\| \le O(\delta). \tag{S8.3}$$

If such an X exists, then the following operator will serve as a solution:

$$W = Q\Pi_{\mathcal{M}} + X\tilde U(I - \Pi_{\mathcal{M}}).$$

To satisfy the conditions (S8.3), we show first that $\mathcal{L}$ and $\mathcal{M}$ are close enough:

$$\|\Pi_{\mathcal{L}} - \Pi_{\mathcal{M}}\| = \big\|\tilde U\Pi_{\mathcal{M}}\tilde U^\dagger - Q\Pi_{\mathcal{M}}Q^\dagger\big\| \le 2\delta,$$

since $\|\tilde U\Pi_{\mathcal{M}} - Q\Pi_{\mathcal{M}}\| \le \delta$. It remains to apply the following lemma.

Lemma. *Let $\mathcal{L}, \mathcal{M} \subseteq \mathcal{N}$ and $\|\Pi_{\mathcal{L}} - \Pi_{\mathcal{M}}\| \leq \varepsilon < 1/2$. Then there is a unitary operator X such that $X\mathcal{L} = \mathcal{M}$ and $\|X - I\| = O(\varepsilon)$.*

Proof. Let $Y = \Pi_{\mathcal{M}}\Pi_{\mathcal{L}} + (I - \Pi_{\mathcal{M}})(I - \Pi_{\mathcal{L}})$. It is immediately evident that Y takes $\mathcal{L}$ into $\mathcal{M}$ and $\mathcal{L}^{\perp}$ into $\mathcal{M}^{\perp}$. The norm $\|Y - I\|$ can be estimated as follows:

$$\begin{aligned}\|Y - I\| &= \|\Pi_{\mathcal{M}}\Pi_{\mathcal{L}} - \Pi_{\mathcal{M}} - \Pi_{\mathcal{L}} + \Pi_{\mathcal{M}}\Pi_{\mathcal{L}}\| \\ &\leq \|(\Pi_{\mathcal{M}} - \Pi_{\mathcal{L}})\Pi_{\mathcal{L}}\| + \|\Pi_{\mathcal{M}}(\Pi_{\mathcal{M}} - \Pi_{\mathcal{L}})\| \leq 2\varepsilon < 1.\end{aligned}$$

The operator Y is not unitary, but the above estimate shows that it is non-degenerate. Therefore we can define the operator $X = Y(Y^{\dagger}Y)^{-1/2}$, which is unitary. Since $Y^{\dagger}Y$ leaves the subspace $\mathcal{L}$ invariant, X takes $\mathcal{L}$ into $\mathcal{M}$. To estimate the norm of X we expand $(Y^{\dagger}Y)^{-1/2}$ into a Taylor series:

$$(Y^{\dagger}Y)^{-1/2} = I + \frac{1}{2}Z + \frac{3}{8}Z^2 + \cdots, \quad \text{where } Z = I - Y^{\dagger}Y.$$

Therefore $\left\|(Y^{\dagger}Y)^{-1/2} - I\right\| \leq (1 - \|Z\|)^{-1/2} - 1 = O(\varepsilon)$, from which we obtain $\|X - I\| = O(\varepsilon)$. □

8.10. Each of the considered rotations generates an everywhere dense subset in the group of rotations about a fixed axis. Indeed, this group is isomorphic to $\mathbb{R}/\mathbb{Z}$, and the generated subset consists of elements of the form $n\alpha \bmod 1$, $n \in \mathbb{Z}$, where α is irrational ($2\pi\alpha$ is the rotation angle). Therefore it remains to prove that the rotations about two different axes generate $\mathbf{SO}(3)$. For this it suffices to show that the subgroup generated by all rotations about two different lines acts transitively on the sphere. This fact becomes obvious by looking at Figure S8.5 (if we can move along two families of circles, then from any point of the sphere we can reach any other point). A rigorous proof is obtained similarly for the solution to Problem 8.11.

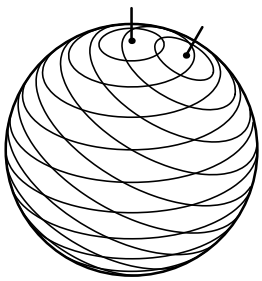


Fig. S8.5. Rotating about two axes in $\mathbb{R}^3$.

Remark. This solution is nonconstructive: we cannot give an upper bound for the number of generators X, X^{-1}, Y, Y^{-1} whose product approximates

a given element $U \in \mathbf{SO}(3)$ with a given precision δ, *even if* X *and* Y *are fixed*. Therefore the implied algorithm for finding such an approximation may exhibit arbitrary long running times. The reason for this nonconstructiveness is as follows. Although the number α is irrational, it might be very closely approximated by rational numbers (this occurs when the coefficients of the continued fraction expansion of α grow rapidly). Then any $r \in \mathbb{R}/\mathbb{Z}$ is approximated by elements of the form $n\alpha$ $(n \in \mathbb{Z})$ with arbitrary precision $\delta > 0$, but the number n can be arbitrarily large: larger than any specified function of δ.

A constructive proof and an effective (for fixed X and Y) algorithm for finding an approximation are rather complicated; see Section 8.3.

8.11. If we set $|\xi'\rangle = V^{-1}|\xi\rangle$, then $H' = V^{-1}HV$ is the stabilizer of $\mathbb{C}(|\xi'\rangle)$. Then the problem takes the following form: prove that the union of the stabilizers of two distinct one-dimensional subspaces generates $\mathbf{U}(\mathcal{M})$.

It suffices to show that the group G generated by $H \cup H'$ acts transitively on the set of unit vectors. Indeed, suppose that for each unit vector $|\psi\rangle$ there exists $U_\psi \in G$ such that $U_\psi|\xi\rangle = |\psi\rangle$. Then

$$\mathbf{U}(\mathcal{M}) = \bigcup_{|\psi\rangle \in \mathcal{M}} U_\psi H.$$

We prove the transitivity of the action of the group G. We note that for any unit vector $|\psi\rangle$,

$$H|\psi\rangle = Q(\vartheta) \stackrel{\text{def}}{=} \big\{|\eta\rangle : |\langle\eta|\xi\rangle| = \cos\vartheta\big\},$$
$$H'|\psi\rangle = Q'(\vartheta') \stackrel{\text{def}}{=} \big\{|\eta\rangle : |\langle\eta|\xi'\rangle| = \cos\vartheta'\big\},$$

where $\vartheta = \vartheta(|\psi\rangle)$, $\vartheta' = \vartheta'(|\psi\rangle)$ denote the angles between $|\psi\rangle$ and $|\xi\rangle$, $|\xi'\rangle$, respectively:

$$\cos\vartheta = |\langle\psi|\xi\rangle|, \quad \cos\vartheta' = |\langle\psi|\xi'\rangle|, \quad 0 \le \vartheta, \vartheta' \le \pi/2.$$

In further formulas we will also use the angle α between the vectors $|\xi\rangle$ and $|\xi'\rangle$: $\cos\alpha = |\langle\xi|\xi'\rangle|$, $0 \le \alpha \le \pi/2$.

It can be verified that for $\dim\mathcal{M} \ge 3$ the angle between the vector $|\xi\rangle$ and elements of $Q'(\vartheta')$ varies from $|\alpha - \vartheta'|$ to $\min\{\alpha + \vartheta', \pi/2\}$. Similarly, the angle between $|\xi'\rangle$ and elements of $Q(\vartheta)$ varies from $|\alpha - \vartheta|$ to $\min\{\alpha + \vartheta, \pi/2\}$. Therefore

$$HQ'(\vartheta') = \bigcup_{|\alpha-\vartheta'| \le \vartheta \le \min\{\alpha+\vartheta', \pi/2\}} Q(\vartheta),$$
$$H'Q(\vartheta) = \bigcup_{|\alpha-\vartheta| \le \vartheta' \le \min\{\alpha+\vartheta, \pi/2\}} Q'(\vartheta').$$

It follows that

$$H'|\xi\rangle = Q'(a),$$
$$HH'|\xi\rangle = \bigcup_{0\le\vartheta\le\min\{2\alpha,\pi/2\}} Q(\vartheta),$$
$$H'HH'|\xi\rangle = \bigcup_{0\le\vartheta'\le\min\{3\alpha,\pi/2\}} Q'(\vartheta'),$$

and so forth. Hence, acting on the vector $|\xi\rangle$ alternately with elements from H' and H sufficiently many times, we can obtain any unit vector $|\psi\rangle$.

8.12. First, we will realize the operator $\Lambda^2(i)\otimes I_{\mathcal{B}}$, i.e., the application of $\Lambda^2(i)$ to the first two qubits. To this end, we will use the operators $\Lambda^2(\sigma^\alpha)$ $(\alpha = x, y, z)$. We note that $\sigma^y = K\sigma^x K^{-1}$, $\sigma^z = H\sigma^x H$, hence

$$\Lambda^2(\sigma^y)[1,2,3] \;=\; K[3]\;\Lambda^2(\sigma^x)[1,2,3]\;K^{-1}[3],$$
$$\Lambda^2(\sigma^z)[1,2,3] \;=\; H[3]\;\Lambda^2(\sigma^x)[1,2,3]\;H[3].$$

Using the identity $\sigma^x\sigma^y\sigma^z = iI_{\mathcal{B}}$, we obtain

$$\Lambda^2(\sigma^x)\Lambda^2(\sigma^y)\Lambda^2(\sigma^z) = \Lambda^2(iI_{\mathcal{B}}) = \Lambda^2(i)\otimes I_{\mathcal{B}}.$$

The circuit for the realization of $\Lambda^2(i)\otimes I_{\mathcal{B}}$ is shown in Figure S8.6. The inverse operator, $\Lambda^2(-i)\otimes I_{\mathcal{B}}$, can be realized in a similar way.

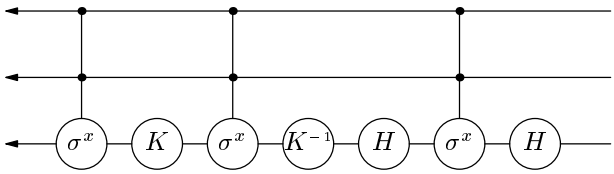


Fig. S8.6. Realization of the operator $\Lambda^2(i)\otimes I_{\mathcal{B}}$ over the standard basis.

Now we consider a new basis,

$$\mathcal{Q}' = \{H,\, \Lambda^2(i),\, \Lambda^2(-i)\}.$$

It it sufficient to show that the applications of elements of this basis to two qubits generate a dense subset of $\mathbf{U}(\mathcal{B}^{\otimes 2})/\mathbf{U}(1)$. Let $X = \Lambda(HKH)$; this operator can be realized as follows:

$$X[1,2] = \; H[2]\;\Lambda(K)[1,2]\;H[2], \qquad \Lambda(K) = \Lambda^2(i).$$

We act with X on $\mathcal{B}^{\otimes 2}$ in two possible ways: $X_1 = X[1,2]$ and $X_2 = X[2,1]$. The operators $Y_1 = X_1X_2^{-1}$, $Y_2 = X_2^{-1}X_1$ are also realizable over the basis $\mathcal{Q}'$.

The operators X_1, X_2 (and consequently also Y_1, Y_2) preserve the vectors $|00\rangle$ and $|\eta\rangle = |01\rangle + |10\rangle + |11\rangle$. Direct calculations show that Y_1 and Y_2 do

not commute and have eigenvalues $1, 1, \lambda_+, \lambda_-$, where $\lambda_\pm = (1 \pm \sqrt{-15})/4 = e^{\pm i\varphi/2}$. The last two eigenvalues characterize the action of Y_1, Y_2 on the subspace $\mathcal{L} = \mathbb{C}\big(|00\rangle, |\eta\rangle\big)^\perp$. In $\mathbf{SO}(3) \cong \mathbf{U}(\mathcal{L})/\mathbf{U}(1)$ an operator with such eigenvalues corresponds to a rotation through the angle φ about some axis. We will show that this angle is incommensurate with π, so that Y_1, Y_2 generate an everywhere dense subset in $\mathbf{U}(\mathcal{L})/\mathbf{U}(1)$ (see Problem 8.10).

If φ/π were rational, then λ_+, λ_- would be roots of 1, and hence algebraic integers. The minimal (over rationals) polynomial for an algebraic integer λ has the form $f_\lambda(x) = x^n + a_1 x^{n-1} + \cdots + a_n$, where $a_j \in \mathbb{Z}$. However, the minimal polynomial for $\lambda_\pm$ is $x^2 - \frac{1}{2}x + 1$; therefore $\lambda_\pm$ are not algebraic integers.

To complete the proof, we apply the result of Problem 8.11 twice. The operators Y_1, Y_2 generate an everywhere dense subset in $\mathbf{U}(\mathcal{L})/\mathbf{U}(1)$, and the operator $V = \Lambda(K)$ preserves $\mathbb{C}(|00\rangle)$ but not $\mathbb{C}(|\eta\rangle)$, so that Y_1, Y_2, $V^{-1}Y_1V$, $V^{-1}Y_2V$ generate an everywhere dense set in $\mathbf{U}\big(\mathcal{L} \oplus \mathbb{C}(|\eta\rangle)\big)/\mathbf{U}(1)$. The operator $H[1]$ does not preserve $C(|00\rangle)$; applying the result of Problem 8.11 once again, we obtain an everywhere dense subset in $\mathbf{U}(\mathcal{B}^{\otimes 2})/\mathbf{U}(1)$.

8.13. It is clear that powers of the operator $R^4 = \exp(4\pi i \alpha \sigma^x) \in \mathbf{SU}(2)$ approximate $\exp(i\varphi\sigma^x)$ for any given φ with any given precision. Hence powers of R approximate operators of the form

$$i^s \exp(i\varphi\sigma^x) = i^s \begin{pmatrix} \cos\varphi & i\sin\varphi \\ i\sin\varphi & \cos\varphi \end{pmatrix}, \qquad s = 0, 1, 2, 3.$$

For $s = 3$ and $\varphi = \pi/2$ this expression yields σ^x, so that we obtain the Toffoli gate: $(\Lambda^2(R))^k \approx \Lambda^2(\sigma^x)$ for suitable k. For $s = 1, 3$ and $\varphi = 0$ we get $\Lambda^2(\pm i I_\mathcal{B}) = \Lambda^2(\pm i) \otimes I_\mathcal{B}$ (the identity factor may be ignored).

Now we show how to eliminate unnecessary control qubits:

$$\sigma^x[1]\ \Lambda(X)\ \sigma^x[1]\ \Lambda(X) = I_\mathcal{B} \otimes X.$$

Thus we can realize $\Lambda(\sigma^x)$, $K = \Lambda(i)$, $K^{-1} = \Lambda(-i)$ and any operator of the form $\exp(i\varphi\sigma^x)$. According to Problem 8.2, these operators form a complete basis.

8.14. a) Let $\Gamma' \subseteq \Gamma$ be a maximal subset with the property that the distance between any pair of distinct elements is greater than $\alpha\delta'$, where $\delta' = \delta/(1-\alpha)$ ("maximal" means that any larger subset does not have this property). Then Γ' is an $\alpha\delta'$-net for Γ. But Γ is a δ-net for R, so Γ' is an $\alpha\delta' + \delta$-net for R. Note that $\alpha\delta' + \delta = \delta'$. The intersection of Γ' with the δ'-neighborhood of R is an α-sparse δ'-net for R.

b) The proof is based on a volume consideration. For our purposes it is sufficient to consider the volume on the ambient space $\mathcal{P} = \mathbf{L}(\mathbb{C}^M) \supseteq \mathbf{SU}(M)$ rather than the intrinsic volume on $\mathbf{SU}(M)$. We regard $\mathcal{P}$ as a

$2M^2$-dimensional real space endowed with a norm. The volume of the a-neighborhood of any point in $\mathcal{P}$ is a^{2M^2} (up to an arbitrary constant factor).

Let $a = \alpha r/(2q)$, $b = r + r/q + a$. Then the a-neighborhoods of the elements of the net do not overlap and are contained in the b-neighborhood of I. Therefore the number of elements does not exceed $(b/a)^{2M^2}$. But

$$\frac{b}{a} = \frac{q}{\alpha}\left(2 + \frac{2+\alpha}{q}\right) \le \frac{5q}{\alpha},$$

since $\alpha < 1$ and $q > 1$.

8.15. The inclusion $[R_a, R_b] \subseteq R_{2ab}$ is almost obvious: if $\|X\| \le a$, $\|Y\| \le b$, then

$$\big\|[X,Y]\big\| = \|XY - YX\| \le 2\|X\|\,\|Y\| \le 2ab.$$

So, we need to prove that $R_{ab/4} \subseteq [R_a, R_b]$. Without loss of generality we may assume that $a = b = 2$. Let us consider an arbitrary $Z \in \mathfrak{su}(M)$ such that $\|Z\| \le 1$. Our goal is to represent Z as $[X,Y]$, where $\|X\|, \|Y\| \le 2$.

Let us choose an arbitrary basis in $\mathbb{C}^M$ in which Z is diagonal,

$$Z = i\begin{pmatrix} z_1 & & 0 \\ & \ddots & \\ 0 & & z_M \end{pmatrix}, \qquad z_1, \dots, z_M \in \mathbb{R}, \quad \sum_{j=1}^{M} z_j = 0, \quad |z_j| \le 1.$$

Lemma. *There exists a permutation $\tau : \{1, \dots, M\} \to \{1, \dots, M\}$ such that $0 \le \sum_{j=1}^{k} z_{\tau(j)} \le 2$ for $k = 0, \dots, M$.*

Proof. We will pick elements from the set $\{z_1, \dots, z_M\}$ one by one. Suppose that $k-1$ elements $z_{\tau(1)}, \dots, z_{\tau(k-1)}$ has been already chosen so that $w_{k-1} = \sum_{j=1}^{k-1} z_{\tau(j)}$ satisfies the inequality $0 \le w_{k-1} \le 2$. The sum of the remaining elements equals $-w_{k-1}$, so there are some $z_p \ge -w_{k-1}$ and $z_q \le 0$ among them. We set $\tau(k) = p$ if $w_{k-1} < 1$, and $\tau(k) = q$ otherwise. In both cases the number $w_k = w_{k-1} + z_{\tau(k)}$ satisfies $0 \le w_k \le 2$. □

By applying the permutation τ to the basis vectors, we can arrange that the partial sums

$$w_k = \sum_{j=1}^{k} z_{\tau(j)}, \qquad k = 0, \dots, M,$$

satisfy the inequality $0 \le w_k \le 2$. Let $v_k = \sqrt{w_k/2}$. Then $Z = XY - YX$, where

$$X = i \begin{pmatrix} 0 & v_1 & 0 & \cdots & 0 \\ v_1 & 0 & v_2 & & \vdots \\ 0 & v_2 & 0 & \ddots & 0 \\ \vdots & & \ddots & \ddots & v_{M-1} \\ 0 & \cdots & 0 & v_{M-1} & 0 \end{pmatrix},$$

$$Y = \begin{pmatrix} 0 & -v_1 & 0 & \cdots & 0 \\ v_1 & 0 & -v_2 & & \vdots \\ 0 & v_2 & 0 & \ddots & 0 \\ \vdots & & \ddots & \ddots & -v_{M-1} \\ 0 & \cdots & 0 & v_{M-1} & 0 \end{pmatrix}.$$

We will estimate the norms of X and Y, using the fact that $0 \le v_k \le 1$. Note that X and Y are conjugate (by the matrix that has i^k on the diagonal and 0 off the diagonal), so $\|X\| = \|Y\|$. Let us examine the matrix $A = i^{-1}X$, which obviously has the same norm, $\|A\| = \max_j |\lambda_j|$, where $\{\lambda_j\}$ is the spectrum of A. All entries of A are nonnegative, so the Perron–Frobenius theorem [**50**] applies. Thus there exists an eigenvalue $\lambda_{\max} = \lambda_{\max}(A)$ such that $|\lambda_j| \le \lambda_{\max}$ for all j. The corresponding eigenvector has nonnegative coefficients. It is easy to see that

$$\lambda_{\max}(A) = \lim_{N\to\infty} \Big(\langle \xi | A^N | \xi \rangle \Big)^{1/N},$$

where $|\xi\rangle$ is an arbitrary vector with positive coefficients. Therefore $\lambda_{\max}(A)$ is a monotone function of the matrix entries, i.e., $\lambda_{\max}$ cannot decrease if the entries increase. It follows that $\|A\|$ does not exceed the largest eigenvalue of the matrix

$$B = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 1 & & \vdots \\ 0 & 1 & 0 & \ddots & 0 \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}.$$

The latter is equal to $2\cos(\pi/(M+1)) < 2$.

8.16. By bringing X to the diagonal form, the inequality (8.19) can be derived from the inequality $|e^{ix} - 1 - ix| \le O(x^2),\ x \in \mathbb{R}$.

To prove (8.20), let us fix X and Y and use the formulas

$$\exp(X+Y)=\lim_{n\to\infty}\bigl(e^{X/n}e^{Y/n}\bigr)^n,\qquad \exp(X)\exp(Y)=\bigl(e^{X/n}\bigr)^n\bigl(e^{Y/n}\bigr)^n.$$

To pass from $P=\bigl(e^{X/n}e^{Y/n}\bigr)^n$ to $Q=\bigl(e^{X/n}\bigr)^n\bigl(e^{Y/n}\bigr)^n$, one needs to pull all $e^{X/n}$ factors to the left, commuting them with $e^{Y/n}$ factors on the way. Thus the difference $Q-P$ can be represented as a sum of $n(n-1)/2$ terms of the form $U(e^{X/n},e^{Y/n}-e^{Y/n}e^{X/n})V$, where U,V are unitary. The norm of each term equals

$$\bigl\|e^{X/n}e^{Y/n}-e^{Y/n}e^{X/n}\bigr\|\le\frac{1}{n^2}\bigl\|[X,Y]\bigr\|+O\left(\frac{1}{n^3}\right),$$

hence

$$\bigl\|\exp(X)\exp(Y)-\exp(X+Y)\bigr\|\le\frac{1}{2}\bigl\|[X,Y]\bigr\|\le\|X\|\,\|Y\|.$$

The inequality (8.21) follows from (8.19). Indeed, let $A\approx B$ denote that

$$\|A-B\|\le O\bigl(\|X\|\,\|Y\|\,\bigl(\|X\|+|Y\|\bigr)\bigr)$$

(assuming that X and Y are fixed). Then

$$\begin{aligned}[\![e^X,e^Y]\!]-I&=\Bigl((e^X-I)(e^Y-I)-(e^Y-I)(e^X-I)\Bigr)e^{-X}e^{-Y}\\&\approx XY-YX=[X,Y]\approx\exp([X,Y])-I.\end{aligned}$$

8.17. a) In view of the result of Problem 8.14a), it is sufficient to show that $[\![\Gamma_1,\Gamma_2]\!]$ is an $\bigl(r_1r_2/4,\,(25/6)r_1r_2/q\bigr)$-net.

Let $V\in S_{r_1r_2/4}$. Due to the property of the group commutator (8.18), there are some $V_1\in S_{r_1}$ and $V_2\in S_{r_2}$ such that

$$d\bigl([\![V_1,V_2]\!],V\bigr)\le O(r_1r_2(r_1+r_2)).$$

Each V_j $(j=1,2)$ can be approximated by an element $U_j\in\Gamma_j$ with precision $\delta_j=r_j/q$. Using the biinvariance of the distance function and the property of the group commutator (8.17), we obtain

$$\begin{aligned}d\bigl([\![U_1,U_2]\!],[\![V_1,V_2]\!]\bigr)&\le d\bigl([\![U_1,U_2]\!],[\![V_1,U_2]\!]\bigr)+d\bigl([\![V_1,U_2]\!],[\![V_1,V_2]\!]\bigr)\\&=d\bigl([\![V_1^{-1}U_1,U_2]\!],I\bigr)+d\bigl(I,[\![V_1,U_2^{-1}V_2]\!]\bigr)\\&\le 2d(V_1,U_1)d(U_2,I)+2d(V_1,I)d(V_2,U_2)\\&\le 2\delta_1(r_2+\delta_2)+2r_1\delta_2=4r_1r_2/q+2r_1r_2/q^2.\end{aligned}$$

Therefore

$$\begin{aligned}d\bigl([\![U_1,U_2]\!],V\bigr)&\le O(r_1r_2(r_1+r_2))+4r_1r2(q^{-1}+q^{-2}/2)\\&\le(4+f(r,q))r_1r_2/q,\end{aligned}$$

where $f(r,q) = q^{-1}/2 + O(r_1 + r_2)q$. If r_1, r_2 are small enough (as in the condition of the problem), then $f(r,q)$ is small too; we may assume that $f(r,q) \le 1/6$. Thus $d\big([\![U_1, U_2]\!], V\big) \le 25/6$.

b) Let $V \in S_{r_1}$. Then there is some $U_1 \in \Gamma_1$ such that $d(V, U_1) \le \delta_1$. Therefore $U_1^{-1}V \in S_{\delta_1} \subseteq S_{r_2}$. It follows that $d(U_1^{-1}V, U_2) \le \delta_2$ for some $U_2 \in \Gamma_2$, but $d(U_1^{-1}V, U_2) = d(V, U_1U_2)$.

c) We just need to iterate the previous argument. Note that the elements $Z_j \in \Gamma_j$ can be found by an effective procedure which involves $O(n)$ group operations and calculations of the distance function.

S9. Problems of Section 9

9.1. First, we apply k copies of the circuit U; then we compute the majority function bitwise, i.e., we apply m copies of an operator M that realizes $\mathrm{MAJ}_\oplus$ with s ancillas. Thus the complete circuit can be represented symbolically as $W = M^{\otimes m}U^{\otimes k}$. We need to estimate the probability of obtaining the correct result $y = F(x)$, given by

$$p(y|x) = \sum_{\substack{y_1,\dots,y_k\\ z_1,\dots,z_k}} \Big|\big\langle y_1, z_1, \dots, y_k, z_k,\, y,\, 0^{ms}\big|W\big|(x, 0^{N-n})^k,\, 0^m,\, 0^{ms}\big\rangle\Big|^2,$$

assuming that $\sum_z \big|\langle F(x), z|U|x, 0^{N-n}\rangle\big|^2 = 1 - \varepsilon_x$, where $\varepsilon_x \le \varepsilon < 1/2$ for each x.

If more than half of the output registers of the initial circuit contain $F(x)$, then the result of the bitwise majority vote will necessarily be $F(x)$. Therefore, similarly to (4.1) on p. 37, we have

$$\begin{aligned} 1 - p(F(x)|x) &\le \sum_{\substack{S\subseteq\{1,\dots,k\},\\ |S|\le k/2}} \sum_{\substack{y_1,\dots,y_k,\\ y_j = F(x) \Leftrightarrow j\in S}} \sum_{z_1,\dots,z_k} \Big|\big\langle y_1, z_1, \dots, y_k, z_k\big|U^{\otimes k}\big|(x, 0^{N-n})^k\big\rangle\Big|^2 \\ &= \sum_{\substack{S\subseteq\{1,\dots,k\},\\ |S|\le k/2}} (1-\varepsilon_x)^{|S|}\varepsilon_x^{k-|S|} < \lambda^k, \qquad \text{where } \lambda = 2\sqrt{(1-\varepsilon)\varepsilon}. \end{aligned}$$

9.2. Let $|\xi\rangle = |x, 0^{N-n}\rangle$ and $\mathcal{M} = |F(x)\rangle \otimes \mathcal{B}^{N-m}$ (cf. Remark 10.1). We have

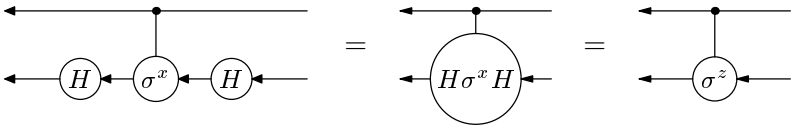
$$\langle\xi|U^\dagger \Pi_{\mathcal{M}} U|\xi\rangle \ge 1 - \varepsilon, \qquad \|\tilde{U} - U\| \le L\delta.$$

Thus, we do the following estimate:

$$\begin{aligned}
&\Big|\langle\xi|\tilde{U}^\dagger\Pi_{\mathcal{M}}\tilde{U}|\xi\rangle - \langle\xi|U^\dagger\Pi_{\mathcal{M}}U|\xi\rangle\Big| \\
&\qquad = \Big|\langle\xi|(\tilde{U}^\dagger - U^\dagger)\Pi_{\mathcal{M}}\tilde{U}|\xi\rangle + \langle\xi|U^\dagger\Pi_{\mathcal{M}}(\tilde{U}-U)|\xi\rangle\Big| \\
&\qquad \le \Big|\langle\xi|(\tilde{U}^\dagger - U^\dagger)\Pi_{\mathcal{M}}\tilde{U}|\xi\rangle\Big| + \Big|\langle\xi|U^\dagger\Pi_{\mathcal{M}}(\tilde{U}-U)|\xi\rangle\Big| \\
&\qquad \le \|\tilde{U}^\dagger - U^\dagger\| + \|\tilde{U}-U\| \le 2L\delta,
\end{aligned}$$

which implies that $\langle\xi|\tilde{U}^\dagger\Pi_{\mathcal{M}}\tilde{U}|\xi\rangle \ge 1-\varepsilon-2L\delta$.

9.3. If we change the basis in the controlled qubit, we get the operator $\Lambda^2(-1) = \Lambda(\sigma^z)$. Indeed,

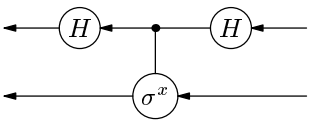


This operator multiplies $|1,1\rangle$ by -1 and does not change the remaining basis vectors.

Let us see what happens if we change the basis in the controlling qubit. The resulting operator is $H[1]\,\Lambda(\sigma^x)[1,2]\,H[1]$; we calculate its matrix elements:

$$\begin{aligned}
&\big\langle a,b\,\big|\,H[1]\,\Lambda(\sigma^x)[1,2]\,H[1]\,\big|\,c,d\big\rangle \\
&\qquad = \sum_{x,y}\left(\frac{1}{\sqrt{2}}(-1)^{ax}\right)\big\langle x,b\,\big|\,\Lambda(\sigma^x)[1,2]\,\big|\,y,d\big\rangle\left(\frac{1}{\sqrt{2}}(-1)^{yc}\right) \\
&\qquad = \frac{1}{2}\sum_{x,y}(-1)^{ax+cy}\,\delta_{x,y}\,\delta_{b,\,y\oplus d} = \frac{1}{2}(-1)^{(a+c)(b+d)}.
\end{aligned}$$

Thus



$$= \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{pmatrix} = \sigma^x[1]\,\sigma^x[2]\,V,$$

where $V = I - 2|\xi\rangle\langle\xi|$ and $|\xi\rangle = \frac{1}{2}\sum_{x,y}|x,y\rangle$. (Recall that a multiqubit version of the operator V was defined on page 85.)

9.4. Part a) follows from part b).

b) Let us describe a circuit that gives an approximate solution. First of all, we write the recursive formula

$$|\eta_{n,q}\rangle = \cos\vartheta\,|0\rangle\otimes|\eta_{n-1,q'}\rangle + \sin\vartheta\,|1\rangle\otimes|\eta_{n-1,q''}\rangle, \tag{S9.1}$$

where

$$\begin{array}{llll} q' = 2^{n-1}, & q'' = q - 2^{n-1}, & \vartheta = \arccos\sqrt{q'/q} & \text{if } q > 2^{n-1}; \\ q' = q, & q'' = 1, & \vartheta = 0 & \text{if } q \le 2^{n-1}. \end{array}$$

We describe a computation procedure based on formula (S9.1). It consists of the following steps.

1. Compute q', q'' and ϑ/π, with the latter number represented as an approximation by l binary digits. Store the results of the computation in supplementary qubits.
2. Apply the operator
$$R(\vartheta) = \begin{pmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{pmatrix}$$
to the first qubit of the register in which we are composing $|\eta_{n,q}\rangle$. (It initially contains $|0^n\rangle$.)
3. In the remaining $n-1$ bits, produce a state depending on the value of the first bit: if it equals 0, then create the state $|\eta_{n-1,q'}\rangle$ (by calling the procedure recursively); otherwise create $|\eta_{n-1,q''}\rangle$.
4. Reverse step 1 to clear the supplementary memory.

The operator $R(\vartheta)$ is realized approximately. Let $\vartheta/\pi = \sum_{k=1}^{l} a_k 2^{-k}$. Then $R(\vartheta) \approx R(\pi/2^l)^{a_l} \cdots R(\pi/2)^{a_1}$ with precision $O(2^{-l})$. Thus the action of the operator $R(\vartheta)$, controlled by ϑ, is represented as the product of operators $\Lambda(R(\pi/2^k))$, where the k-th bit of the number ϑ/π controls the application of the operator $R(\pi/2^k)$.

The overall precision of the constructed circuit is $\delta = O(n2^{-l})$; its size, expressed in terms of the length of the input and the precision, is $\mathrm{poly}(n + \log(1/\delta))$.

c) We describe the realization of the Fourier transform operator found by D. Coppersmith [**19**] and, independently, by D. Deutsch.

We enumerate the qubits in descending order from $n-1$ to 0. Thus a number x, $0 \le x < n$, is represented as $\overline{x_{n-1} \cdots x_0} = \sum_{k=0}^{n-1} 2^k x_k$, so that the exponent in the definition of the operator F_q $(q = 2^n)$ can be written as follows:

$$\frac{xy}{2^n} = \sum_{k=0}^{n-1} \sum_{j=0}^{n-1} 2^{k+j-n} x_k y_j \equiv \sum_{k+j<n} 2^{k+j-n} x_k y_j \pmod 1.$$

It is convenient to reverse the bit order in x, i.e., to replace k by $n-1-k$. This way the Fourier transform operator is represented in the form

$$F_{2^n} = V_n R_n,$$

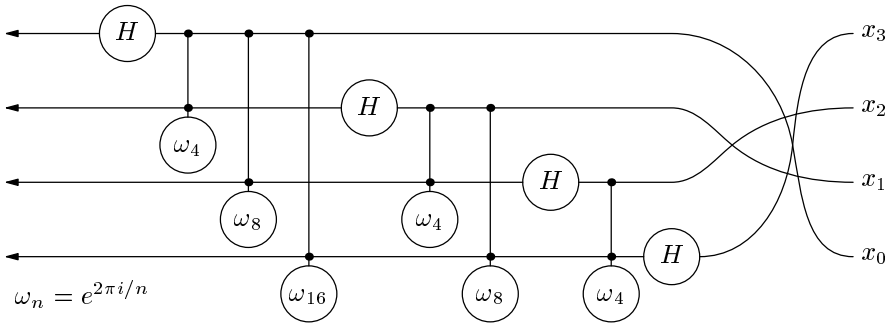


Fig. S9.1. Realization of the Fourier transform operator F_{2^n} (for $n = 4$).

where $R_n : |x_{n-1}, \dots, x_0\rangle \mapsto |x_0, \dots, x_{n-1}\rangle$, and V_n has the following matrix elements:

$$\langle y_{n-1}, \dots, y_0|V_n|x_{n-1}, \dots, x_0\rangle = \frac{1}{2^{n/2}} \exp\left(2\pi i \sum_{0\le j\le k<n} 2^{-(k-j+1)} x_k y_j\right).$$

Let us analyze the above equation. If we only keep terms with $j = k$ in the sum, we will get the matrix elements of the operator $H^{\otimes n}$. It is seen by inspection (and can also be proved by induction) that the remaining terms are reproduced by this formula:

$$\begin{aligned} V_n \;&=\; H[n-1]\, P_{n-1}\, H[n-2]\, P_{n-2} \,\cdots\, H[1]\, P_1\, H[0], \\ \langle y|P_k|x\rangle &= \exp\left(2\pi i \textstyle\sum_{j=0}^{k-1} x_k y_j\right) \delta_{y,x}. \end{aligned} \tag{S9.2}$$

Indeed, computing the matrix element $\langle y|V|x\rangle$ from formula (S9.2) amounts to the summation over paths from x to y. Looking at any path with nonzero contribution, we see that x_k passes through $H[0], \dots, H[k-1]$ unchanged, whereas $y_0, \dots, y_{k-1}$ pass through $H[n-1], \dots, H[k]$ unchanged.

It remains to realize the operators P_k:

$$P_k = S_{k,k-1}\, S_{k,k-2} \,\cdots\, S_{k,0}, \quad \text{where } S_{k,j} = \Lambda^2\big(e^{2\pi i/2^{k-j+1}}\big)[k,j] \quad (k > j).$$

The resulting circuit for the operator F_{2^n} is shown in Figure S9.1.

9.5. BPP $\subseteq$ BQP. A classical probabilistic computation can be represented by an invertible circuit $U_L \cdots U_1$, which, together with the input word x, uses a random sequence $r \in \mathcal{B}^s$ of 0s and 1s. (In addition to the result, the circuit can also produce some garbage — this does not matter.) For the quantum simulation of this circuit, we regard U_j as unitary operators permuting the basis vectors and, instead of the random word r, we prepare

the state

$$|\psi\rangle = H^{\oplus s}|0^s\rangle = 2^{-s/2}\sum_{r\in\mathcal{B}^s}|r\rangle.$$

BQP $\subseteq$ PP. Let a circuit $U_L\cdots U_1$ evaluate the predicate $Q(x)$ for $|x| = n$ with error probability $\le 1/3$, the total number of bits in the circuit being equal to N. The probability $p(x)$ of obtaining the result 1 can be expressed in terms of the projection $\Pi^{(1)} = |1\rangle\langle 1|$ applied to the first qubit:

$$\begin{aligned} p(x) &= \big\langle x, 0^{N-n}\big|U_1^\dagger U_2^\dagger\cdots U_L^\dagger\,\Pi^{(1)}[1]\,U_L U_{L-1}\cdots U_1\big|x, 0^{N-n}\big\rangle \\ &= 2^{-h}\big\langle x, 0^{N-n}\big|V_L V_{L-1}\cdots V_{-L+1}V_{-L}\big|x, 0^{N-n}\big\rangle. \end{aligned} \tag{S9.3}$$

Here $V_L,\dots,V_{-L}$ are the operators $U_1^\dagger,\dots,\Pi^{(1)}[1],\dots,U_1$, which are renumbered and also renormalized as follows: if $U_k = H[m]$ (or if $U_k^\dagger = H[m]$), then the corresponding operator V_j equals $\sqrt{2}H[m]$. The number of H gates in the circuit is denoted by h.

The matrix elements of the operators $V_j \in \big\{\sqrt{2}H, K, K^\dagger, \Lambda(\sigma^x), \Lambda^2(\sigma^x), \Pi^{(1)}\big\}$ belong to the set

$$M = \{0,\ +1,\ -1,\ +i,\ -i\}.$$

Multiplying the matrices, we obtain a sum of numbers from the set M. Since the quantity $p(x)$ we are interested in is real, we can limit ourselves to summing ± 1. The multiplicities of the summands are expressed in the form

$$\#_a(x) = \big|\{w:\ C_a(x,w) = 1\}\big|, \qquad a\in\{1,-1\},$$

where the predicates $C_a(x,w)$ will be defined below. Thus we obtain the representation

$$p(x) = 2^{-h}\big(\#_1(x) - \#_{-1}(x)\big).$$

The remaining part of the proof does not involve any quantum mechanics at all. First, we need an explicit description of the predicates $C_a(x,w)$. To this end, we express the matrix elements of the product $V_L\cdots V_{-L}$ in (S9.3) as a sum over all paths from the initial to the final state:

$$(V_L\cdots V_{-L})_{ab} = \sum_{x_{L-1},\dots,x_{-L+1}} (V_L)_{a x_{L-1}}\cdots(V_{-L})_{x_{-L+1}b}.$$

By definition, $C_a(u_{-L},\dots,u_L)$ equals 1 if and only if

$$u_{-L} = u_L = (x, 0^{N-n}) \quad\text{and}\quad \prod_{j=-L+1}^{L}(V_j)_{u_j u_{j-1}} = a.$$

It is easy to see that $C_a\in\mathbf{P}$: we only need to represent the matrix elements as powers of i and sum the exponents modulo 4.

If $Q(x) = 0$, then $p(x) \le 1/3$; if $Q(x) = 1$, then $p(x) \ge 2/3$. Thus $Q(x) = 1$ if and only if

$$p(x) = 2^{-h}\big(\#_1(x) - \#_{-1}(x)\big) > \frac{1}{2}.$$

This is equivalent to the condition

$$\#_{-1}(x) + 2^{h-1} < \#_1(x).$$

It remains to verify that both sides of the last inequality can be represented in the form

$$f(x) = \big|\{y : F(x,y) = 1\}\big|, \qquad F \in \mathbf{P}.$$

(Functions f of this form constitute the so-called class #P.) We have already proved that $\#_a(x)$ has the specified form, so it will suffice to show that the class #P is closed under addition. Let $g(x) = \big|\{y : G(x,y) = 1\}\big|$, $G \in \mathbf{P}$. Then $f(x) + g(x) = \big|\{(y,z) : T(x,y,z) = 1\}\big|$, where $T(x,y,0) = F(x,y)$ and $T(x,y,1) = G(x,y)$. The proof is completed.

PP $\subseteq$ PSPACE. This is obvious. We introduce two counters: one for R_0, the other for R_1. We scan through all possible values of y and add 1 to the k-th counter ($k = 0, 1$) if $R_k(x,y) = 1$. Then we compare the values of the counters.

S10. Problems of Section 10

10.1. Let $\rho = \sum_k p_k |\xi_k\rangle\langle\xi_k|$. We verify conditions 1)–3) for ρ.

1): This is obvious.

2): $\langle\eta|\rho|\eta\rangle = \sum_k p_k \langle\eta|\xi_k\rangle\langle\xi_k|\eta\rangle = \sum_k p_k \big|\langle\eta|\xi_k\rangle\big|^2 \ge 0$.

3): $\operatorname{Tr}\rho = \sum_k p_k \langle\xi_k|\xi_k\rangle = \sum_k p_k = 1$.

Conversely, if ρ satisfies 1)–3), then $\rho = \sum_k \lambda_k |\xi_k\rangle\langle\xi_k|$, where λ_k are the eigenvalues and $\{|\xi_k\rangle\}$ is an orthonormal basis of eigenvectors of ρ.

10.2. We note that $\mathcal{N} \otimes \mathcal{F} \cong \mathbf{L}(\mathcal{F}^*, \mathcal{N})$, so that the vector $|\psi\rangle \in \mathcal{N} \otimes \mathcal{F}$ can be translated to a linear map $X : \mathcal{F}^* \to \mathcal{N}$. The Schmidt decomposition for $|\psi\rangle$ is basically the singular value decomposition for X (see formula (6.2)) — we just need to change the designation of the bra-vectors $\langle\nu_j| \in (\mathcal{F}^*)^* \cong \mathcal{F}$ to $|\eta_j\rangle$. The condition $\lambda_j \le 1$ follows from the fact that λ_j^2 are the nonzero eigenvalues of the operator $XX^\dagger = \operatorname{Tr}_{\mathcal{F}}\big(|\psi\rangle\langle\psi|\big)$, which implies that $\sum_j \lambda_j^2 = 1$.

10.3. As follows from the solution of the previous problem, the condition $\operatorname{Tr}_{\mathcal{F}}\big(|\psi_1\rangle\langle\psi_1|\big) = \operatorname{Tr}_{\mathcal{F}}\big(|\psi_2\rangle\langle\psi_2|\big)$ allows us to choose Schmidt decompositions

for $|\psi_1\rangle$ and $|\psi_2\rangle$ with identical λ_j and $|\xi_j\rangle$. We write down these decompositions:

$$|\psi_k\rangle = \sum_j \lambda_j |\xi_j\rangle \otimes |\eta_j^{(k)}\rangle, \qquad k = 1, 2.$$

Since $\{|\eta_j^{(1)}\rangle\}$ and $\{|\eta_j^{(2)}\rangle\}$ are orthonormal families, there exists a unitary operator U such that $U|\eta_j^{(1)}\rangle = |\eta_j^{(2)}\rangle$ for all j. Then

$$(I_{\mathcal{N}} \otimes U)|\psi_1\rangle = \sum_j \lambda_j |\xi_j\rangle \otimes U|\eta_j^{(1)}\rangle = \sum_j \lambda_j |\xi_j\rangle \otimes |\eta_j^{(2)}\rangle = |\psi_2\rangle.$$

10.4. First, we prove (10.6). Let $A = \sum_j \lambda_j |\psi_j\rangle\langle\nu_j|$ be a singular value decomposition of A (see Problem 6.4). Recall that $\lambda_j > 0$ and $\langle\psi_j|\psi_k\rangle = \langle\nu_j|\nu_k\rangle = \delta_{jk}$. The numbers λ_j are precisely the nonzero eigenvalues of $\sqrt{A^\dagger A}$, so that $\|A\|_{\mathrm{tr}} = \sum_j \lambda_j$. Therefore

$$|\operatorname{Tr} AB| = \Big|\sum_j \lambda_j \langle\nu_j|B|\psi_j\rangle\Big| \le \sum_j \lambda_j \big|\langle\nu_j|B|\psi_j\rangle\big| \le \sum_j \lambda_j \|B\| = \|A\|_{\mathrm{tr}}\|B\|$$

for any B. On the other hand, if U is a unitary operator that takes $|\psi_j\rangle$ to $|\nu_j\rangle$, then $\operatorname{Tr} AU = \|A\|_{\mathrm{tr}}$.

To prove (10.7), suppose that $\sum_k |\xi_k\rangle\langle\eta_k| = A$, and U is the operator defined above. Then

$$\|A\|_{\mathrm{tr}} = |\operatorname{Tr} AU| = \Big|\sum_k \langle\eta_k|U|\xi_k\rangle\Big| \le \sum_k \big|\langle\eta_k|U|\xi_k\rangle\big| \le \sum_k \big\||\xi_k\rangle\big\| \, \big\||\eta_k\rangle\big\|.$$

But if we set $|\xi_k\rangle = \lambda_k|\psi_k\rangle$ and $|\eta_k\rangle = |\nu_k\rangle$, then $\sum_k \big\||\xi_k\rangle\big\| \, \big\||\eta_k\rangle\big\| = \|A\|_{\mathrm{tr}}$.

Finally, we prove that $\|\cdot\|_{\mathrm{tr}}$ is a norm. The only nontrivial property is the triangle inequality. It can be derived as follows:

$$\begin{aligned}\|A_1 + A_2\|_{\mathrm{tr}} &= \max_{U\in\mathbf{U}(\mathcal{N})} \big|\operatorname{Tr}(A_1 + A_2)U\big| \\ &\le \max_{U\in\mathbf{U}(\mathcal{N})} |\operatorname{Tr} A_1 U| + \max_{U\in\mathbf{U}(\mathcal{N})} |\operatorname{Tr} A_2 U| = \|A_1\|_{\mathrm{tr}} + \|A_2\|_{\mathrm{tr}}.\end{aligned}$$

10.5. Property a):

$$\|AB\|_{\mathrm{tr}} = \max_{U\in\mathbf{U}(\mathcal{N})} |\operatorname{Tr} ABU| \le \max_{U\in\mathbf{U}(\mathcal{N})} \|A\|_{\mathrm{tr}}\|BU\| \le \|A\|_{\mathrm{tr}}\|B\|.$$

Property b) is a special case of c), so we prove c). Let $A \in \mathcal{N}\otimes\mathcal{M}$; then

$$\|\operatorname{Tr}_{\mathcal{M}} A\|_{\mathrm{tr}} = \max_{U\in\mathbf{U}(\mathcal{N})} \big|\operatorname{Tr}\big((\operatorname{Tr}_{\mathcal{M}} A)U\big)\big| = \max_{U\in\mathbf{U}(\mathcal{N})} \big|\operatorname{Tr}\big(A(U\otimes I_{\mathcal{M}})\big)\big| \le \|A\|_{\mathrm{tr}}.$$

Property d):

$$\|A\otimes B\|_{\mathrm{tr}} = \operatorname{Tr}\sqrt{(A\otimes B)^\dagger(A\otimes B)} = \operatorname{Tr}\Big(\sqrt{A^\dagger A}\otimes\sqrt{B^\dagger B}\Big) = \|A\|_{\mathrm{tr}}\|B\|_{\mathrm{tr}}.$$

10.6. a) Let $|\xi\rangle$ and $|\eta\rangle$ be two unit vectors, $a = \langle\xi|\eta\rangle$. We can represent $|\eta\rangle$ as $a|\xi\rangle + \sqrt{1-|a|^2}|\psi\rangle$, where $|\psi\rangle$ is another unit vector orthogonal to $|\xi\rangle$. Hence

$$\big\||\xi\rangle - |\eta\rangle\big\|^2 = |1-a|^2 + (1-|a|^2) = 2(1-\operatorname{Re} a).$$

In the definition of the fidelity distance, one can multiply $|\eta\rangle$ by an arbitrary phase factor without leaving the minimization domain. This corresponds to multiplying a by the same factor. Therefore the minimum is achieved when a is real, nonnegative and the largest possible, i.e., $a = \sqrt{F(\rho,\gamma)}$.

b) Let $\mathcal{F} = \mathcal{N}^*$, so that the vectors $|\xi\rangle, |\eta\rangle \in \mathcal{N}\otimes\mathcal{N}^*$ can be associated with operators $X, Y \in \mathbf{L}(\mathcal{N})$ (due to the isomorphism $\mathcal{N}\otimes\mathcal{N}^* \cong \mathbf{L}(\mathcal{N})$). The condition $\operatorname{Tr}_{\mathcal{F}}(|\xi\rangle\langle\xi|) = \rho$ becomes $XX^\dagger = \rho$. One solution to this equation is $X = \sqrt{\rho}$; the most general solution is $X = \sqrt{\rho}\,U$, where U is an arbitrary unitary operator (cf. Problem 10.3). Similarly, $Y = \sqrt{\gamma}\,V$, where V is unitary. Thus

$$\langle\xi|\eta\rangle = \operatorname{Tr}(X^\dagger Y) = \operatorname{Tr}\big(U^\dagger\sqrt{\rho}\sqrt{\gamma}\,V\big) = \operatorname{Tr}\big(\sqrt{\rho}\sqrt{\gamma}\,W\big), \quad \text{where } W = VU^\dagger,$$
$$F(\rho,\gamma) = \max_{W\in\mathbf{U}(\mathcal{N})}\big|\operatorname{Tr}\big(\sqrt{\rho}\sqrt{\gamma}W\big)\big|^2 = \big\|\sqrt{\rho}\sqrt{\gamma}\big\|_{\mathrm{tr}}^2.$$

c) Let $|\xi\rangle$ and $|\eta\rangle$ provide the maximum in (10.8). Then

$$\begin{aligned}\|\rho-\gamma\|_{\mathrm{tr}} &= \big\|\operatorname{Tr}_{\mathcal{F}}\big(|\xi\rangle\langle\xi| - |\eta\rangle\langle\eta|\big)\big\|_{\mathrm{tr}}\\ &\le \big\||\xi\rangle\langle\xi| - |\eta\rangle\langle\eta|\big\|_{\mathrm{tr}} = 2\sqrt{1-\big|\langle\xi|\eta\rangle\big|^2} = 2\sqrt{1-F(\rho,\gamma)}.\end{aligned}$$

Thus $F(\rho,\gamma) \le 1 - \frac{1}{4}\|\rho-\gamma\|_{\mathrm{tr}}^2$, which is the required upper bound for the fidelity.

To obtain the lower bound, we will need the following lemma.

Lemma. *Let X and Y be nonnegative Hermitian operators. Then*

$$\operatorname{Tr}(X-Y)^2 \le \|X^2 - Y^2\|_{\mathrm{tr}}.$$

Proof. Let $|\psi_j\rangle$, λ_j be orthonormal eigenvectors and the corresponding eigenvalues of the operator $X-Y$. We have the following bound:

$$\|X^2-Y^2\|_{\mathrm{tr}} \ge \sum_j\big|\langle\psi_j|(X^2-Y^2)|\psi_j\rangle\big|.$$

(Indeed, the right-hand side can be represented as $\mathrm{Tr}((X^2-Y^2)U)$, where $U=\sum_j \pm|\psi_j\rangle\langle\psi_j|$.) To proceed, we need to estimate each term in the sum,

$$\begin{aligned}&\langle\psi_j|(X^2-Y^2)|\psi_j\rangle\\ &\qquad=\frac{1}{2}\langle\psi_j|(X-Y)(X+Y)|\psi_j\rangle+\frac{1}{2}\langle\psi_j|(X+Y)(X-Y)|\psi_j\rangle\\ &\qquad=\lambda_j\langle\psi_j|(X+Y)|\psi_j\rangle,\\ &\langle\psi_j|(X+Y)|\psi_j\rangle\ge|\lambda_j|.\end{aligned}$$

Thus

$$\sum_j\bigl|\langle\psi_j|(X^2-Y^2)|\psi_j\rangle\bigr|\ge\sum_j\lambda_j^2=\mathrm{Tr}(X-Y)^2.$$

□

Now we use the lemma:

$$\begin{aligned}\sqrt{F(\rho,\gamma)}&=\bigl\|\sqrt{\rho}\sqrt{\gamma}\bigr\|_{\mathrm{tr}}\ge\mathrm{Tr}\bigl(\sqrt{\rho}\sqrt{\gamma}\bigr)\\ &=1-\frac{1}{2}\,\mathrm{Tr}\bigl(\sqrt{\rho}-\sqrt{\gamma}\bigr)^2\ge1-\frac{\|\rho-\gamma\|_{\mathrm{tr}}}{2}.\end{aligned}$$

S11. Problems of Section 11

11.1. We will solve this problem together with Problem 11.2 if we prove the following three things:

a) Any superoperator of type 2 or 3 (as described in the main text) has an operator sum decomposition (11.2).

b) The set of superoperators of the form (11.2) is closed under multiplication.

c) Any such superoperator can be represented as $\mathrm{Tr}_{\mathcal{F}}(V\cdot V^\dagger)$.

We proceed with the proofs.

a) Superoperators of type 3 already have the required form. For the partial trace $\mathrm{Tr}_{\mathcal{F}}$ we have the following representation:

$$\mathrm{Tr}_{\mathcal{F}}\,\rho=\sum_m W_m\rho W_m^\dagger,\qquad W_m=I_{\mathcal{N}}\otimes\langle m|:\ \mathcal{N}\otimes\mathcal{F}\to\mathcal{N},$$

where $\{|m\rangle\}$ is an orthonormal basis of $\mathcal{F}$. We note that $W_m\bigl(|j,k\rangle\bigr)=\delta_{mk}|j\rangle$ and $W_m^\dagger\bigl(|j\rangle\bigr)=|j,m\rangle$.

b) Let $T=\sum_m A_m\cdot A_m^\dagger$ and $R=\sum_k B_k\cdot B_k^\dagger$. Then

$$RT=\sum_k B_k\Bigl(\sum_m A_m\cdot A_m^\dagger\Bigr)B_k^\dagger=\sum_{k,m}(B_kA_m)\cdot(B_kA_m)^\dagger,$$

$$\sum_{k,m}(B_kA_m)^\dagger(B_kA_m)=\sum_m A_m^\dagger\Bigl(\sum_k B_k^\dagger B_k\Bigr)A_m=\sum_m A_m^\dagger A_m=I.$$

c) Let a superoperator T be decomposed into the sum (11.2) of s terms, and let $\mathcal{F}$ be an s-dimensional space with the basis vectors denoted by $|m\rangle$. The linear map

$$V = \sum_m A_m \otimes |m\rangle \ : \ \ |\xi\rangle \mapsto \sum_m A_m|\xi\rangle \otimes |m\rangle$$

is an isometric embedding since

$$V^\dagger V = \sum_{k,m} \bigl(A_k^\dagger \otimes \langle k|\bigr)\bigl(A_m \otimes |m\rangle\bigr) = \sum_{k,m} A_k^\dagger A_m \, \langle k|m\rangle = I.$$

Moreover, $T = \mathrm{Tr}_{\mathcal{F}}(V \cdot V^\dagger)$. Indeed,

$$\mathrm{Tr}_{\mathcal{F}}(V\rho V^\dagger) = \sum_{m,k} \mathrm{Tr}_{\mathcal{F}}\bigl(A_m \rho A_k^\dagger \otimes |m\rangle\langle k|\bigr) = \sum_m A_m \rho A_m^\dagger.$$

11.2. See the solution to the previous problem.

11.3.

$$\begin{aligned}
&\mathrm{Tr}_{\mathcal{F}}\bigl((U\otimes Y)\rho(U\otimes Y)^\dagger\bigr) \\
&\qquad = \mathrm{Tr}_{\mathcal{F}}\Bigl((U\otimes Y)\sum_{j,l,k,m} \rho_{jlkm}|j,l\rangle\langle k,m|\,(U\otimes Y)^\dagger\Bigr) \\
&\qquad = \sum_{j,l,k,m} \rho_{jlkm}\,\mathrm{Tr}_{\mathcal{F}}\Bigl(\bigl(U|j\rangle\langle k|U^\dagger\bigr)\otimes\bigl(Y|l\rangle\langle m|Y^\dagger\bigr)\Bigr) \\
&\qquad = \sum_{j,l,k,m} \rho_{jlkm}\bigl(U|j\rangle\langle k|U^\dagger\bigr)\,\underbrace{\mathrm{Tr}\bigl(Y|l\rangle\langle m|Y^\dagger\bigr)}_{\delta_{lm}} \\
&\qquad = U(\mathrm{Tr}_{\mathcal{F}}\,\rho)U^\dagger.
\end{aligned}$$

11.4. The physical realizability of T is equivalent to the existence of a decomposition $T = \sum_m A_m \cdot A_m^\dagger$ such that $\sum_m A_m^\dagger A_m = I$. In the coordinate form, these equations read as follows:

$$\begin{aligned}
T_{(j'j)(k'k)} &= \bigl\langle j'\bigr|\, T(|j\rangle\langle k|)\,\bigl|k'\bigr\rangle \\
&= \sum_m \langle j'|A_m|j\rangle\,\langle k|A_m^\dagger|k'\rangle = \sum_m a_{m(j'j)}a^*_{m(k'k)}, \qquad \text{(S11.1)}
\end{aligned}$$

$$\sum_{m,l} a^*_{m(lk)}a_{m(lj)} = \delta_{kj}, \qquad \text{(S11.2)}$$

where $a_{m(j'j)} = \langle j'|A_m|j\rangle$. If we replace each index pair in parentheses by a single index, (S11.1) becomes $T_{JK} = \sum_m a_{mJ}a^*_{mK}$. This is a general form of a nonnegative Hermitian matrix; therefore (S11.1) is equivalent to conditions b) and c) in question. Equation (S11.2) is equivalent to condition a), provided (S11.1) is the case.

11.5. Properties a) and b) are equivalent to properties a) and b) in the previous problem.

It follows from the operator sum decomposition that a physically realizable superoperator takes nonnegative operators to nonnegative operators. On the other hand, if T is physically realizable, it has the form $T:\ \rho\mapsto \operatorname{Tr}_{\mathcal{F}}(V\rho V^\dagger)$, hence the superoperator

$$T\otimes I_{\mathbf{L}(\mathcal{G})}:\ \rho\mapsto \operatorname{Tr}_{\mathcal{F}}\big((V\otimes I_{\mathcal{G}})\rho(V\otimes I_{\mathcal{G}})^\dagger\big)$$

is also physically realizable. Therefore $T\otimes I_{\mathbf{L}(\mathcal{G})}$ takes nonnegative operators to nonnegative operators.

For the proof of the converse assertion, we will deduce property c) of the previous problem from property c) of the present problem. Specifically, we will show that the matrix (T_{JK}) (where $J=(j'j)$ and $K=(k'k)$) corresponds to an operator Y that can be represented as $(T\otimes I_{\mathbf{L}(\mathcal{G})})X$ for some nonnegative Hermitian operator $X\in\mathbf{L}(\mathcal{N}\otimes\mathcal{G})$.

Let $\dim\mathcal{G}=\dim\mathcal{N}$, and let $|j\rangle$ denote the basis vectors in both spaces. We set

$$X=\sum_{j,k}|j\rangle\langle k|\otimes|j\rangle\langle k|=|\psi\rangle\langle\psi|,\qquad \text{where } |\psi\rangle=\sum_j|j\rangle\otimes|j\rangle.$$

Then

$$Y=(T\otimes I_{\mathbf{L}(\mathcal{G})})X=\sum_{j',j,k',k}T_{(j'j)(k'k)}\,|j'\rangle\langle k'|\otimes|j\rangle\langle k|,$$

hence $\langle j',j|Y|k',k\rangle=T_{(j'j)(k'k)}$.

11.6. Let us represent T in the form $T=\operatorname{Tr}_{\mathcal{F}'}(V\rho V^\dagger)$, where $V:\mathcal{N}\to\mathcal{N}\otimes\mathcal{F}\otimes\mathcal{F}'$ is a unitary embedding (cf. Problem 11.1). By assumption, for any pure state $|\xi\rangle\in\mathcal{N}$ we have

$$\operatorname{Tr}_{\mathcal{F}\otimes\mathcal{F}'}\big(V|\xi\rangle\langle\xi|V^\dagger\big)=\operatorname{Tr}_{\mathcal{F}}\big(\operatorname{Tr}_{\mathcal{F}'}\big(V|\xi\rangle\langle\xi|V^\dagger\big)\big)=\operatorname{Tr}_{\mathcal{F}}\big(T(|\xi\rangle\langle\xi|)\big)=|\xi\rangle\langle\xi|.$$

Therefore $|\psi\rangle=V|\xi\rangle$ is a product state: $V|\xi\rangle=|\xi\rangle\otimes|\eta\rangle$ (this follows from the observation made after the formulation of Problem 10.2). *A priori*, $|\eta\rangle$ depends on the vector $|\xi\rangle$, but the linearity of V implies that $|\eta\rangle$ is actually constant. Therefore $TX=X\otimes\gamma$, where $\gamma=\operatorname{Tr}_{\mathcal{F}'}(|\eta\rangle\langle\eta|)$.

11.7. A superoperator T of the type $\mathbf{L}(\mathcal{N})\to\mathbf{L}(\mathcal{N})\times\{1,\dots,r\}$ has the form $T=\sum_m T_m\otimes|m\rangle\langle m|$, where $T_m:\mathbf{L}(\mathcal{N})\to\mathbf{L}(\mathcal{N})$. If T is physically realizable, then it satisfies conditions a)–c) of Problem 11.5, hence each superoperator T_m satisfies conditions b) and c). If T is consistent with (11.4), then $T_m(|\xi\rangle\langle\xi|)=\delta_{mj}|\xi\rangle\langle\xi|$ for any $|\xi\rangle\in\mathcal{L}_j$; this condition extends by linearity as follows:

$$T_mX=\delta_{mj}X\qquad\text{for any } X\in\mathbf{L}(\mathcal{L}_j). \tag{S11.3}$$

Under these assumptions, we will prove that $T_m = \Pi_{\mathcal{L}_m} \cdot \Pi_{\mathcal{L}_m}$.

Let $\bigl\{|\xi_{j,p}\rangle : p = 1, \ldots, \dim \mathcal{L}_j\bigr\}$ be an orthonormal basis of the space $\mathcal{L}_j$. Formula (S11.3) determines the value of $T_m(|\xi_{j,p}\rangle\langle\xi_{k,s}|)$ in the case $j = k$. Therefore it is sufficient to prove that if $j \ne k$, then $T_m(|\xi_{j,p}\rangle\langle\xi_{k,s}|) = 0$.

Let us fix m, j, p, k, s and denote the operator $T_m(|\xi_{j,p}\rangle\langle\xi_{k,s}|)$ by A. It suffices to prove that $\operatorname{Tr} AB = 0$ for any B or, equivalently, for any B of the form $|\eta\rangle\langle\eta|$. Let $a = \operatorname{Tr}(A|\eta\rangle\langle\eta|) = \bigl\langle\eta\bigl|T_m(|\xi_{j,p}\rangle\langle\xi_{k,s}|)\bigr|\eta\bigr\rangle$. Consider the function

$$f(x,y) \stackrel{\text{def}}{=} \bigl\langle\eta\bigl|T_m(|\psi\rangle\langle\psi|)\bigr|\eta\bigr\rangle, \quad \text{where } |\psi\rangle = x|\xi_{j,p}\rangle + y|\xi_{k,s}\rangle,$$
$$f(x,y) = \delta_{mj}|\langle\eta|\xi_{j,p}\rangle|^2\,|x|^2 + axy^* + a^*x^*y + \delta_{mk}|\langle\eta|\xi_{k,s}\rangle|^2\,|y|^2.$$

Since the operator $T_m(|\psi\rangle\langle\psi|)$ is nonnegative, $f(x,y) \ge 0$ for any x and y. But the condition $j \ne k$ implies that $\delta_{mj} = 0$ or $\delta_{mk} = 0$. Therefore $a = 0$.

11.8. In formula (11.7), let k run from 1 to r, whereas $\rho \in \mathbf{L}(\mathcal{N})$. We define the "larger space" to be $\mathcal{N}' = \mathcal{N} \otimes \mathcal{M}$, where $\mathcal{M} = \mathbb{C}\bigl(|1\rangle, \ldots, |r\rangle\bigr)$. The isometric embedding $V : \mathcal{N} \to \mathcal{N}'$ (which takes ρ to $\gamma = V\rho V^\dagger$) and the subsequent projective measurement $\gamma \mapsto \sum_j \operatorname{Tr}(\gamma\Pi_{\mathcal{L}_j}) \cdot (j)$ are defined by the formulas

$$V|\xi\rangle = \sum_k \sqrt{X_k}|\xi\rangle \otimes |k\rangle, \qquad \mathcal{L}_j = \mathcal{N} \otimes \mathbb{C}(|j\rangle).$$

It is clear that $\Pi_{\mathcal{L}_j} = I_{\mathcal{N}} \otimes |j\rangle\langle j|$, hence $\operatorname{Tr}(V\rho V^\dagger\Pi_{\mathcal{L}_j}) = \operatorname{Tr}(\rho X_j)$.

11.9. We assume that the measurement is destructive (which is indicated in Figure S11.1 by the measured qubits being discarded to the trash bin). Thus the measurement is the following transformation of two quantum bits into two classical bits:

$$T : \rho \mapsto \sum_{a,b} \langle\xi_{ab}|\rho|\xi_{ab}\rangle \cdot (a,b).$$

To realize the transformation T, Alice applies the unitary operator

$$H[1]\,\Lambda(\sigma^x)[1,2] : \ |\xi_{ab}\rangle \mapsto |b,a\rangle,$$

interchanges the qubits and measures them in the basis $\{|0\rangle, |1\rangle\}$. Then she sends the measurement results to Bob.

Suppose the initial state of the first qubit was ρ. After the measurement, the overall state of the system becomes

$$\gamma = \sum_{a,b}\bigl(a,b,\gamma_{ab}\bigr) = \sum_{a,b} p_{ab} \cdot \bigl(a,b,\,p_{ab}^{-1}\gamma_{ab}\bigr), \qquad p_{ab} = \operatorname{Tr}\gamma_{ab},$$

$$\gamma_{ab} = (T \otimes I_{\mathbf{L}(\mathcal{B})})\Bigl(\rho \otimes |\xi_{00}\rangle\langle\xi_{00}|\Bigr) = \bigl(\langle\xi_{ab}| \otimes I_{\mathcal{B}}\bigr)\Bigl(\rho \otimes |\xi_{00}\rangle\langle\xi_{00}|\Bigr)\bigl(|\xi_{ab}\rangle \otimes I_{\mathcal{B}}\bigr).$$

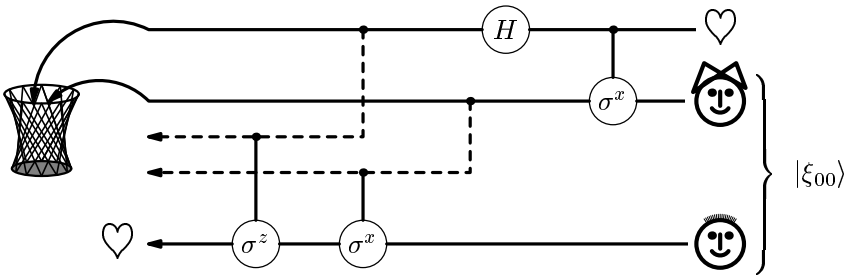


Fig. S11.1. The circuit for quantum teleportation. The ♡ symbol indicates the quantum state being teleported. 😺 and 😼 denote Alice's and Bob's halves of the auxiliary state $|\xi_{00}\rangle$. The dashed lines represent classical bits sent by Alice to Bob.

Here p_{ab} is the probability to get the measurement outcome (a, b), whereas $p_{ab}^{-1}\gamma_{ab}$ is the corresponding conditional quantum state. Note that $\langle\xi_{ab}|$ and $|\xi_{ab}\rangle$ are regarded as operators of types $\mathcal{B}^{\otimes 2} \to \mathbb{C}$ and $\mathbb{C} \to \mathcal{B}^{\otimes 2}$ (resp.), so that $\langle\xi_{ab}| \otimes I_{\mathcal{B}} : \mathcal{B}^{\otimes 3} \to \mathcal{B}$ and $|\xi_{ab}\rangle \otimes I_{\mathcal{B}} : \mathcal{B} \to \mathcal{B}^{\otimes 3}$.

We now describe Bob's actions aimed at the recovery of the initial state ρ. Without loss of generality we may assume that $\rho = |\psi\rangle\langle\psi|$. (Indeed, the whole process of measurement and recovery can be described by a superoperator. If it preserves pure states, it also preserves mixed states, due to the linearity.) In this case the state after the measurement is also pure: $\gamma_{ab} = |\psi_{ab}\rangle\langle\psi_{ab}|$. Let $|\psi\rangle = z_0|0\rangle + z_1|1\rangle$; then

$$
\begin{aligned}
|\psi_{ab}\rangle &= \big(\langle\xi_{ab}| \otimes I_{\mathcal{B}}\big)\Big(|\psi\rangle \otimes |\xi_{00}\rangle\Big) = \big(\langle\xi_{ab}| \otimes I_{\mathcal{B}}\big)\left(\sum_{c,d} z_c|c\rangle \otimes \frac{1}{\sqrt{2}}|d, d\rangle\right) \\
&= \frac{1}{\sqrt{2}}\sum_{c,d} z_c\langle\xi_{ab}|c, d\rangle\,|d\rangle = \frac{1}{2}\sum_{c,d} z_c(-1)^{bc}\delta_{c\oplus a,d}\,|d\rangle \\
&= \frac{1}{2}\sum_{c} z_c(-1)^{bc}\,|a \oplus c\rangle = \frac{1}{2}(\sigma^x)^a(\sigma^z)^b|\psi\rangle.
\end{aligned}
$$

Note that the probability $p_{ab} = \langle\psi_{ab}|\psi_{ab}\rangle = \frac{1}{4}$ does not depend on the initial state $|\psi\rangle$. (In fact, if it depended on $|\psi\rangle$, then the recovery would not be possible; this follows from the result of Problem 11.6.) To restore the initial state, Bob applies the operators σ^x and σ^z with classical control: the controlling parameters are the measured values of a and b. The result is as follows:

$$
(\sigma^z)^b(\sigma^x)^a\frac{|\psi_{ab}\rangle}{\sqrt{p_{ab}}} = |\psi\rangle.
$$

Remark. One may ask this question: what if there is some other quantum system, which is not involved in the teleportation procedure, but the qubit being teleported is initially entangled with it? Will the state be preserved in this case? The answer is "yes". Indeed, we have proved that the measurement followed by the recovery effects the superoperator $R = I_{\mathbf{L}(\mathcal{B})}$ on the teleported qubit. When the additional system is taken into account, the superoperator becomes $R \otimes I_{\mathbf{L}(\mathcal{G})} = I_{\mathbf{L}(\mathcal{B}\otimes\mathcal{G})}$.

11.10. To express the fidelity $F\big(\mathrm{Tr}_{\mathcal{M}}(A\rho A^\dagger), \mathrm{Tr}_{\mathcal{M}}(B\gamma B^\dagger)\big)$, we start with some purifications of ρ and γ over the auxiliary space $\mathcal{G} = \mathcal{N}^*$, i.e., $\rho = \mathrm{Tr}_{\mathcal{G}}(|\xi\rangle\langle\xi|)$, $\gamma = \mathrm{Tr}_{\mathcal{G}}(|\eta\rangle\langle\eta|)$, where $|\xi\rangle, |\eta\rangle \in \mathcal{N}\otimes\mathcal{G}$. The states

$$|\xi'\rangle = (A\otimes I_{\mathcal{G}})|\xi\rangle, \qquad |\eta'\rangle = (B\otimes I_{\mathcal{G}})|\eta\rangle, \qquad\qquad |\xi'\rangle,\, |\eta'\rangle \in \mathcal{F}\otimes\mathcal{M}\otimes\mathcal{G}$$

are particular purifications of $\rho' = \mathrm{Tr}_{\mathcal{M}}(A\rho A^\dagger)$ and $\gamma' = \mathrm{Tr}_{\mathcal{M}}(B\gamma B^\dagger)$ ($\rho', \gamma' \in \mathbf{L}(\mathcal{F})$) over the auxiliary space $\mathcal{M}\otimes\mathcal{G}$. The fidelity can be defined in terms of general purifications over the same space.[1] As follows from the result of Problem 10.3, all purifications of ρ' and γ' have the form $(I_{\mathcal{F}}\otimes U)|\xi'\rangle$ or $(I_{\mathcal{F}}\otimes V)|\eta'\rangle$ for some unitary U and V. Therefore

$$\begin{aligned}
\sqrt{F(\rho',\gamma')} &= \max_{U,V\in\mathbf{U}(\mathcal{M}\otimes\mathcal{G})} \big|\langle\eta'|(I_{\mathcal{F}}\otimes V^\dagger)(I_{\mathcal{F}}\otimes U)|\xi'\rangle\big| \\
&= \max_{W\in\mathbf{U}(\mathcal{M}\otimes\mathcal{G})} \big|\langle\eta'|I_{\mathcal{F}}\otimes W|\xi'\rangle\big| \\
&= \max_{W\in\mathbf{U}(\mathcal{M}\otimes\mathcal{G})} \big|\big\langle\eta\big|(B^\dagger\otimes I_{\mathcal{G}})(I_{\mathcal{F}}\otimes W)(A\otimes I_{\mathcal{G}})\big|\xi\big\rangle\big| \\
&= \max_{W\in\mathbf{U}(\mathcal{M}\otimes\mathcal{G})} \Big|\mathrm{Tr}\Big(W\,\mathrm{Tr}_{\mathcal{F}}\big((A\otimes I_{\mathcal{G}})|\xi\rangle\langle\eta|(B^\dagger\otimes I_{\mathcal{G}})\big)\Big)\Big| \\
&= \max_{W\in\mathbf{U}(\mathcal{M}\otimes\mathcal{G})} \Big|\mathrm{Tr}\Big(W(T\otimes I_{\mathbf{L}(\mathcal{G})})(|\xi\rangle\langle\eta|)\Big)\Big| = \big\|(T\otimes I_{\mathbf{L}(\mathcal{G})})(|\xi\rangle\langle\eta|)\big\|_{\mathrm{tr}}.
\end{aligned}$$

The fidelity should be maximized over ρ and γ, which is equivalent to maximizing the last expression over all unit vectors $|\xi\rangle$ and $|\eta\rangle$. On the other hand, Theorem 11.1 and formula (10.7) imply that

$$\begin{aligned}
\|T\|_\diamond = \|T\otimes I_{\mathbf{L}(\mathcal{G})}\|_1 &= \sup_{A\neq 0} \frac{\|(T\otimes I_{\mathbf{L}(\mathcal{G})})A\|_{\mathrm{tr}}}{\|A\|_{\mathrm{tr}}} \\
&= \sup_{|\xi_k\rangle,|\eta_k\rangle} \frac{\big\|(T\otimes I_{\mathbf{L}(\mathcal{G})})\sum_k |\xi_k\rangle\langle\eta_k|\big\|_{\mathrm{tr}}}{\sum_k \big\||\xi_k\rangle\big\|\,\big\||\eta_k\rangle\big\|} \\
&= \max\Big\{\Big\|(T\otimes I_{\mathbf{L}(\mathcal{G})})(|\xi\rangle\langle\eta|)\Big\|_{\mathrm{tr}} :\ \big\||\xi\rangle\big\| = \big\||\eta\rangle\big\| = 1\Big\}.
\end{aligned}$$

The two results agree.

[1] The definition applies directly only if $\dim\mathcal{F} = (\dim\mathcal{N})(\dim\mathcal{M})$. As far as the general case is concerned, we refer to the remark after formula (10.8).

S12. Problems of Section 12

12.1. We will use the following simple property of the operator norm: if $X_j \in \mathbf{L}(\mathcal{N}_j, \mathcal{M}_j)$ and $X = \bigoplus_j X_j : \bigoplus_j \mathcal{N}_j \to \bigoplus_j \mathcal{M}_j$, then $\|X\| = \max_j \|X_j\|$. Indeed, the set of the eigenvalues for $X^\dagger X$ is the union of the corresponding sets for $X_j^\dagger X_j$.

Let $V : \mathcal{K} \to \mathcal{K} \otimes \mathcal{B}^{\otimes N}$ be the standard embedding, i.e., $V|\xi\rangle = |\xi\rangle \otimes |0^N\rangle$. Then $\|\tilde{U}_j V - V U_j\| \le \delta$ for each j. Therefore

$$\Big\|\tilde{W}\big(I_{\mathcal{N}} \otimes V\big) - \big(I_{\mathcal{N}} \otimes V\big)W\Big\| = \Big\|\bigoplus_j \Pi_{\mathcal{L}_j} \otimes \big(\tilde{U}_j V - V U_j\big)\Big\| \le \delta.$$

12.2. We have

$$\tilde{U} = \sum_{j \in \Omega} \Pi_{\mathcal{L}_j} \otimes P_j, \qquad P_j = \sum_{y \in \Delta} Q_y \otimes \big(R_j^\dagger \Pi_{\mathcal{M}_y} R_j\big).$$

Due to the result of Problem 12.1, it is sufficient to show that P_j approximates $Q_{f(j)}$ for each j.

Let $|\xi\rangle \in \mathcal{K}$. We need to estimate the norm of the vector $|\psi\rangle = |\tilde{\eta}\rangle - |\eta\rangle \otimes |0^N\rangle$, where $|\eta\rangle = Q_{f(j)}|\xi\rangle$ and $|\tilde{\eta}\rangle = P_j\big(|\xi\rangle \otimes |0^N\rangle\big)$. Such an estimate follows from the calculation

$$\begin{aligned}
\langle\psi|\psi\rangle &= 2 - \big(\langle\eta| \otimes \langle 0^N|\big)\,|\tilde{\eta}\rangle - \langle\tilde{\eta}|\,\big(|\eta\rangle \otimes |0^N\rangle\big) = 2 - 2\mathrm{Re}\big(\langle\eta| \otimes \langle 0^N|\big)\,|\tilde{\eta}\rangle \\
&= 2 - 2\mathrm{Re} \sum_{y \in \Delta} \langle\xi|Q_{f(j)}^\dagger Q_y|\xi\rangle\, \langle 0^N|R_j^\dagger \Pi_{\mathcal{M}_y} R_j|0^N\rangle \\
&= 2 \sum_{y \in \Delta} \Big(1 - \mathrm{Re}\langle\xi|Q_{f(j)}^\dagger Q_y|\xi\rangle\Big)\, \mathbf{P}(y|j) \\
&\le 2 \sum_{y \ne f(j)} 2\mathbf{P}(y|j) \;\le\; 4\varepsilon.
\end{aligned}$$

Thus $\big\||\psi\rangle\big\| \le 2\sqrt{\varepsilon}$.

In the case where V is the copy operator, $Q_{f(j)}^\dagger Q_y = \delta_{y,f(j)} I_{\mathcal{K}}$, so we get 2ε instead of 4ε in the above estimate. Hence $\big\||\psi\rangle\big\| \le \sqrt{2\varepsilon}$.

S13. Problems of Section 13

13.1. We denote the required probability by $p(X, l)$. If $h_1, \ldots, h_l$ do not generate the whole group X, then they are contained in some maximal proper subgroup $Y \subset X$. For each Y the probability of such an event does not exceed 2^{-l}, because $|Y| \le |X|/2$. Therefore $p(X, l) \ge 1 - K(X) \cdot 2^{-l}$, where $K(X)$ is the number of maximal proper subgroups of the group X. Subgroups of an Abelian group X are in one-to-one correspondence with subgroups of the character group X^*; maximal proper subgroups correspond

to minimal nonzero subgroups. Each minimal nonzero subgroup is generated by a single nonzero element, so that $K(X) < |X^*| = |X|$.

13.2. We construct a classical operator $V_b \in \mathbf{L}(\mathcal{B} \otimes \mathcal{B}^{\otimes n})$ (the basis vectors in $\mathcal{B}^{\otimes n}$ are numbered from 0 to $2^n - 1$) such that

$$V_b|0,0\rangle = |0,1\rangle, \qquad V_b|1,0\rangle = |1,b\rangle.$$

Then the circuit $V_b^{-1}[0,B]\; U[B,A]\; V_b[0,B]$ realizes the operator $\Lambda(U_b)[0,A]$, where B denotes a set of n ancillas.

13.3. Let us calculate the expected value of $\exp\bigl(h\bigl(\sum_{r=1}^s v_r - sp\bigr)\bigr)$ and choose h so that to minimize the result:

$$\mathbf{E}\Bigl[\exp\Bigl(h\Bigl(\sum_{r=1}^s v_r - sp\Bigr)\Bigr)\Bigr] = \Bigl(e^{-hp}\bigl((1-p_*) + p_* e^h\bigr)\Bigr)^s = \exp\bigl(-H(p,p_*)s\bigr)$$

for $h = \ln\frac{p}{1-p} - \ln\frac{p_*}{1-p_*}$, where $H(p,p_*) = (1-p)\ln\frac{1-p}{1-p_*} + p\ln\frac{p}{p_*}$. Now we use the obvious inequality $\mathbf{E}[e^f] \ge \mathbf{Pr}[f \ge 0]$, which holds for any random variable f. Note that $h \ge 0$ if $p \ge p_*$, and $h \le 0$ if $p \le p_*$. Thus we get the inequalities

$$\mathbf{Pr}\Bigl[s^{-1}\sum_{r=1}^s v_r \ge p\Bigr] \;\le\; \exp\bigl(-H(p,p_*)s\bigr) \quad \text{if } p \ge p_*,$$
$$\mathbf{Pr}\Bigl[s^{-1}\sum_{r=1}^s v_r \le p\Bigr] \;\le\; \exp\bigl(-H(p,p_*)s\bigr) \quad \text{if } p \le p_*.$$

This is a sharper version of Chernoff's bound (cf. [**61**]).

The function $H(p,p_*)$ (called the relative entropy) can be represented as $(1-p)\ln\frac{1}{1-p_*} + p\ln\frac{1}{p_*} - H(p)$, where $H(p) = (1-p)\ln\frac{1}{1-p} + p\ln\frac{1}{p}$ is the entropy of the probability distribution ($w_0 = 1-p$, $w_1 = p$). It is easy to check that

$$H(p_*,p_*) = 0, \quad \frac{\partial H(p,p_*)}{\partial p}\Big|_{p=p_*} = 0, \quad \frac{\partial^2 H(p,p_*)}{\partial p^2} = -\frac{\partial^2 H(p)}{\partial p^2} \ge 4,$$

hence $H(p,p_*) \ge 2(p-p_*)^2$. The inequality (13.4) follows.

13.4. The Fourier transform operator $F = F_q$ acts on $n = \lceil \log_2 q \rceil$ qubits; more precisely, it acts on the subspace $\mathcal{N} = \mathbb{C}\bigl(|0\rangle, \dots, |q-1\rangle\bigr) \subseteq \mathcal{B}^{\otimes n}$. Let

$$|\psi_x\rangle = \frac{1}{\sqrt{q}} \sum_{y=0}^{q-1} \exp\left(-2\pi i \frac{xy}{q}\right) |y\rangle, \qquad x = 0, \dots, q-1.$$

These are the eigenvectors of the operator $X : |y\rangle \mapsto |(y+1) \bmod q\rangle$, the corresponding eigenvalues being equal to $\lambda_x = e^{2\pi i \varphi_x}$, $\varphi_x = x/q$. Obviously,

$F|x\rangle = |\psi_{-x}\rangle$. The Fourier transform operator is also characterized by its action on the vectors $|\psi_x\rangle$: $F|\psi_x\rangle = |x\rangle$.

Thus, we need to transform $|\psi_x\rangle$ into $|x\rangle$. The general scheme of the solution is as follows:

$$|\psi_x\rangle \otimes |0\rangle \stackrel{W}{\longmapsto} |\psi_x\rangle \otimes |x\rangle \stackrel{\leftrightarrow}{\longmapsto} |x\rangle \otimes |\psi_x\rangle \stackrel{V}{\longmapsto} |x\rangle \otimes |\psi_0\rangle \stackrel{I\otimes U^{-1}}{\longmapsto} |x\rangle \otimes |0\rangle.$$

(The extra $|0\rangle$ in the first and the last expression corresponds to ancillas.) We will realize each of the operators W, V, U with precision $\delta' = \delta/3$.

W is a measuring operator with respect to the orthogonal decomposition of $\mathcal{N}$ into the eigenspaces of X; it performs a garbage-free measurement of the parameter x that corresponds to the eigenvalue λ_x. To realize W with precision δ', we need to measure x with error probability $\leq \varepsilon = (\delta')^2/2 = \Theta(2^{-2l})$ and remove the garbage (see Section 12.3). To measure x, we estimate the corresponding phase $\varphi_x = x/q$ with precision $2^{-(n+1)}$ (this is a different kind of precision!), multiply by q and round to an integer.

According to Theorem 13.3, the phase estimation is done by a circuit of size $O(n \log n + nl)$ and depth $O(\log n + \log l)$ with the aid of the operator

$$\Upsilon_m(X) : |p, y\rangle \mapsto |(y + p) \bmod q\rangle, \qquad p = 0, \ldots, 2^m - 1, \quad y = 0, \ldots, q-1,$$

where $m = n + k$, $k = O(\log l + \log\log n)$. Note that the operator $\Upsilon_m(X)$ itself has smaller complexity: it can be realized by a circuit of size $O(nk+k^2)$ and depth $O(\log n + (\log k)^2)$ (see Problem 2.14b). However, the multiplication of the estimated phase by q makes a significant contribution to the overall size, namely, $O(n^2)$.

The operator V acts as follows:

$$V|x, y\rangle = \exp\left(-2\pi i \frac{xy}{q}\right)|x, y\rangle.$$

To realize this operator, we compute xy/q with precision 2^{-m}, where $m = l + O(1)$. More exactly, we find a $p \in \{0, \ldots, 2^m - 1\}$ such that

$$\left|2^{-m}p - xy/q\right|_{\mathrm{mod}1} \leq 2^{-m}.$$

Then we apply the operator $\Upsilon_m(e^{2\pi i/2^m})$ controlled by this p, and uncompute p.

When estimating the value of xy/q, we operate with $\Theta(l)$-digit real numbers; therefore p is computed by a circuit of size $O(l^2 \log l)$ and depth $O((\log l)^2)$ (see Problem 2.14a). The operator $\Upsilon_m(e^{2\pi i/2^m})$ has approximately the same complexity (see Lemma 13.4). Thus V is realized by an $O(l^2 \log l)$-size, $O((\log l)^2)$-depth circuit over the standard basis.

Finally, we need to realize the operator $U : |0\rangle \mapsto |\psi_0\rangle$. If $q = 2^n$, this is very simple: $U = H^{\otimes n}$. The general case was considered in Problem 9.4a,

but the procedure proposed in the solution does not parallelize. We now describe a completely different procedure, which lends itself to parallelization. Instead of constructing the vector $|\psi_0\rangle = |\xi_{0,q}\rangle$, we will try to create the vector

$$|\xi_{a,q}\rangle = \sqrt{\frac{1-e^{-2a}}{1-e^{-2aq}}} \sum_{x=0}^{q-1} e^{-ax}|x\rangle$$

for $a = c_1 2^{-(n+l)}$, where c_1 is a suitable constant. It is obvious that

$$\big\||\xi_{a,q}\rangle - |\xi_{0,q}\rangle\big\| \le O(aq),$$

so the desired precision $\Theta(2^{-l})$ is achieved this way.

Let us consider the vector $|\xi_{a,\infty}\rangle$, which belongs to the infinite-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^{\mathbb{N}}$ (where $\mathbb{N} = \{0, 1, 2, \dots\}$). Of course, it is impossible to create this state exactly, but the m-qubit state $|\xi_{a,2^m}\rangle$ may be a good approximation, provided m is large enough. Clearly,

$$\big\||\xi_{a,r}\rangle - |\xi_{a,\infty}\rangle\big\| \le O(e^{-ar}),$$

so it suffices to choose $r = 2^m$ such that $e^{-ar} \le c_2 2^{-l}$ for a suitable constant c_2. Thus $m = \big\lceil \log_2\big(-a^{-1}\ln(c_2 2^{-l})\big)\big\rceil = n + \Theta(l)$. We will show how to construct the state $|\xi_{a,2^m}\rangle$ later.

Now, we invoke the function

$$G : \mathbb{N} \to \mathbb{N} \times \{0, \dots, q-1\}, \qquad G(x) = \big(\lfloor x/q \rfloor, (x \bmod q)\big)$$

and the corresponding linear map $\widehat{G_q} : \mathcal{H} \to \mathcal{H} \otimes \mathbb{C}\big(|0\rangle, \dots, |q-1\rangle\big)$. It transforms the state $|\xi_{a,\infty}\rangle$ into $|\xi_{aq,\infty}\rangle \otimes |\xi_{a,q}\rangle$. Thus, the state $|\xi_{a,q}\rangle$ can be obtained as follows: we create $|\xi_{a,\infty}\rangle$, split it into $|\xi_{a,q}\rangle$ and $|\xi_{aq,\infty}\rangle$, and get rid of the last state (which is as difficult as creating it).

In the computation, we must replace the operator $\widehat{G_q}$ by its finite version, $|x, 0^n\rangle \mapsto \big|\lfloor x/q \rfloor, (x \bmod q)\big\rangle$, where $x \in \{0, \dots, 2^m - 1\}$. Note that the ratio x/q ranges from 0 to $2^{m-n+1} = 2^{O(l)}$, hence the corresponding circuit has size $O(nl + l^2 \log l)$ and depth $O(\log n + (\log l)^2)$.

Thus, it remains to show how to create the state $|\xi_{b,2^m}\rangle$ for $b = a$ and $b = aq$. This is not hard because $|\xi_{b,2^m}\rangle$ is a product state:

$$|\xi_{b,2^m}\rangle = |\eta(\theta_{m-1})\rangle \otimes \dots \otimes |\eta(\theta_0)\rangle, \qquad |\eta(\theta)\rangle = \cos(\pi\theta)|0\rangle + \sin(\pi\theta)|1\rangle,$$

where $\theta_j = (1/\pi)\arctan(e^{-2^m b})$. Each vector $|\eta(\theta_j)\rangle$ is obtained from $|0\rangle$ as follows:

$$|\eta(\theta_j)\rangle = \exp(-i\pi\theta_j\sigma^y)|0\rangle = K^{-1}H\exp(i\pi\theta_j\sigma^z)H\,|0\rangle.$$

Moreover, all these vectors can be easily constructed at once by a circuit over the standard basis. Indeed,

$$\exp(i\pi\theta_{m-1}\sigma^z)\otimes\cdots\otimes\exp(i\pi\theta_0\sigma^z)\,|x_{m-1},\dots,x_0\rangle$$
$$=\exp\left(2\pi i\sum_{j=0}^{m-1}(1/2-x_j)\theta_j\right)|x_{m-1},\dots,x_0\rangle.$$

Therefore we just need to compute the sum in the exponent with precision $\Theta(2^{-l})$, and proceed by analogy with the operator V.

The circuits for the creation of the vectors $|\xi_{a,2^m}\rangle$ and $|\xi_{aq,2^m}\rangle$ have size $O(nl+l^2\log l)$ and depth $O(\log n+(\log l)^2)$. This estimate does not include the complexity of computing the numbers θ_j because such computation belongs to the pre-processing stage. We mention, however, that it can be done in time $\mathrm{poly}(l)$.

To summarize, the Fourier transform operator F_q is realized by a circuit with the the following parameters:

$$\text{size}=O(n^2+l^2\log l),\qquad \text{depth}=O(\log n+(\log l)^2).$$

S15. Problems of Section 15

15.1. We represent the total Hilbert space in the form $\mathcal{B}^{\otimes n}=\mathcal{K}\otimes\mathcal{L}$, where $\mathcal{K}$ is the state space of qubits in A, and $\mathcal{L}$ is the state space of the remaining qubits. We need to reconstruct the state ρ from $\mathrm{Tr}_{\mathcal{K}}\,\rho$.

Let us write an operator sum decomposition for the superoperator $T:\rho\mapsto\mathrm{Tr}_{\mathcal{K}}\,\rho$ (cf. Problem 11.2):

$$T=\sum_m A_m\cdot A_m^\dagger,\qquad A_m=\langle m|\otimes I_{\mathcal{L}}:\ \mathcal{K}\otimes\mathcal{L}\to\mathcal{L},$$

where $\{|m\rangle\}$ is an orthonormal basis of $\mathcal{K}$. We see that $T\in\mathcal{D}\cdot\mathcal{D}^\dagger$, where

$$\mathcal{D}=\mathcal{K}^*\otimes I_{\mathcal{L}}\subseteq\mathbf{L}(\mathcal{K}\otimes\mathcal{L},\,\mathcal{L}).$$

If $X,Y\in\mathcal{D}$, then $Y^\dagger X\in\mathbf{L}(\mathcal{K})\otimes I_{\mathcal{L}}=\mathcal{E}(A)$. Consequently, the code $\mathcal{M}$ corrects errors from $\mathcal{D}$. It remains to apply Theorem 15.3.

15.2 (Cf. [**11, 41**]). Suppose that $\mathcal{M}$ is a code of type $(4,1)$ correcting one-qubit errors. Then it must detect two-qubit errors, in particular errors in qubits $1,2$ as well as in qubits $3,4$. This means that an arbitrary state $\rho\in\mathbf{L}(\mathcal{M})$ can be restored both from the first two qubits and from the last two qubits (cf. Problem 15.1). Let us show that this is impossible.

Let $\mathcal{N}_1$ be the space of qubits $1,2$, and $\mathcal{N}_2$ the space of qubits $3,4$. Then $\mathcal{M}$ is a subspace of $\mathcal{N}_1\otimes\mathcal{N}_2$. Denote the inclusion map $\mathcal{M}\to\mathcal{N}_1\otimes\mathcal{N}_2$ by V. We have assumed the existence of error-correcting transformations

— physically realizable superoperators $P_1 : \mathcal{N}_1 \to \mathcal{M}$ and $P_2 : \mathcal{N}_2 \to \mathcal{M}$ satisfying

$$P_1 \operatorname{Tr}_{\mathcal{N}_2}(V\rho V^\dagger) = P_2 \operatorname{Tr}_{\mathcal{N}_1}(V\rho V^\dagger) = \rho \quad \text{for any } \rho \in \mathcal{M}.$$

Therefore we can define a physically realizable superoperator

$$P = (P_1 \otimes P_2)(V \cdot V^\dagger) : \ \mathcal{M} \to \mathcal{M} \otimes \mathcal{M},$$

which has the following properties:

$$\operatorname{Tr}_{\mathcal{N}_2} P\rho = \operatorname{Tr}_{\mathcal{N}_2}\bigl((P_1 \otimes P_2)(V\rho V^\dagger)\bigr) = P_1 \operatorname{Tr}_{\mathcal{N}_2}(V\rho V^\dagger) = \rho,$$
$$\operatorname{Tr}_{\mathcal{N}_1} P\rho = \operatorname{Tr}_{\mathcal{N}_1}\bigl((P_1 \otimes P_2)(V\rho V^\dagger)\bigr) = P_2 \operatorname{Tr}_{\mathcal{N}_1}(V\rho V^\dagger) = \rho.$$

According to Problem 11.6, the first identity implies that $P\rho = \rho \otimes \gamma_2$, where γ_2 does not depend on ρ. Similarly, $P\rho = \gamma_1 \otimes \rho$. We have arrived at a contradiction: $\gamma_1 \otimes \rho = \rho \otimes \gamma_2$ for any ρ.

15.3. We will only give the idea of the solution. It is sufficient to examine the phase component $g^{(z)}$ of the error $g = g^{(x)} + g^{(z)}$ (the classical component $g^{(x)}$ is treated similarly). A syndrome bit equals 1 if and only if the star of the corresponding vertex contains an odd number of edges from $g^{(z)}$. Therefore we obtain the following problem. Let D be the boundary of a 1-chain C with $\mathbb{Z}_2$-coefficients (i.e., D is a set of an even number of lattice vertices); we need to find such a chain $C_{\min}$ which contains the smallest number of edges.

It is not difficult to figure out that $C_{\min}$ is the disjoint union of paths that connect pairs of vertices of the set D (two different paths cannot have a common edge). Therefore the problem of determining the error by its syndrome is reduced to the the following *weighted matching problem*. There is a graph G (in our case, the complete graph whose vertices are the vertices of the lattice) with a weight assigned to each edge (in our case, the shortest path length on the lattice). It is required to find a perfect matching with minimal total weight. (A perfect matching on a bipartite graph was defined in Problem 3.4, but here we talk about matching on an arbitrary unoriented graph.)

There exist polynomial algorithms solving the weighted matching problem (see, for example, [**52**, Chapter 11], where an algorithm based on ideas of linear programming is described).

Appendix A

Elementary Number Theory

In this Appendix we outline some basic definitions and theorems of arithmetic. This, of course, is not a substitute for more detailed books; see e.g., [**70, 33**].

A.1. Modular arithmetic and rings. One says that a *is congruent to* b *modulo* q and writes

$$a \equiv b \pmod{q}$$

if $a - b$ is a multiple of q. This condition can be also expressed by the notation $q \mid (a - b)$, which reads "q divides $a - b$". A set of all (mod q)-congruent integers is called a congruence class. (For example, the set of even numbers and the set of odd numbers are congruence classes modulo 2.) Each class can be characterized by its canonical representative, i.e., an integer $r \in \{0, \dots, q-1\}$. We write

$$a \bmod q = r,$$

which means precisely that r is the *residue* of a (the remainder of integer division of a by q), i.e., $a = mq + r$, where $m \in \mathbb{Z}$, $r \in \{0, \dots, q-1\}$. In most cases we do not need to make a distinction between congruence classes and their canonical representatives, so the term "residue" refers to both. Thus $7 \bmod 3 = 1$, but we may also say that $(7 \bmod 3)$ is the congruence class containing 7 (i.e., the set $\{\dots, -5, -2, 1, 4, 7, \dots\}$) which has 1 as is canonical representative. In any case, the operation $x \mapsto (x \bmod q)$ takes an integer to a residue.

Residues can be added, subtracted or multiplied by performing the corresponding operation on integers they represent. Thus $r = r_1 r_2$ (modular multiplication) if and only if $a \equiv a_1 a_2 \pmod q$, where $a \bmod q = r$, $a_1 \bmod q = r_1$, and $a_2 \bmod q = r_2$. It is important to note that a_1 and a_2 can be replaced by any (mod q)-congruent numbers, the product being congruent too. Indeed,

$$\text{if } a_1 \equiv b_1 \text{ and } a_2 \equiv b_2, \quad \text{then } a_1 a_2 \equiv b_1 b_2 \pmod q.$$

What are the common properties of integer arithmetic operations and operations with residues? In the most abstract form, the answer is that both integers and (mod q) residues form commutative rings.

Definition A.1. A *ring* is a set R equipped with two binary operations, "+" and "$\cdot$" (the dot is usually suppressed in writing), and two special elements, 0 and 1, so that the following relations hold:

$$\begin{gathered}(a+b)+c = a+(b+c), \qquad a+b = b+a, \qquad a+0 = a,\\ (ab)c = a(bc), \qquad 1\cdot a = a\cdot 1 = a,\\ (a+b)c = ac+bc, \qquad c(a+b) = ca+cb.\end{gathered}$$

For any $a \in R$ there exists an element v such that $a + v = 0$.

If, in addition, $ab = ba$ for any a and b, then R is called a *commutative ring*.

In what follows we consider only commutative rings, so we omit the adjective "commutative".

Note that the element v in the above definition is unique. Indeed, if another element v' satisfies $a + v' = 0$, then

$$v' = v' + 0 = v' + (a+v) = (v'+a) + v = (a+v') + v = 0 + v = v + 0 = v.$$

Such v is denoted by $-a$. The relations on the list imply other well-known relations, for example, $a \cdot 0 = 0$. Indeed,

$$\begin{aligned}a\cdot 0 &= a\cdot 0 + a\cdot 0 + (-(a\cdot 0))\\ &= a(0+0) + (-(a\cdot 0)) = a\cdot 0 + (-(a\cdot 0)) = 0.\end{aligned}$$

The difference between two elements of a ring is defined as $a - b \stackrel{\text{def}}{=} a + (-b)$. Any ring becomes an Abelian group if we forget about the multiplication and 1, but keep + and 0. This group is called the *additive group* of the ring. The ring of residues modulo q is denoted by $\mathbb{Z}/q\mathbb{Z}$, whereas the corresponding additive group is denoted by $\mathbb{Z}_q$ (this is just the cyclic group of order q).

What are the differences between the ring of integers $\mathbb{Z}$ and residue rings $\mathbb{Z}/q\mathbb{Z}$? There are many; for example, $\mathbb{Z}$ is infinite while $\mathbb{Z}/q\mathbb{Z}$ is finite.

Another important distinction is as follows. For integers, $xy = 0$ implies that $x = 0$ or $y = 0$. This is not true for all residue rings (specifically, this is false in the case where q is a composite number). Example: $2 \cdot 5 \equiv 0 \pmod{10}$, although both 2 and 5 represent nonzero elements of $\mathbb{Z}/10\mathbb{Z}$. We say that an element x of a ring R is a *zero divisor* if $\exists y \neq 0\ (xy = 0)$. For example $0, 2, 3, 4, 6, 8, 9, 10$ are zero divisors in $\mathbb{Z}/12\mathbb{Z}$, whereas $1, 5, 7, 11$ are not. It will be shown below that r is a zero divisor in $\mathbb{Z}/q\mathbb{Z}$ if and only if r and q (regarded as integers) have a nontrivial common divisor.[1]

Let us introduce another important concept. An element $x \in R$ is called *invertible* if there exists y such that $xy = 1$; in this case we write $y = x^{-1}$. For example, $7 = 4^{-1}$ in $\mathbb{Z}/9\mathbb{Z}$, since $4 \cdot 7 \equiv 1 \pmod{9}$. It is obvious that if a and b are invertible, then ab is also invertible, and that $(ab)^{-1} = a^{-1}b^{-1}$. Therefore invertible elements form an Abelian group with respect to multiplication, which is denoted by R^*. For example, $\mathbb{Z}^* = \{1, -1\}$ and $(\mathbb{Z}/12\mathbb{Z})^* = \{1, 5, 7, 11\}$. In the latter case, invertible elements are exactly the elements which are not zero divisors. This is not a coincidence.

Proposition A.1. *If an element $x \in R$ is invertible, then x is not a zero divisor. In the case where R is finite, the converse is also true.*

Proof. Suppose that $xy = 0$. Then $y = (x^{-1}x)y = x^{-1}(xy) = x \cdot 0 = 0$.

Now let us assume that x is not a zero divisor, and R is finite. Then some elements in the sequence $0, x, x^2, x^3, \ldots$ must repeat, so there are some $n > m \geq 0$ such that $x^n = x^m$. Therefore $x^m(x^{n-m} - 1) = 0$. This implies that $x^{m-1}(x^{n-m} - 1) = 0$ because x is not a zero divisor. Iterating this argument, we get $x^{n-m} - 1 = 0$, i.e., $x^{n-m} = 1$. Hence $x^{-1} = x^{n-m-1}$. □

A.2. Greatest common divisor and unique factorization. One of the most fundamental properties of integers is as follows.

Theorem A.2. *Let a and b be integers, at least one of which is not 0. Then there exists $d \geq 1$ such that*

$$d \mid a, \qquad d \mid b, \qquad d = ma + nb, \qquad \text{where } m, n \in \mathbb{Z}.$$

Such d is called the *greatest common divisor* of a and b and denoted by $\gcd(a, b)$. Explanation of the name: firstly, d divides a and b by definition. Inasmuch as $d = ma + nb$, any common divisor of a and b also divides d; therefore it is not greater than d.

There are several ways to prove Theorem A.2. There is a constructive proof which actually provides an efficient algorithm for finding the numbers

[1] One can easily prove this assertion by factoring r and q into prime numbers. The argument, however, relies on the fact that the factorization is *unique*. The uniqueness of factorization is actually a theorem which requires a proof. We will derive this theorem from an equivalent of the above assertion.

d, m and n. This algorithm will be described in Section A.6. For now, we will use a shorter but more abstract argument. We note that the set $M = \{ma + nb : m, n \in \mathbb{Z}\}$ is a group with respect to addition, and that $a, b \in M$. Therefore Theorem A.2 can be obtained from the following more general result.

Theorem A.3. *Let $M \neq \{0\}$ be a subgroup in the additive group of integers. Then M is generated by a single element $d \geq 1$, i.e., $M = (d)$, where $(d) \stackrel{\text{def}}{=} \{kd : k \in \mathbb{Z}\}$.*

Proof. It is clear that M contains at least one positive element. Let d be the smallest positive element of M. Obviously, $(d) \subseteq M$, so it suffices to prove that any integer $x \in M$ is contained in (d).

Let $r = x \bmod d$, i.e., $x = kd + r$, where $0 \leq r < d$. Then $r = x - kd \in M$ (because $x, d \in M$, and M is a group). Since d is the smallest positive element in M, we conclude that $r = 0$. □

Now we derive a few corollaries from Theorem A.2.

Corollary A.2.1. *The residue $r = (a \bmod b) \in \mathbb{Z}/b\mathbb{Z}$ is invertible if and only if $\gcd(a, b) = 1$.*

Proof. The residue $r = (a \bmod b)$ being invertible means that $ma \equiv 1 \pmod b$ for some m, i.e., the equation $ma + nb = 1$ is satisfied for some integers m and n. But this is exactly the condition that $\gcd(a, b) = 1$. □

If p is a prime number, then every nonzero element of the ring $\mathbb{Z}/p\mathbb{Z}$ is invertible. A ring with this property is called a *field*.[2] The field $\mathbb{Z}/p\mathbb{Z}$ is also denoted by $\mathbb{F}_p$.

Corollary A.2.2. *If $\gcd(x, q) = 1$ and $\gcd(y, q) = 1$, then $\gcd(xy, q) = 1$.*

Proof. Using Corollary A.2.1, we reformulate the statement as follows: if $(x \bmod q)$ and $(y \bmod q)$ are invertible residues, then $(xy \bmod q)$ is also invertible. But this is obvious. □

Theorem A.4 ("Unique factorization"). *Any nonzero integer x can be represented in the form $x = \pm p_1 \cdots p_m$ where $p_1, \ldots, p_m$ are prime numbers. This representation is unique up to the order of factors.*

Proof. *Existence.* Without loss of generality we may assume that x is positive. If $x = 1$, then the factorization is trivial: the number of factors is zero. For $x > 1$ we use induction. If x is a prime, then we are done; otherwise $x = yz$ for some $y, z < x$, and y and z are already factored into primes.

[2] Fields play an important role in many parts of mathematics, e.g., in linear algebra: vectors with coefficients in an arbitrary field have essentially the same properties as real or complex vectors.

Uniqueness. This is less trivial. Again, we use induction: assume that $x > 1$ and that the uniqueness holds for all numbers from 1 through $x - 1$. Suppose that x has two factorizations:

$$x = p_1 \cdots p_m = q_1 \cdots q_n.$$

First, we show that $p_1 \in \{q_1, \ldots, q_n\}$. Indeed, if it were not the case, we would have $\gcd(p_1, q_j) = 1$ for all j. Hence $\gcd(p_1, x) = 1$ (due to Corollary A.2.2), which contradicts the fact that $p_1 \mid x$ (due to the first factorization).

By changing the order of q's, we can arrange that $p_1 = q_1$. Then we infer that there is a number $y < x$ with two factorizations, namely, $y = p_2 \cdots p_m = q_2 \cdots q_n$. By the induction assumption, the factorizations of y coincide (up to the order of factors), so that the same is true for $x = p_1 y$. □

It is often convenient to gather repeating prime factors, i.e., to write

$$x = \pm p_1^{\alpha_1} \cdots p_k^{\alpha_k},$$

where all p_j are distinct. Theorem A.4 implies many other "obvious" properties of integers, e.g., the following one.

Corollary A.4.1. *Let a and b be nonzero integers, and*

$$c = \frac{|ab|}{\gcd(a, b)}.$$

Then, for any integer x, the conditions $a \mid x$ and $b \mid x$ imply that $c \mid x$.

The number c is called the *least common multiple* of a and b.

A.3. Chinese remainder theorem. Let b and q be positive integers such that $b \mid q$. Then (mod q)-residues can be unambiguously converted to (mod b)-residues. Indeed, if $x' \equiv x''$ (mod q), then $x' \equiv x''$ (mod b). Therefore a map $\lambda_{q,b} : \mathbb{Z}/q\mathbb{Z} \to \mathbb{Z}/b\mathbb{Z}$ is defined, for example,

$$\lambda_{6,3} : \quad 0 \mapsto 0,\ \ 1 \mapsto 1,\ \ 2 \mapsto 2,\ \ 3 \mapsto 0,\ \ 4 \mapsto 1,\ \ 5 \mapsto 2.$$

This map is a *ring homomorphism*, i.e., it is consistent with the arithmetic operations (see Definition A.2 below).

Now, let $b_1 \mid q$ and $b_2 \mid q$. Any (mod q)-residue x can be converted into a (mod b_1)-residue x_1, as well as (mod b_2)-residue x_2. Thus a map $x \mapsto (x_1, x_2)$ is defined; we denote it by $\lambda_{q,(b_1,b_2)}$.

Theorem A.5 (Chinese remainder theorem). *Let $q = b_1 b_2$, where b_1 and b_2 are positive integers such that $\gcd(b_1, b_2) = 1$. Then the map*

$$\lambda_{q,(b_1,b_2)} : \mathbb{Z}/q\mathbb{Z} \to (\mathbb{Z}/b_1\mathbb{Z}) \times (\mathbb{Z}/b_2\mathbb{Z})$$

is an isomorphism of rings.

(The ring structure on the Cartesian product of rings will be defined below; see Definition A.3.)

Abstract terminology aside, Theorem A.5 says that the map $\lambda_{q,(b_1,b_2)}$ is one-to-one. In other words, for any a_1, a_2 the system

$$\begin{aligned} x &\equiv a_1 \pmod{b_1}, \\ x &\equiv a_2 \pmod{b_2} \end{aligned} \tag{A.1}$$

has a unique, up to (mod q)-congruence, solution. Indeed the existence of a solution says that $\lambda_{q,(b_1,b_2)}$ is a surjective (onto) map, whereas the uniqueness is equivalent to the condition that $\lambda_{q,(b_1,b_2)}$ is injective.

Proof. We will first prove that $\lambda_{q,(b_1,b_2)}$ is injective, i.e., any two solutions to system (A.1) are congruent modulo q. Let x' and x'' be such solutions; then $x = x' - x''$ satisfies

$$\begin{aligned} x &\equiv 0 \pmod{b_1}, \\ x &\equiv 0 \pmod{b_2}, \end{aligned}$$

i.e., $b_1 \mid x$ and $b_2 \mid x$. Therefore $c \mid x$, where c is the least common multiple of b_1 and b_2 (see Corollary A.4.1). But $\gcd(b_1, b_2) = 1$, hence $c = b_1 b_2 = q$.

Thus, $\lambda_{q,(b_1,b_2)}$ maps the set $\mathbb{Z}/q\mathbb{Z}$ to the set $(\mathbb{Z}/b_1\mathbb{Z}) \times (\mathbb{Z}/b_2\mathbb{Z})$ injectively. Both sets consist of the same number of elements, $q = b_1 b_2$; therefore $\lambda_{q,(b_1,b_2)}$ is a one-to-one map. □

We will now explain the abstract terms used in the formulation of Theorem A.5 and derive one corollary.

Definition A.2. Let A and B be rings. Denote the zero and the unit elements in these rings by $0_A, 0_B, 1_A, 1_B$, respectively. A map $f : A \to B$ is called a *ring homomorphism* if

$$f(x+y) = f(x) + f(y), \qquad f(xy) = f(x)\, f(y), \qquad f(1_A) = 1_B$$

for any $x, y \in A$. (Note that the property $f(0_A) = 0_B$ follows automatically.)

If the homomorphism f is a one-to-one map, it is called an *isomorphism*. (In this case the inverse map f^{-1} exists and is also an isomorphism.)

Definition A.3. The *direct product* of rings A_1 and A_2 is the set of pairs $A_1 \times A_2 = \{(x_1, x_2) :\ x_1 \in A_1,\, x_2 \in A_2\}$ endowed with componentwise ring operations:

$$\begin{gathered} (x_1, x_2) + (y_1, y_2) = (x_1 + y_1,\, x_2 + y_2), \\ (x_1, x_2) \cdot (y_1, y_2) = (x_1 y_1, x_2 y_2), \\ 0_{A_1 \times A_2} = (0_{A_1}, 0_{A_2}), \qquad 1_{A_1 \times A_2} = (1_{A_1}, 1_{A_2}). \end{gathered}$$

Similarly one can define the product of any number of rings.

Corollary A.5.1. *If $q = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ is the factorization of q, then there exists a ring isomorphism*

$$\mathbb{Z}/q\mathbb{Z} \;\cong\; \mathbb{Z}/p_j^{\alpha_j}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_j^{\alpha_j}\mathbb{Z}.$$

A.4. The structure of finite Abelian groups. We assume that the reader is familiar with the basic concepts of group theory (group homomorphism, cosets, quotient group, etc.) and can use them in simple cases, e.g., $\mathbb{Z}_4/\mathbb{Z}_2 \cong \mathbb{Z}_2$ but $\mathbb{Z}_4 \ncong \mathbb{Z}_2 \times \mathbb{Z}_2$. Also important is Lagrange's theorem, which says that the order of a subgroup divides the order of the group.

First, we consider a cyclic group $G = \{\zeta^k : k = 0, \dots, q-1\} \cong \mathbb{Z}_q$. Note that the choice of the *generator* $\zeta \in G$ is not unique: any element of the form ζ^k, where $\gcd(k, q) = 1$, also generates the group. Another simple observation: if $q = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ is the factorization of q, then

$$\mathbb{Z}_q \cong \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_k}, \qquad q_j = p_j^{\alpha_j}. \tag{A.2}$$

(We emphasize that here $\cong$ stands for an isomorphism of groups, not rings.) This property follows from Corollary A.5.1 — we just need to keep the additive structure of the rings and forget about the multiplication. We will call the group $\mathbb{Z}_{p^\alpha}$ (where p is prime) a *primitive cyclic group.*

Theorem A.6. *Let G be a finite Abelian group of order $q = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$. Then G can be decomposed into a direct product of primitive cyclic groups:*

$$G \;\cong\; \left(\prod_{r=1}^{\alpha_1} \left(\mathbb{Z}_{p_1^r}\right)^{m(p_1,r)} \right) \times \cdots \times \left(\prod_{r=1}^{\alpha_k} \left(\mathbb{Z}_{p_k^r}\right)^{m(p_k,r)} \right). \tag{A.3}$$

The numbers $m(p_j, r)$ are uniquely determined by the group G.

Note that the isomorphism in (A.3) may not be unique; there is no preferred way to choose one decomposition over another. However, the products in parentheses are defined canonically by the group G.

Corollary A.6.1. *If an Abelian group G of order q is not cyclic, then there is a nontrivial divisor $n \mid q$ such that $\forall x \in G\, (x^n = 1)$.*

Proof. Let $q = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ be the factorization of q. The group G is cyclic if and only if

$$m(p_j, r) = \begin{cases} 1 & \text{if } r = \alpha_j, \\ 0 & \text{otherwise.} \end{cases}$$

Since G is not cyclic, the above condition is violated for some j. However, $\sum_r r \cdot m(p_j, r) = \alpha_j$; therefore $m(p_j, \alpha_j) = 0$. If $n = q/p_j$, then $x^n = 1$ for all $x \in G$. □

The proof of Theorem A.6 requires some preparation. Since G is Abelian, the map $\varphi_a : x \mapsto x^a$ (where a is an arbitrary integer) is a homomorphism of G into itself. Let us define the following subgroups in G:

$$\begin{aligned} G^{(a)} &= \operatorname{Im}\varphi_a \ = \bigl\{x^a : \ x \in G\bigr\}, \\ G_{(a)} &= \operatorname{Ker}\varphi_a = \bigl\{x \in G : \ x^a = 1\bigr\}. \end{aligned} \tag{A.4}$$

Lemma A.7. *If* $\gcd(a,b) = 1$*, then* $G_{(ab)} = G_{(a)} \times G_{(b)}$*. In other words, for any* $z \in G_{(ab)}$ *there are unique* $x \in G_{(a)}$ *and* $y \in G_{(b)}$ *such that* $xy = z$*.*

Proof. Let $ma + nb = 1$. Then we can choose $x = z^{nb}$, $y = z^{ma}$, which proves the existence of x and y.

On the other hand, $G_{(a)} \cap G_{(b)} = \{1\}$. Indeed, if $u \in G_{(a)} \cap G_{(b)}$, then $u = u^{ma+nb} = u^{ma}u^{nb} = 1 \cdot 1 = 1$. This implies the uniqueness. □

Lemma A.8. *If* p *is a prime factor of* $|G|$*, then* G *contains an element of order* p*.*

Proof. We will use induction on $q = |G|$. Suppose that $q > 1$ and that the lemma holds for all Abelian groups of order $q' < q$. It suffices to show that G has an element whose order is a multiple of p.

Let x be a nontrivial element of G. If p divides the order of x, then we are done. Otherwise, let H be the subgroup generated by x, and $G' = G/H$ the corresponding quotient group. In this case p divides $|G'|$.

By the induction assumption, there is a nontrivial element $y' \in G'$ of order p. It is clear that $(y')^k = 1$ if and only if $p \mid k$. Let $y \in G$ be a member of the coset y' (recall that the quotient group is formed by cosets). Then $y^k \in H$ if and only if $p \mid k$. Therefore the order of y is a multiple of p. □

Proof of Theorem A.6. Lemma A.7 already shows that

$$G = \prod_{j=1}^{k} G_{(q_j)}, \qquad \text{where } \ q_j = p_j^{\alpha_j}.$$

By Lemma A.8, $|G_{(q_j)}|$ has no prime factors other than p_j; therefore $|G_{(q_j)}| = q_j = p_j^{\alpha_j}$. We need to split the subgroups $G_{(q_j)}$ even further.

Let us fix j and drop it from notation. The subgroup $G_{(p)}$ is special in that all its elements have order p (or 1). Therefore we may regard it as a linear space over the field $\mathbb{F}_p$ (the group multiplication plays the role of vector addition). Let

$$L_{p,r} = G_{(p)} \cap G^{(p^r)}, \qquad m(p,r) = \dim L_{p,r-1} - \dim L_{p,r}.$$

It is clear that $G_{(p)} = L_{p,0} \supseteq L_{p,1} \supseteq \cdots \supseteq L_{p,\alpha-1} \supseteq L_{p,\alpha} = \{1\}$.

We choose a basis in the space $L_{p,0}$ in such a way that the first $m(p,\alpha)$ basis vectors belong to $L_{p,\alpha-1}$, the next $m(p,\alpha-1)$ vectors belong to $L_{p,\alpha-2}$,

and so on. For each basis vector $e \in L_{p,r-1} \setminus L_{p,r}$ (where $\setminus$ denotes the set difference, i.e., $e \in L_{p,r-1}$ but $e \notin L_{p,r}$) we find an element $v \in G_q$ such that $v^{p^{r-1}} = e$. Powers of v form a cyclic subgroup of order p^r. One can show that G_q is the direct product of these subgroups (the details are left to the reader). □

A.5. The structure of the group $(\mathbb{Z}/q\mathbb{Z})^*$. Let $q = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ be the factorization of q. Due to Corollary A.5.1, there is a group isomorphism

$$(\mathbb{Z}/q\mathbb{Z})^* \cong \left(\mathbb{Z}/p_1^{\alpha_1}\mathbb{Z}\right)^* \times \cdots \times \left(\mathbb{Z}/p_k^{\alpha_k}\mathbb{Z}\right)^*.$$

Therefore it is sufficient to study the group $(\mathbb{Z}/p_k^{\alpha}\mathbb{Z})^*$, where p is a prime number. We begin with the case $\alpha = 1$.

Let p be a prime number. All nonzero (mod p)-residues are invertible, hence $\left|(\mathbb{Z}/p\mathbb{Z})^*\right| = p - 1$. The order of a group element always divides the order of the group (by Lagrange's theorem); therefore the order of any element in $(\mathbb{Z}/p\mathbb{Z})^*$ divides $p-1$. Thus we have obtained the following result.

Theorem A.9 (Fermat's little theorem). *If p is a prime number and $x \not\equiv 0 \pmod{p}$, then $x^{p-1} \equiv 1 \pmod{p}$.*

The next theorem fully characterizes the group $(\mathbb{Z}/p\mathbb{Z})^*$.

Theorem A.10. *If p is a prime number, then $(\mathbb{Z}/p\mathbb{Z})^*$ is a cyclic group of order $p - 1$.*

Proof. Suppose the Abelian group $G = (\mathbb{Z}/p\mathbb{Z})^*$ is not cyclic. By Corollary A.6.1, there exists some integer n, $0 < n < p - 1$, such that $x^n = 1$ for all $x \in G$. Therefore the equation $x^n - 1 = 0$ has $p - 1$ solutions in the field $\mathbb{F}_p$ (the solutions are the nonzero elements of the field).

The function $f(x) = x^n - 1$ is a polynomial of degree n with $\mathbb{F}_p$ coefficients. A polynomial of degree n with coefficients in an arbitrary field has at most n roots in that field. Indeed, if a_1 is a root, then $f(x) = (x - a_1)f_1(x)$, where f_1 is a polynomial of degree $n - 1$. If a_2 is another root, then $f_1(a_2) = (a_2 - a_1)^{-1} f(a_2) = 0$; therefore $f_1(x) = (x - a_2)f_2(x_2)$. This process can continue for at most n steps because each polynomial f_k has degree $n - k$.

But $f(x)$ has $p - 1 > n$ roots. We have arrived at a contradiction. □

Example. The group $(\mathbb{Z}/13\mathbb{Z})^*$ is generated by the element 2; see the table:

$k \in \mathbb{Z}_{12}$	0	1	2	3	4	5	6	7	8	9	10	11
$(2^k \bmod 13) \in (\mathbb{Z}/13\mathbb{Z})^*$	1	**2**	4	8	3	**6**	12	**11**	9	5	10	**7**

Note that 6, 11 and 7 are also generators of $(\mathbb{Z}/13\mathbb{Z})^*$. Indeed, if ζ is a generator of $(\mathbb{Z}/p\mathbb{Z})^*$, then the function $k \mapsto \zeta^k$ maps $\mathbb{Z}_{p-1}$ to $(\mathbb{Z}/p\mathbb{Z})^*$ isomorphically. Thus the element ζ^k generates the group $(\mathbb{Z}/p\mathbb{Z})^*$ if and

only if k generates $\mathbb{Z}_{p-1}$, i.e., if $\gcd(k,\, p-1) = 1$. In our case, the numbers $2, 6, 11, 7$ (the generators of $(\mathbb{Z}/13\mathbb{Z})^*$) correspond to the invertible (mod 12)-residues $k = 1, 5, 7, 11$.

Theorem A.11. *Let p be a prime number, and $\alpha \geq 1$ an integer.*

1. *If $p \neq 2$, then $(\mathbb{Z}/p^\alpha\mathbb{Z})^* \cong \mathbb{Z}_{p-1} \times \mathbb{Z}_{p^{\alpha-1}} \cong \mathbb{Z}_{(p-1)p^{\alpha-1}}$.*
2. *$(\mathbb{Z}/2^\alpha\mathbb{Z})^* \cong \mathbb{Z}_2 \times \mathbb{Z}_{2^{\alpha-2}}$ for $\alpha \geq 2$.*

(The isomorphism $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p^{\alpha-1}} \cong \mathbb{Z}_{(p-1)p^{\alpha-1}}$ is due to formula (A.2).)

Proof. An element $x \in \mathbb{Z}/p^\alpha\mathbb{Z}$ is invertible if and only if $(x \bmod p) \neq 0$. Therefore $\left|(\mathbb{Z}/p^\alpha\mathbb{Z})^*\right| = (p-1)p^{\alpha-1}$.

Let us denote $G = (\mathbb{Z}/p^\alpha\mathbb{Z})^*$ and introduce a sequence of subgroups $G \supseteq H_0 \supset H_1 \supset \cdots \supset H_{\alpha-1} = \{1\}$ defined as follows:

$$H_r = \left\{1 + p^{r+1}x :\ x \in \mathbb{Z}/p^{\alpha-r-1}\mathbb{Z}\right\} \quad \text{for } r = 0, \ldots, \alpha-1. \tag{A.5}$$

Note that if $a \in H_{r-1}$, then $a^p \in H_r$ (for $1 \leq r \leq \alpha-1$). Indeed,

$$(1 + p^r x)^p = 1 + \binom{p}{1}p^r x + \binom{p}{2}p^{2r}x^2 + \cdots \in H_r \quad (x \in \mathbb{Z}/p^{\alpha-r}\mathbb{Z}). \tag{A.6}$$

In the rest of the proof we consider the two cases separately.

1. $p \neq 2$. In this case $G = G_{(p-1)} \times G_{(p^{\alpha-1})}$ (the notation was defined in Section A.4; see (A.4)). The subgroup $G_{(p^{\alpha-1})}$ is easily identified: it coincides with H_0 defined by (A.5). Indeed, if $a \in H_0$, then $a^{p^{\alpha-1}} = 1$; therefore $H_0 \subseteq G_{(p^{\alpha-1})}$. On the other hand, $|G_{(p^{\alpha-1})}| = |H_0| = p^{\alpha-1}$.

We will not find the subgroup $G_{(p-1)}$ explicitly, but rather give an abstract argument:

$$G_{(p-1)} \cong G/G_{(p^{\alpha-1})} = G/H_0 \cong (\mathbb{Z}/p\mathbb{Z})^* \cong \mathbb{Z}_{p-1}.$$

It remains to check that $G_{(p^{\alpha-1})} = H_0$ is isomorphic to $\mathbb{Z}_{p^{\alpha-1}}$. To this end, it suffices to prove that any element of the set difference $H_0 \setminus H_1$ has order $p^{\alpha-1}$. If we apply the map $\varphi_p : x \mapsto x^p$ repeatedly, H_0 is mapped to H_1, then to H_2, and so on. We need to show that this shift over the subgroups takes place one step at a time, i.e., if $a \in H_{r-1} \setminus H_r$, then $a^p \in H_r \setminus H_{r+1}$.

The condition $a \in H_{r-1} \setminus H_r$ can be represented as follows: $a = 1 + p^r x$, where $x \in \mathbb{Z}/p^{\alpha-r}\mathbb{Z}$ and $x \not\equiv 0 \pmod p$. We use Equation (A.6) again. Note that the terms denoted by the ellipsis contain p raised to the power $3r \geq r+2$ or higher, so they are not important. Moreover, $\binom{p}{2} = \frac{p(p-1)}{2}$ is divisible by p. Therefore,

$$a^p = (1 + p^r x)^p \equiv 1 + p^{r+1}x + p^{r+1}\binom{p}{2}p^{r-1}x^2 \equiv 1 + p^{r+1}x \pmod{p^{r+2}},$$

so that $a^p \in H^r$ but $a^p \notin H^{r+1}$.

2. $p = 2$. The only reason why the previous proof does not work is that $\binom{2}{2} = 1$ is not divisible by 2. But the last argument is still correct for $r > 1$; therefore $H_1 \cong \mathbb{Z}_{p^{\alpha-2}}$. On the other hand, $G = \{1, -1\} \times H_1$. □

A.6. Euclid's algorithm. Once again, let a and b be integers, at least one of which is not 0. How does one compute $\gcd(a, b)$ and solve the equation $ma + nb = \gcd(a, b)$ efficiently?

Euclid's algorithm. Without loss of generality, we may assume that $b > 0$. We set $x_0 = a,\ y_0 = b$ and iterate the transformation

$$(x_{j+1}, y_{j+1}) = (y_j,\ x_j \bmod y_j) \tag{A.7}$$

until we get a pair of the form $(x_t, y_t) = (d, 0)$. This d is equal to $\gcd(a, b)$. Indeed, any common divisor of x and y is a common divisor of y and $x \bmod y$, and vice versa. Therefore $\gcd(a, b) = \gcd(d, 0) = d$.

The complexity of the algorithm will be estimated later, after we describe additional steps that are needed to find m and n satisfying $ma + nb = d$.

We first give some analysis of the algorithm. Procedure (A.7) can be represented as follows:

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{pmatrix} k_j & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_{j+1} \\ y_{j+1} \end{pmatrix}, \qquad \begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}, \tag{A.8}$$

where $k_j = \lfloor x_j / y_j \rfloor$. Note that k_0 is an arbitrary integer, while $k_1, \ldots, k_{t-1}$ are positive; moreover, $k_{t-1} > 1$ if $t > 1$. Thus we have

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} k_0 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} k_{t-1} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} d \\ 0 \end{pmatrix}. \tag{A.9}$$

The product of the matrices here is denoted by A_t. Let us also introduce partial products,

$$A_j = \begin{pmatrix} k_0 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} k_{j-1} & 1 \\ 1 & 0 \end{pmatrix}. \tag{A.10}$$

It is easy to see that $\det(A_j) = (-1)^j$, and that

$$A_j = \begin{pmatrix} p_j & p_{j-1} \\ q_j & q_{j-1} \end{pmatrix}, \qquad \begin{aligned} &p_0 = 1, \quad p_{-1} = 0, \quad p_{j+1} = k_j p_j + p_{j-1}, \\ &q_0 = 0, \quad q_{-1} = 1, \quad q_{j+1} = k_j q_j + q_{j-1}. \end{aligned} \tag{A.11}$$

Equation (A.9) says that $a = p_t d$ and $b = q_t d$. (Therefore p_t/q_t is an irreducible fraction representation of the rational number a/b.) On the other hand, $p_t q_{t-1} - p_{t-1} q_t = \det(A_t) = (-1)^t$, hence

$$(-1)^t (q_{t-1} a - p_{t-1} b) = d.$$

Thus we have solved the equation $ma + nb = d$. We summarize the result as follows.

Extended Euclid's algorithm (for solving the equation $ma + nb = d$). We iterate transformation (A.7), computing the ratios $k_j = \lfloor x_j/y_j \rfloor$ on the way. Then we compute p_j, q_j according to (A.11) (this can be made a part of the iterative procedure as well). The answer to the problem is as follows:

$$m = (-1)^t q_{t-1}, \quad n = (-1)^{t-1} p_{t-1}. \tag{A.12}$$

Let us estimate the complexity of the algorithm in terms of the problem size s, i.e., the total number of digits in the binary representations of a and b. Using formula (A.8) and the conditions $k_1, \dots, k_{t-1} \geq 1,\ d \geq 1$, we obtain the componentwise inequality

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \geq \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{t-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} F_t \\ F_{t-1} \end{pmatrix}, \quad \text{where } F_t = \frac{\tau^t - (-\tau)^{-t}}{\sqrt{5}}.$$

(Here $F_0 = 0,\ F_1 = 1,\ F_{j+1} = F_j + F_{j-1}$ are the Fibonacci numbers, whereas $\tau = \frac{1+\sqrt{5}}{2}$ is the golden ratio.) Therefore $b = x_1 \geq \Omega(\tau^t)$, and $t \leq O(\log b) = O(s)$. Each application of transformation (A.7), as well as the computation of k_j, p_j, q_j, are performed by $O(s^2)$-size circuits. Therefore the overall complexity is $O(s^3)$.

A.7. Continued fractions. Euclid's algorithm can be viewed as a procedure for converting the fraction $z = a/b$ into the irreducible fraction p_t/q_t. It turns out that some steps in this procedure (namely, the computation of $k_0, \dots, k_{t-1}$) can be formulated in terms of rational numbers, or even real numbers. Indeed, let us define $z_j = x_j/y_j$. Then equations (A.7) and (A.8) become

$$z_{j+1} = \frac{1}{\operatorname{frac}(z_j)}, \qquad k_j = \lfloor z_j \rfloor, \qquad \text{where } \operatorname{frac}(x) \stackrel{\text{def}}{=} x - \lfloor x \rfloor, \tag{A.13}$$

$$z = z_0, \qquad z_j = k_j + \frac{1}{z_{j+1}}, \qquad z_t = \infty, \tag{A.14}$$

(Note that $z_j > 1$ for all $j \geq 1$.) Thus we obtain a representation of z in the form of a finite *continued fraction* $[k_0; k_1, \dots, k_{t-1}]$ with *terms* k_j, which is defined as follows:

$$[k_0; k_1, \dots, k_{t-1}] \stackrel{\text{def}}{=} k_0 + \cfrac{1}{k_1 + \cfrac{1}{\cdots\cdots + \cfrac{1}{k_{t-1} + \frac{1}{\infty}}}}, \qquad \begin{array}{l} k_j \in \mathbb{Z}, \\ k_1, \dots, k_{t-1} \geq 1. \end{array} \tag{A.15}$$

We call a continued fraction *canonical* if $t = 1$ or $k_{t-1} > 1$; the procedure described by equation (A.13) guarantees this property. (If we started with an irrational number z, we would get an infinite continued fraction, which is always considered canonical.)

Proposition A.12. 1. *Any real number has exactly one canonical continued fraction representation.*

2. *A rational number with canonical representation* $[k_0; k_1, \dots, k_{t-1}]$ *has exactly one noncanonical representation, namely,* $[k_0; k_1, \dots, k_{t-1} - 1, 1]$.

(The proof is left as an exercise to the reader.)

What are continued fractions good for? We will see that the first j terms of the canonical continued fraction for z provide a good approximation of z by a rational number p_j/q_j, meaning that $|z - p_j/q_j| = O(q_j^{-2})$. All sufficiently good approximations are obtained by this procedure (see Theorem A.13 below). Put it in a different way: if z is a sufficiently good approximation of a rational number p/q, we can find that number by examining the continued fraction representation of z.

To deal with partial continued fraction expansions, we define the function

$$[k_0; k_1, \dots, k_{j-1}](u) \stackrel{\text{def}}{=} k_0 + \cfrac{1}{k_1 + \cfrac{1}{\cdots\cdots\cdots \cfrac{}{k_{j-1} + \frac{1}{u}}}}. \tag{A.16}$$

It allows us to represent z as follows: $z = [k_0; k_1, \dots, k_{j-1}](z_j)$.

To obtain an explicit formula for function (A.16), we note that it is a composition of fractional linear functions,

$$[k_0; k_1, \dots, k_{j-1}](u) = g_{k_0}(g_{k_1}(\cdots g_{k_{j-1}}(u)\cdots)), \quad \text{where } g_k(u) = \frac{ku+1}{u}.$$

Composing fractional linear functions is equivalent to multiplying 2×2 matrices: if $f_1(u) = (a_1u + b_1)/(c_1u + d_1)$ and $f_2(v) = (a_2v + b_2)/(c_2v + d_2)$, then $f_1(f_2(v)) = (av + b)/(cv + d)$, where

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} a_1 & b_1 \\ c_1 & d_1 \end{pmatrix} \begin{pmatrix} a_2 & b_2 \\ c_2 & d_2 \end{pmatrix}.$$

Therefore

$$[k_0; k_1, \dots, k_{j-1}](u) = \frac{p_j u + p_{j-1}}{q_j u + q_{j-1}}, \tag{A.17}$$

where the integers p_j, q_j are defined by equation (A.11).

Substituting $u = \infty$ into (A.17), we obtain the rational number

$$\frac{p_j}{q_j} = [k_0; k_1, \dots, k_{j-1}]. \tag{A.18}$$

This number is called the *j-th convergent* of z. For example, $p_t/q_t = z$, $p_1/q_1 = k_0 = \lfloor z \rfloor$; we may also define the 0-th convergent, $p_0/q_0 = 1/0 = \infty$. Note that the continued fraction in (A.18) is not necessarily canonical.

Let us examine the properties of convergents. Inasmuch as $p_jq_{j-1} - p_{j-1}q_j = \det(A_j) = (-1)^j$ and $q_0 < q_1 < q_2 < \cdots$, the following relations hold:

$$\text{(A.19)}\quad \frac{p_j}{q_j} - \frac{p_{j-1}}{q_{j-1}} = \frac{(-1)^j}{q_jq_{j-1}}, \qquad \frac{p_1}{q_1} < \frac{p_3}{q_3} < \cdots \le z \le \cdots < \frac{p_2}{q_2} < \frac{p_0}{q_0}.$$

(To put z in the middle, we have used the fact that $z = [k_0; k_1, \ldots, k_{j-1}](z_j)$ for $1 < z_j \le \infty$.) This justifies the name "convergent".

Theorem A.13. *Let z be a real number, p/q an irreducible fraction, $q > 1$.*

1. *If p/q is a convergent of z, then $|z - p/q| < 1/(q(q+1))$.*
2. *If $|z - p/q| < 1/(q(2q-1))$, then p/q is a convergent of z.*

Proof. Let us consider a more general problem: given the number $w = p/q$, find the set of real numbers z that have w among their convergents. We can represent w as a canonical continued fraction $[k_0; k_1, \ldots, k_{t-1}]$ and define p_j, q_j $(j = 0, \ldots, t)$ using this fraction. Note that $t > 1$ (because w is not an integer), and that $p_t = p$, $q_t = q$. It is easy to see that three cases are possible.

1. $z = w = p_t/q_t$.
2. The canonical continued fraction for z has the form $[k_0; k_1, \ldots, k_{t-1}, \ldots]$. Then $z = (p_tz_t + p_{t-1})/(q_tz_t + q_{t-1})$ for $1 < z_t < \infty$; therefore z lies between p_t/q_t and $(p_t + p_{t-1})/(q_t + q_{t-1})$ (the ends of the interval are not included).
3. The canonical continued fraction for z is $[k_0; k_1, \ldots, k_{t-1}-1, 1, \ldots]$. In this case $z = (p_tz_t + p_{t-1})/(q_tz_t + q_{t-1})$, where

$$k_t + \frac{1}{z_t} \;=\; k_t - 1 + \frac{1}{1 + \dfrac{1}{z_{t+1}}}, \qquad 1 < z_{t+1} < \infty.$$

   Thus $z_t < -2$, so that z lies between p_t/q_t and $(2p_t - p_{t-1})/(2q_t - q_{t-1})$.

Combining these cases, we conclude that w is a convergent of z if and only if $z \in I$, where I is the open interval with these endpoints:

$$\text{(A.20)}\quad \begin{aligned} \frac{p_t + p_{t-1}}{q_t + q_{t-1}} &= \frac{p_t}{q_t} - (-1)^t \frac{1}{q_t(q_t + q_{t-1})}, \\ \frac{2p_t - p_{t-1}}{2q_t - q_{t-1}} &= \frac{p_t}{q_t} + (-1)^t \frac{1}{q_t(2q_t - q_{t-1})}. \end{aligned}$$

But $1 \le q_{t-1} \le q_t - 1$; therefore

$$S\Big(p_t/q_t\ ,\ 1/(q_t(2q_t-1))\Big) \;\subseteq\; I \;\subseteq\; S\Big(p_t/q_t\ ,\ 1/(q_t(q_t+1))\Big),$$

where $S(x, \delta)$ stands for the δ-neighborhood of x. □

Bibliography

[1] J. F. Adams, *Lectures on Lie groups*, W. A. Benjamin, Inc., New York–Amsterdam, 1969.

[2] L. M. Adleman, J. DeMarrais and M. A. Huang, *Quantum computability*, SIAM J. Comput. **26** (1997), 1524–1540.

[3] D. Aharonov and M. Ben-Or, *Fault tolerant quantum computation with constant error*, e-print `quant-ph/9611025`; extended version, e-print `quant-ph/9906129`.

[4] D. Aharonov, A. Kitaev, and N. Nisan, *Quantum circuits with mixed states*, STOC'29, 1997; e-print `quant-ph/9806029`.

[5] A. V. Aho and J. D. Ullman, *Principles of compiler design*, Addison-Wesley, Reading, MA, 1977.

[6] L. Babai and S. Moran, *Arthur–Merlin games: A randomized proof system and a hierarchy of complexity classes*, J. Comput. System Sci. **36** (1988), 254–276.

[7] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, *Elementary gates for quantum computation*, Phys. Rev. Ser. **A52** (1995), 3457–3467; e-print `quant-ph/9503016`.

[8] D. A. Barrington, *Bounded-width polynomial-size branching programs recognize exactly those languages in* NC^1, J. Comput. System Sci. **38** (1989), 150–164.

[9] P. Beame, S. Cook, and H. J. Hoover, *Log depth curcuits for division and related problems*, SIAM J. Comput. **15** (1986), 994–1003.

[10] C. H. Bennett, *Logical reversibility of computations*, IBM J. Res. Develop. **17** (1973), 525–532.

[11] C. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. Wootters, *Teleporting an unknown quantum state via dual classical and Einstein–Podolsky–Rosen channel*, Phys. Rev. Lett. **70** (1993), 1895–1899.

[12] C. Bennett, D. DiVincenzo, J. Smolin, and W. Wootters, *Mixed state entanglement and quantum error correction*, Phys. Rev. **A54** (1996), 3824–3851; e-print `quant-ph/9604024`.

[13] D. Boneh and R. Lipton, *Quantum cryptoanalysis of hidden linear functions*, Proc. of Advances in Cryptology—CRYPTO-95, Lecture Notes Computer Science, vol. 963, Springer-Verlag, Berlin, 1995, pp. 424–437.

[14] R. Boppana and M. Sipser, *The complexity of finite functions*, Handbook of Theoretical Computer Science. Volume A, Algorithms and Complexity, Ch. 14. J. van Leeuwen (ed.), Elsevier, Amsterdam; MIT Press, Cambridge, MA, 1990, pp. 757–804.

[15] N. Bourbaki, *Lie Groups and Lie Algebras*, Hermann, Paris, 1971.

[16] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane *Quantum error correction and orthogonal Geometry*, Phys. Rev. Lett. **78** (1997), 405–408; e-print `quant-ph/9605005`.

[17] A. R. Calderbank and P. W. Shor, *Good quantum error-correcting codes exist*, Phys. Rev. **A54** (1996), 1098–1106; e-print `quant-ph/9512032`.

[18] R. Cleve and J. Watrous, *Fast parallel circuits for the quantum Fourier transform*, FOCS'41, 2000, pp. 526–536; e-print `quant-ph/0006004`.

[19] D. Coppersmith, *An approximate Fourier transform useful in quantum factoring*, Technical Report RC19642, IBM, 1994; e-print `quant-ph/0201067`.

[20] D. Deutsch, *Quantum theory, the Church–Turing principle and the universal quantum computer*, Proc. Roy. Soc. London **A400** (1985), 97–117.

[21] ———, *Quantum computational networks*, Proc. Roy. Soc. London. **A425** (1989), 73–90.

[22] P. Erdös and J. Spencer, *Probabilistic methods in combinatorics*, Academic Press, New York, 1974.

[23] R. P. Feynman, *Simulating physics with computers*, Internat. J. Theoret. Phys. **21**(6/7) (1982), 467–488.

[24] ———, *Quantum mechanical computers*, Optics News, **11**, February 1985, p. 11.

[25] L. Fortnow and M. Sipser, *Are there interactive protocols for Co-NP-languages?*, Inform. Process. Lett. **28** (1988), 249–251.

[26] M. H. Freedman, *P/NP, and the quantum field computer*, Proc. Nat. Acad. Sci. U.S.A. **95** (1998), 98–101.

[27] M. H. Freedman and A. Yu. Kitaev, *Diameter of homogeneous spaces*, unpublished.

[28] M. R. Garey and D. S. Johnson, *Computers and intractability*, Freeman, New York, 1983.

[29] L. Grover, *A fast quantum mechanical algorithm for database search*, STOC'28, 1996, pp. 212–219.

[30] J. Gruska, *Quantum Computing*, McGraw-Hill, London, 1999.

[31] A. W. Harrow, B. Recht and I. L. Chuang, *Tight bounds on discrete approximation of quantum gates*, e-print `quant-ph/0111031`.

[32] R. Impagliazzo and A. Wigderson. P = BPP *if* E *requires exponential circuits: Derandomizing the* XOR *lemma*, STOC'29, 1997.

[33] A. J. Khinchin, *Continued fractions*, Univ. of Chicago Press, 1992.

[34] A. A. Kirillov, *Elements of the theory of representations*, Springer-Verlag, New York, 1976.

[35] A. Yu. Kitaev, *Fault-tolerant quantum computation by anyons*, e-print `quant-ph/9707021`.

[36] A. Yu. Kitaev, *Quantum computations: algorithms and error correction*, Uspehi Mat. Nauk **52** (1997), no. 6, 53–112; English transl., Russian Math. Surveys **52** (1997), no. 6, 1191–1249.

[37] A. Kitaev, A. Shen, M. Vyalyi, *Classical and Quantum Computations*, Moscow, 1999 (in Russian); available at `http://www.mccme.ru/free-books`.

[38] A. Yu. Kitaev and J. Watrous, *Parallelization, amplification, and exponential time simulation of quantum interactive systems*, STOC'32, 2000, pp. 608–617.

[39] S. C. Kleene, *Mathematical logic*, Wiley, New York, 1967.

[40] ———, *Introduction to metamathematics*, Van Nostrand, New York, 1952.

[41] E. Knill and R. Laflamme, *A theory of quantum error-correcting codes*, e-print `quant-ph/9604034`.

[42] E. Knill, R. Laflamme, and W. Zurek, *Threshold accuracy for quantum computation*, e-print `quant-ph/9610011`.

[43] D. E. Knuth, *The art of computer programming*, Addison-Wesley, Reading, MA, 1973.

[44] A. I. Kostrikin and Yu. I. Manin, *Linear algebra and geometry*, Nauka, Moscow, 1986; English transl., Gordon and Breach, New York, 1989.

[45] R. Landauer, *Irreversibility and heat generation in the computing process*, IBM J. Res. Develop. **3** (1961), 183–191.

[46] C. Lautemann, *BPP and the polynomial hierarchy*, Inform. Process. Lett. **17** (1983), 215–217.

[47] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correction codes*, North Holland, New York, 1981.

[48] A. I. Maltsev, *Algorithms and recursive functions*, Wolters-Noordhof, Groningen, 1970.

[49] Yu. I. Manin *Computable and Incomputable*, Moscow, 1980 (in Russian).

[50] M. Marcus and H. Minc. *A survey of matrix theory and matrix inequalities*, Allyn and Bacon, Boston, 1964.

[51] M. A. Nielsen and I. L. Chuang *Quantum computation and quantum information*, Cambridge University Press, 2000.

[52] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[53] V. V. Prasolov, *Problems and theorems in linear algebra*, Amer. Math. Soc., Providence, RI, 1994.

[54] H. Rogers, *Theory of recursive functions and effective computability*, MIT Press, Cambridge, MA, 1987.

[55] A. Schrijver, *Theory of linear and integer programming*, Wiley-Interscience, Chichester, NY, 1986.

[56] J. P. Serr, *Lie algebras and Lie groups*, W. A. Benjamin, Inc., New York–Amsterdam, 1965.

[57] I. R. Shafarevich, *Basic notions of algebra*, Springer-Verlag, New York, 1997.

[58] A. Shamir, *IP=PSPACE*, J. ACM **39** (1992), 869–877.

[59] A. Shen, *IP=PSPACE: simplified proof*, J. ACM **39** (1992), 878–880.

[60] J. R. Shoenfield, *Mathematical logic*, Addison-Wesley, Reading, MA, 1967.

[61] ———, *Degrees of unsolvability*, Elsevier, New York, 1972.

[62] P. W. Shor, *Algorithms for quantum computation: Discrete log and factoring*, FOCS'35, 1994, pp. 124–134.

[63] ———, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput. **26** (1997), 1484–1509; e-print `quant-ph/9508027`.

[64] ———, *Scheme for reducing decoherence in quantum memory*, Phys. Rev. **A52** (1995), 2493–2496.

[65] ———, *Fault-tolerant quantum computation*, FOCS'37, 1996, pp. 56–65; e-print `quant-ph/9605011`.

[66] D. Simon, *On the power of quantum computation*, FOCS'35, 1994, pp. 116–123.

[67] M. Sipser, *Introduction to the theory of computation*, PWS, Boston, 1997.

[68] A. M. Steane, *Multiple particle interference and quantum error correction*, Proc. Roy. Soc. London **A452** (1996), p. 2551; e-print `quant-ph/9601029`.

[69] C. Umans, *Pseudo-random generators for all hardnesses*, to appear in STOC 2002 and Complexity 2002 joint session; `http://www.research.microsoft.com/~umans/research.htm`.

[70] I. M. Vinogradov, *Elements of number theory*, Dover, New York, 1954.

[71] J. Watrous, *On quantum and classical space-bounded processes with algebraic transition amplitudes*, FOCS'40, 1999, pp. 341–351; e-print `cs.CC/9911008`.

[72] J. Watrous, *PSPACE has constant-round quantum interactive proof systems*, FOCS'40, 1999, pp. 112–119; e-print: `cs.CC/9901015`.

[73] T. Yamakami and A. C. Yao, $\mathrm{NQP}_{\mathbb{C}} = \mathrm{co} - \mathrm{C}_{=}\mathbf{P}$, Inform. Process. Lett. **71** (2) (1999), 63–69; e-print `quant-ph/9812032`.

[74] A. C.-C. Yao, *Quantum circuit complexity*, FOCS'34, 1993, pp. 352–361.

[75] C. Zalka, *Grover's quantum searching algorithm is optimal*, e-print `quant-ph/9711070`.

Index

This book presents a concise introduction to an emerging and increasingly important topic, the theory of quantum computing. The development of quantum computing exploded in 1994 with the discovery of its use in factoring large numbers—an extremely difficult and time-consuming problem when using a conventional computer. In less than 300 pages, the authors set forth a solid foundation to the theory, including results that have not appeared elsewhere and improvements on existing works.

The book starts with the basics of classical theory of computation, including NP-complete problems and the idea of complexity of an algorithm. Then the authors introduce general principles of quantum computing and pass to the study of main quantum computation algorithms: Grover's algorithm, Shor's factoring algorithm, and the Abelian hidden subgroup problem. In concluding sections, several related topics are discussed (parallel quantum computation, a quantum analog of NP-completeness, and quantum error-correcting codes).

This is a suitable textbook for a graduate course in quantum computing. Prerequisites are very modest and include linear algebra, elements of group theory and probability, and the notion of an algorithm (on a formal or an intuitive level). The book is complete with problems, solutions, and an appendix summarizing the necessary results from number theory.

ISBN 978-0-8218-3229-5

9 780821 832295

GSM/47.S

For additional information
and updates on this book, visit

www.ams.org/bookpages/gsm-47

AMS *on the* Web
www.ams.org