Logic and **L**ogical **P**hilosophy
Volume 18 (2009), 313–332
DOI: 10.12775/LLP.2009.016

Krzysztof Wójtowicz

THEORY OF QUANTUM COMPUTATION AND PHILOSOPHY OF MATHEMATICS Part I

Abstract. The aim of this paper is to present some basic notions of the theory of quantum computing and to compare them with the basic notions of the classical theory of computation. I am convinced, that the results of quantum computation theory (QCT) are not only interesting in themselves, but also should be taken into account in discussions concerning the nature of mathematical knowledge.

The philosophical discussion will however be postponed to another paper. QCT seems not to be well-known among philosophers (at least not to the degree it deserves), so the aim of this paper is to provide the necessary technical preliminaries presented in a way accessible to the general philosophical audience.

Keywords: quantum algorithm, Shor's algorithm, quantum computation theory, Deutsch's algorithm, philosophy of mathematics

1. Classical computations

1.1. What is a computation?

The notion of computation is a very general and broad one. In the most general sense, any task, which requires some kind of problem-solving can be viewed as performing some computation. In this most general sense, we could speak of a computation when parking a car, playing tennis, looking for our keys *etc.* But such a notion of computation would

Received December 4, 2009
 ISSN: 1425-3305

be much too general and vague to allow for a fruitful analysis. Other examples (which are intuitively more computational in nature) are e.g. finding the shortest route from one place to another, computing the area of a land or calculating our tax. In every of these cases there is an input, a process of "computation" and an output. But the general questions "what does it mean to compute?" or "what are the common features of the computational processes?" do not have an obvious answer. It is intuitively quite clear, that we would not consider choosing the best holiday offer to be a computation—at least not in the strict sense. So what are the criteria for being a computational process?

We will not go into the philosophical discussion concerning the notion of computation, assuming in particular the Church-Turing thesis as a working hypothesis. What is more important for us is the observation, that every time we perform a computational task (i.e. we compute), we apply to some physical resources (i.e. to make use of an appropriate physical system). The evolution of that system corresponds to the desired output. A straightforward example of such a system is of course any PC.[1] But there may be also other devices, which help us in solving problems, and the question arises, which of them be called computational devices and whether the processes performed by these devices correspond to our intuitive notion of computation. A nice example of such a process is given by Aharonov in [Aharonov 1998]. The famous Spanish architect, Gaudi used a system of ropes while designing the *la sagrada familia* (holy family church) in Barcelona. The plan was very complicated, as there where many towers and arcs between them emerging in unexpected places. It was not possible to solve the system of equations which described the equilibrium of the system—but Gaudi simply took ropes corresponding to each of the arcs, tied them together, and tied the system to the ceiling, obtaining the mirror image of the projected church. The solution was "computed" by gravity! The question arises: was that really a computation—and was that system of ropes really a computing device . . . ?

The formal notion of computation should be sufficiently general to cover a substantial part of our computational tasks. Of course, the particular algorithms designed to solve particular computational problems differ, but it is reasonable to expect, that they should all be incorporated

[1] In many cases this physical system is our brain.

within one single model. This model should cover all possible computational task (and consequently—there should be one universal computing device, suitable to perform all these tasks).[2]

Since the seminal works of Turing there is such a general formal model of computation available—i.e. the Turing machine model.[3] The important feature of the Turing machine model is that there exists a universal Turing machine—i.e. a machine U, which can simulate the instruction table of any Turing machine M (on any given input). Intuitively, that means, that our universal machine U is presented with the input consisting of the instruction table of the machine M, and the input for M. U is able to look up in the instruction table for M what has to be done at any step, and simulates the behavior of M.[4] That means, that all computational tasks can be performed on the universal Turing machine,[5] so it fulfills our desire of having one general model of computation.

The Turing machine model has natural motivations, and according to the Church-Turing thesis, it is the adequate formal counterpart of the

[2]After all, it would be rather awkward, if we had to build different machines for different computing tasks (to put it very crude: one machine for performing arithmetical operations, another one for finding the shortest route on a map, and still another for checking whether a certain formula is a tautology of the propositional calculus).

[3]A Turing machine consists of an unbounded tape (divided into cells), a head which scans the symbol from the tape (and is also capable of writing new symbols onto the tape) and a transition function. At any moment of the computation, the Turing machine occupies one of a finite number of possible internal states. Two states of the machines are distinguished: the initial state and the terminal state (if the machine enters the terminal state, it means of course, that the computation terminates, and the tape contains the desired solution). At the beginning of the computation, an input (a finite string of symbols) is presented on the tape. The computation proceeds as follows: given the internal state, and the content of the cell on the tape (scanned by the head), the transition function determines: (1) the symbol that will be written on the tape; (2) the internal state the Turing machine goes into; (3) the displacement of the head on the tape (it can move left, right, or do nothing). The number of symbols which can occur on the tape is always fixed (and finite). But as we can always simulate the functioning of such a Turing machine on a machine which uses only two symbols (0 and 1), therefore—without loss of generality—we can consider only such machines (and in this paper we will assume, that symbols on the tape are just 0 and 1).

[4]Of course, this would not be very convenient—if we design algorithms, we always have in mind the particular problem in question. But such a simulation is possible (with an at most polynomial slowdown).

[5]Of course, programming such an universal Turing machine would be rather tedious, but it is—in principle—possible.

informal, intuitive notion of computability. The answer to the question "what does it mean to compute?" is therefore "it means to perform operations which can be performed (simulated) on a Turing machine". Of course, as it involves both a mathematical notion, and an informal notion, the Church-Turing thesis cannot be proved, but can only be argued for. One of the arguments in favor of the Church-Turing thesis rests on the fact, that various formal models of computation have been proposed (recursive functions, λ-calculus *etc.*), and all of them turned out to be equivalent to the Turing machine model.

In this paper we will restrict our attention to the classical model of computation—in particular I will not even touch upon the subject of hypercomputability. In last years, the notion of hypercomputability, supertasks *etc.* has attracted some attention of computer scientists and philosophers[6]. I short, a hypercomputer is a device, which can perform non-algorithmic computational tasks (i.e. it can yield solutions to problems, which cannot be solved by a Turing machine). The question whether there are non-algorithmic processes in nature (and whether such processes could be harnessed to solve non-algorithmic problems) remains open; some authors claim, that such processes do in fact exist, some consider them to be merely a theoretical possibility.[7] Even if there are no physical hypercomputers, one important conclusion has to be drawn from these considerations: computation cannot be viewed only as an abstract, theoretical notion, and our understanding of it should at least take into account the (theoretical and practical) physical limitations.

However, it should be stressed, that quantum computers are not hypercomputers—they cannot solve classically unsolvable problems, but in some cases they can be much quicker (and this would be their advantage, if quantum computers would in fact be built).

1.2. Decidable problems

The first distinction, that has to be made is between algorithmically solvable (computable, decidable) and unsolvable problems. A decidable problem is a problem, for which a Turing machine (intuitively—an algo-

[6]For supertasks cf. e.g. Laraudagoitia 1996], for recent discussion on hypercomputation e.g. [Davis 2006], [Stannett 2006].

[7]For discussion cf. e.g. [Pitovsky 1990], [Hogarth 1994].

rithm) can be defined which solves the problem: given the input (which represents the problem in a formally suitable way), the Turing machine yields the solution of this problem. Some simple examples of decidable problems are: (a) checking whether $n = k + l$; (b) checking whether n is a prime number; (c) checking whether two graphs are isomorphic; (d) finding the shortest route on a map between two towns A and B; (e) checking whether a formula of the propositional calculus is satisfiable; *etc.* There are lots of number-theoretic and combinatorial (e.g. graph-theoretic) problems which are decidable. But of course there are also non-decidable problems (for which no *general* algorithms exists); the most famous of them is the halting problem.[8]

In this paper we are interested in decidable problems only. The general notion of decidability abstracts form practical limitations (i.e. limitations of time, space and other physical resources). But it is quite clear, that we have an intuitive notion of the computational difficulty of the problem. Everybody knows, that it is easier to multiply two numbers (say 37 and 47), that to factor a given number (say 1739) into primes: we have to perform much more computational steps in order to factor a number that to multiply two numbers.

This intuitive notion of difficulty has a formal counterpart—the notion of computational complexity. This notion enables us to classify computational tasks according to their difficulty.[9] Notice, that we mean *computational* not conceptual complexity. The algorithm for checking, whether α is a tautology of the propositional calculus is conceptually very simple. But this method requires exponentially many computational steps.[10]

[8]We can describe any Turing machine in a binary code, so we can also present this description M to another machine U as input data. In particular, we can present the description M and the initial data x to the Turing machine U and ask the question, whether M will stop when presented with x as input. It turns out, that such a general decision algorithm (which would yield the answer for every possible M and x) does not exist.

[9]Here we are interested in the complexity in time only; we do not consider the space limitations.

[10]For n variables, the corresponding matrix has the size 2^n. For $n = 10$, we have to check up 1024 cases, for $n = 20$ the number increases to 1048576. As the increase is exponential, so it is impossible to solve the problem in a computationally tractable way e.g. for $n = 300$. Our lives are too short, and our universe is too small to perform this task.

The satisfaction problem SAT for propositional calculus (i.e. the problem of checking, whether a valuation v which makes the formula α true exists) is a nice example of a computationally difficult problem. The only known algorithm for solving SAT needs an exponential time to solve it, as we have to take into account all the 2^n valuations. SAT is therefore an example of a hard (computationally difficult) problem. We will call a problem *tractable*, if it can be solved in polynomial time, while *intractable* problems are those which can only be solved in an exponential time. That leads us to an important notion of two classes of computational problems. The class **P** (for Polynomial) is the class that contains all the computational decision problems that can be *solved* by a deterministic Turing machine in polynomial time. So the P-class contains tractable problems. The class **NP** (for Non-deterministic Polynomial) contains all those computational decision problems whose *proposed* solution can be *verified* by a deterministic Turing machine at polynomial cost. In short: you can find a solution to a P-problem in a polynomial time, but in the case of a NP-problem you can *check* in polynomial time, that the proposed solution in fact is a solution.[11] But where does the 'N' (for 'Non-deterministic') come from? It is because we can also consider non-deterministic Turing machines—i.e. machines, which have a slight different transition function, which allows them to choose freely between many possibilities at any step. The cost of the solution is defined as the shortest computational path—i.e. we allow our Turing machine to make clever guesses.[12]

An important subclass of NP-problems consists of the NP-complete problems. This class has a very important property: either all NP-complete problems are tractable, or none of them is.[13] This is because all the problems in this class are polynomially reducible to each other: if we find a polynomial solution to one of these problem, then we will be able

[11] A simple example may illuminate the point: it is easy to verify, that the prime factors of 851 are 23 and 37—provided, we are given these numbers in advance. So if you guess thus solution, you need only a polynomial time to check it up.

[12] The number 851 from the last footnote can be factored non-deterministically within a polynomial time: our Turing machine chooses the shortest part, leading to the desired result.

[13] Observe, that NP is defined as the class of problems, for which a polynomial *verification* exists. The possibility, that a polynomial solution algorithm exists remains open (even if it may not seem very probable).

to reformulate this algorithm to polynomial algorithms for all the other NP-complete problems. And any NP-problem reduces (polynomially) to any of the NP-complete problems.[14]

SAT is an NP-problem: there is a non-deterministic polynomial algorithm solving it. This is quite obvious: you just guess the correct answer (i.e. a valuation satisfying the formula α) and check within a polynomial time, that this answer is in fact the correct one. Moreover, SAT is also NP-complete: any other problem in the class NP is polynomially reducible to SAT. That means, that if we find a polynomial algorithm for solving SAT, we will also have polynomial algorithms for solving all the other problems from the class NP. That means in particular, that finding an effective, computationally tractable algorithm for solving the SAT problem would be of great importance for a whole class of problems. However, it does not seem very probable that such a solution can be found.[15]

But the practical intractability of computational problems can be advantageous. For example, the fact, that factoring natural numbers into primes is computationally difficult lies at the heart of many contemporary cryptographic protocols. But there is a quantum algorithm which solves the factoring problem within a polynomial time. If quantum computers existed, the RSA cryptographic protocol could be cracked quite easily.

Why is such an increase in speed possible? This is because in QCT we make use of some special features of the quantum world. In the classical model (Turing machine) we only transform 0-1 sequences in a mechanical way, not appealing to the laws of quantum mechanics.[16] We do not need quantum mechanics in order to explain, how the 0-1 strings of bits evolve within the Turing machine (or the classical computer). Things are quite different in the case of quantum algorithms.

[14] Cf. e.g. [Garey, Johnson 1979], [Hopcroft, Ullman 1979].

[15] If there is a polynomial solution to the SAT problem, then P = NP. The question, whether P = NP is in open problem—probably the most famous open problem in theoretical computer science.

[16] A Turing machine could be made of wood or steel and powered by horses or by steam, as no quantum phenomena are in use.

2. Computations in the quantum world

2.1. Elementary notions of QCT

From the classical point of view, the basic unit of information is a bit, which is either 0 or 1. A full description of the state of the bit is therefore given by a single number: 0 or 1.

In QCT, the basic unit of information is a qubit—the quantum counterpart of the bit. Qubits are much more complicated than bits, they cannot be described by the Boolean values 0 and 1, as they can occupy more states. The classical Boolean values have their counterparts—two distinguished (basic) states of the qubit, usually denoted by $|0\rangle$ and $|1\rangle$. But qubits can be also in a superposition of $|0\rangle$ and $|1\rangle$—i.e. they are (speaking in informal terms) somehow in both states at the same time. Such a superposition is described by the expression $a_0|0\rangle + a_1|1\rangle$, where the coefficients a_0 and a_1 are two complex numbers such that $|a_0|^2 + |a_1|^2 = 1$. From the formal point of view, a qubit is a vector of length 1 in a two-dimensional complex Hilbert space.[17] A physical realization of a qubit is e.g. a photon (there are numerous other examples), but in this paper we are not interested in the "hardware", but rather in the theoretical foundations of QCT.[18]

The classical computation consists of computational steps, transforming the initial 0-1-sequence of bits.[19] We just put the initial date on the tape of the Turing machine, and start the process. After a finite number of steps (provided the machine indeed halts, but here we restrict our attention only to such cases) the machine enters the terminal state, and the tape contains the solution. Of course, at any moment of the computation, the tape contains a certain finite 0-1 string.

[17] Readers who are not familiar with the concept of the Hilbert space can just think of the qubits as of certain combinations of two basic objects.

[18] The fact, that the coeffictients are complex numbers makes the situation somehow peculiar (from our classical point of view)—we can perfectly imagine mixing up e.g. two different colors in proportions 73% and 27%, but we cannot mix them up in proportions $(1 + i)/2$ and $(1 - i)/2$. How could we mix up portions of information with complex coefficients? It does not make any sense from the classical point of view, as the Turing machine transforms only strings of classical bits, and not their superposition.

[19] Remember, that we can encode any alphabet by 0-1 strings, so we can assume, that we are interested in 0-1 strings only.

The technical details are not of primary importance here and will often be omitted. What is really interesting is *how* a quantum computation enables us to make use of the very special features of the quantum world.

2.2. Quantum register. Entanglement

In QCT we consider not only single qubits, but also strings of qubits—quantum registers. Such registers have some quite special properties, which have no counterpart in the classical world. Quantum algorithms make use of these properties, and this makes them (at least in some cases) extremely powerful in comparison with the available classical algorithms.

A single qubit has the form $a_0|0\rangle+a_1|1\rangle$—i.e. we need 2 complex numbers to describe its state. How many complex parameters are needed to describe the state of a quantum register—e.g. a system of 10 photons? Consider first a system of 10 points on the plane, and imagine that we are interested in the positions of the points only. For each of the points, we need two parameters (x, y) in order to describe its position, so 20 parameters are sufficient to provide a complete description of the system of 10 points. This is quite obvious: every point has a position on its own, independent of the other points, and in order to describe the position of a single point we do not have to worry about the other points (why should we?). Therefore, one might be tempted to think, that we also need 20 parameters in order to describe the quantum system of 10 qubits (photons). But in the quantum world things are not always so simple: it can happen, that the qubits constituting the quantum register do not have a state on their own. That means, that the register of 10 qubits is in a certain state *as a whole*, but it does not make sense to speak of the states of the individual qubits. In such cases we speak of quantum entanglement (a notion, which will be defined in a more precise way later). Because of that fact, the description of the quantum system is much more complicated than a person trying to apply intuitions from classical physics might expect. The dimension of the system increases exponentially with the increase of the number of qubits: in the general case, the description of the quantum register consisting of n qubits requires 2^n parameters (as this is the dimension of the Hilbert space needed to provide the description).[20] Of course, we could not even dream of writing down such a description for $n = 100$.

[20] In order to describe the state of the quantum register of 10 qubits we need in the general case $2^{10} = 1024$ parameters, not just 20. In other words, the dimension of the

This is very different from the case of classical bits of information. To understand the underlying mechanism, consider the example of two qubits treated as one single quantum system. The states of the qubits are $a_0|0\rangle + a_1|1\rangle$ and $b_0|0\rangle + b_1|1\rangle$ correspondingly. The state of the quantum register can be written as a product (a tensor product) of these two states:

$$(a_0|0\rangle + a_1|1\rangle) \otimes (b_0|0\rangle + b_1|1\rangle)$$

(we will omit the symbol $\otimes$). If we treat this as an algebraic expression and perform the multiplication, we obtain:

$$a_0b_0|0\rangle|0\rangle + a_0b_1|0\rangle|1\rangle + a_1b_0|1\rangle|0\rangle + a_1b_1|1\rangle|1\rangle$$

To simplify the notation, we will write $|00\rangle$ instead of $|0\rangle|0\rangle$; $|01\rangle$ instead of $|0\rangle|1\rangle$ *etc.* The result is:

$$a_0b_0|00\rangle + a_0b_1|01\rangle + a_1b_0|10\rangle + a_1b_1|11\rangle.$$

The vectors $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$ form a basis for the 2-qubit quantum register. The vector $a_0b_0|00\rangle + a_0b_1|01\rangle + a_1b_0|10\rangle + a_1b_1|11\rangle$ describes the state of the 2-qubit register, such that the first qubit is in the state $a_0|0\rangle + a_1|1\rangle$, and the second in the state $b_0|0\rangle + b_1|1\rangle$. In general, the state of any two-qubit register can be described as: $c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$ (where c_{00}, c_{01}, c_{10}, c_{11} are complex numbers satisfying the condition $|c_{00}|^2 + |c_{01}|^2 + |c_{00}|^2 + |c_{11}|^2 = 1$).

Of course, the state $a_0b_0|00\rangle + a_0b_1|01\rangle + a_1b_0|10\rangle + a_1b_1|11\rangle$ is simply a product of two states: $(a_0|0\rangle + a_1|1\rangle)(b_0|0\rangle + b_1|1\rangle)$. But not every state of the 2-qubit register can be represented as a tensor product: in most cases, the vector $c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$ does not allow for such a factorization.[21] In such cases we say, that the two qubits are *entangled*. The presence of entanglement is a very special feature of quantum systems and has no classical counterpart.[22]

Hilbert space describing the system of 10 qubits is $2^{10} = 1024$ (from the formal point of view, this Hilbert space is the tensor product of the spaces corresponding to the qubits forming the register).

[21] This is only possible, when the system of equations: $c_{00} = x_0y_0$, $c_{01} = x_0y_1$, $c_{10} = x_1y_0$, $c_{11} = x_1y_1$, $|x_0|^2 + |x_1|^2 = 1$, $|y_0|^2 + |y_1|^2 = 1$ has a solution in complex numbers.

[22] A simple example of a two-qubit entangled register is $1/\sqrt{2}(|00\rangle + |11\rangle)$.

For three (and more) qubits, the situation is analogous: consider the system consisting of three qubits: $a_0|0\rangle + a_1|1\rangle$, $b_0|0\rangle + b_1|1\rangle$, $c_0|0\rangle + c_1|1\rangle$. Again, we multiply them just like algebraic equations, obtaining the product, which can be written down (using the obvious abbreviations, e.g. $|010\rangle$ instead of $|0\rangle|1\rangle|0\rangle$ *etc.*) as:

$$a_0b_0c_0|000\rangle + a_0b_0c_1|001\rangle + a_0b_1c_0|010\rangle + a_0b_1c_1|011\rangle + a_1b_0c_0|100\rangle + a_1b_0c_1|101\rangle + a_1b_1c_0|110\rangle + a_1b_1c_1|111\rangle.$$

The eight vectors $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$, $|111\rangle$ form a base of a 3-qubit register, and in the general case, the state of such a register can be presented as:

$$a_{000}|000\rangle + a_{001}|001\rangle + a_{010}|010\rangle + a_{011}|011\rangle + a_{100}|100\rangle + a_{101}|101\rangle + a_{110}|110\rangle + a_{111}|111\rangle$$

If such an expression cannot be presented as a tensor product of three qubits $a_0|0\rangle + a_1|1\rangle, b_0|0\rangle + b_1|1\rangle, c_0|0\rangle + c_1|1\rangle$ we are again confronted with quantum entanglement: the qubits forming the register do not have a state on their own.[23] In the general case of an entangled n-qubit register, we need 2^n components in order to describe the state of this register (the dimension of the corresponding Hilbert space is 2^n).[24] That shows in particular, that the computer simulation of the evolution of a quantum system in an efficient way is not possible: to describe the evolution of a system of n qubits, we would have to store (and describe the evolution of) 2^n values at once. Of course, this is not possible in the case of e.g. 300 qubits.

2.3. Quantum gates

We can view a classical computation as a action of a Boolean network on initial data.[25] Such a circuit consists of Boolean gates, and each gate

[23] An example of a three-qubit entangled register is $1/\sqrt{2}(|000\rangle + |111\rangle)$.

[24] If we can present the state of the n-qubit register as a tensor product we need $2n$ coefficients only. But in the general (entangled) case we need 2^n coefficients.

[25] A Boolean network is simply a directed graph, with nodes which are associated with Boolean functions (i.e. functions $f\colon \{0,1\}^n \to \{0,1\}^m$). Every gate transforms the input (which is a string of n bits) into an output (a string of m bits). Given a string of bits as input, such a circuit transforms it into another string. This model is (in a suitable sense) equivalent to the Turing machine model, and for the purpose of introducing quantum computations it is much more convenient.

transforms the initial string of bits into an output (which is another string of bits). The quantum counterpart of such a Boolean gate is a quantum gate, which acts either on one qubit or on a quantum register. The action of a quantum gate on a qubit can be described as:

$$V\colon a_0|0\rangle + a_1|1\rangle \to b_0|0\rangle + b_1|1\rangle,$$

where $a_0|0\rangle + a_1|1\rangle$ is the input (the initial state of the qubit), and $b_0|0\rangle + b_1|1\rangle$ is the output, i.e. the final state of the qubit (remember, that a_0, a_1, b_0, b_1 are complex coefficients). In the general case a quantum gate transforms an n-qubit register, giving another register as an output.

A quantum computation consists of a sequence of such transitions, performed on a quantum register. These quantum gates manipulate the information stored in the qubit, or in the quantum register (the system of qubits). From the mathematical point of view, quantum gates are linear unitary operations, i.e. they preserve the norm of the quantum state (but these technical details are not important here).[26]

Every quantum gate is a linear operator on the appropriate Hilbert space (the dimension of this space is 2^n, where n is the size of the register). Due to the linearity of the operator, it is sufficient to define the action of the operator on the basic states $|0\rangle$ and $|1\rangle$ of the particular qubits.[27] A simple example is the Hadamard gate H, which acts in the following way:

$$|0\rangle \to 1/\sqrt{2}(|0\rangle + |1\rangle),$$
$$|1\rangle \to 1/\sqrt{2}(|0\rangle - |1\rangle).$$[28]

A straightforward computation shows, that $H^2 = \mathrm{Id}$.

[26] A quantum gate acting on one qubit is a unitary matrix of dimension 2. A general quantum gate acting on n qubits is an unitary matrix of dimension 2^n. If U is such a gate, then $U^*U = \mathrm{Id}$ (identity), where U^* is the adjont of U (obtained from U by transposing the matrix and complex-conjugating it). These technical details are not so important here, but it should be mentioned, that quantum gates are reversible: for any quantum gate U there is another quantum gate V, which inverts U.

[27] If V is such an operator, the linearity of V means simply, that $V(a_0|0\rangle + a_1|1\rangle) = a_0V|0\rangle + a_1V|1\rangle$.

[28] In this paper we abstract from the physical details, but—for the sake of illustration—notice, that the Hadamard gate describes a photon passing through an interferometer. A simple description can be found e.g. in [Deutsch *et al* 2000].

Now consider the following thought experiment, which will give us some insight into the peculiarities of the quantum world and of quantum computation. Imagine a random device, which—independently of the input (which is 0 or 1) gives us as output 0 or 1 with the same probability ½ (e.g. a fair coin toss). If such an operation is performed twice (i.e. we simply concatenate two such devices M) it is quite obvious, that the output will likewise be random: we just feed the first device with the input 0 or 1, perform the random operation, observe the output, pass the output to the second device, perform the second random operation and observe the output. Of course the second output is random, regardless of the first outcome.

This is how the classical coin works. However, in the quantum world, strange things happen: we can set up a random operation U, with the strange property, that performing it twice gives us a deterministic outcome. This is of course very counterintuitive, as our intuitions are modeled by the classical (i.e. macroscopic) world. A classical coin like this does not exist. But the "quantum coin" acts in the following way:

$$U(0) = 0 \text{ or } 1 \text{ with probability } ½,$$
$$U(1) = 0 \text{ or } 1 \text{ with probability } ½.$$

But(!)

$$U^2(0) = 1\,,$$
$$U^2(1) = 0\,.$$

From the logical point of view, the operation U^2 is the negation. So U can be viewed as the square root of the negation. In classical logic, such a logical operation does not exist. But there is a quantum device, which acts exactly in this way—i.e. it can be (in a sense) viewed as an experimental realization of $\sqrt{\text{NOT}}$.

What is its formal counterpart? Consider the operator U, defined as:

$$U\colon |0\rangle \to ½(1-i)|0\rangle + ½(1+i)|1\rangle,$$
$$U\colon |1\rangle \to ½(1+i)|0\rangle + ½(1-i)|1\rangle.$$

A straightforward computation shows, that $UU|0\rangle = 1$, and $UU|1\rangle = 0$. $U^2(p) = \neg p$, so U is the square root of the negation.

But why do we claim, that $\sqrt{\text{NOT}}$, acting e.g. on photons is a random device? There is no sign of randomness in the definition of the operator U. Indeed, U acts on the vectors in the Hilbert space in a purely deterministic way. But in order to "extract" the information from the quantum system we have to perform a measurement. A measurement is probabilistic in nature, as stated by one of the basic postulates of quantum mechanics (which we remind here in a simplified form, concerning only qubits):

- If we perform the measurement on a qubit, being in the state $a_0|0\rangle + a_1|1\rangle$, there are two possible outcomes: 0 and 1. The probability of obtaining the result 0 equals $|a_0|^2$, and the probability of obtaining 1 equals $|a_1|^2$. After the measurement the state of the quantum system is projected onto one of the basic states: if the outcome was 0 (resp. 1), the state after the measurement is projected onto $|0\rangle$ (resp. $|1\rangle$).[29]

In particular, we usually cannot learn from the outcome of the measurement, what was the state of the system before the measurement (we cannot tell whether the state was e.g. $½(1-i)|0\rangle + ½(1+i)|1\rangle$ or rather $½i|0\rangle + \sqrt{¾}|1\rangle$). For example, if the outcome of the measurement was 0, the only information about the state of the system *before* the measurement, is that $a_0 \neq 0$. After the measurement the state of the system collapses to $|0\rangle$ (and of course the outcome of the next measurement will be 0 with probability 1). So (with few exceptions), the act of measurement causes changes in the state of the system.

Let us turn back to $\sqrt{\text{NOT}}$. It transforms $|0\rangle$ to $½(1-i)|0\rangle + ½(1+i)|1\rangle$. If we now perform the measurement, we will obtain 0 or 1 with equal probabilities ½. That means, that the procedure consisting of:

1. preparing the quantum system in the state $|0\rangle$,
2. applying the $\sqrt{\text{NOT}}$ operation to this system,
3. performing the measurement,

is a purely random procedure, just like tossing a random coin. But if we *do not* perform the measurement after the first application of $\sqrt{\text{NOT}}$,

[29] In particular, if the result of the measurement was e.g. 0, than the state is projected onto $|0\rangle$, and the next measurement will yield the result 0 with probability 1. The coefficients a_0 and a_1 are called *probability amplitudes*. This postulate explains the sense of the condition $|a_0|^2 + |a_1|^2 = 1$—this is to satisfy the additivity axiom in probability theory.

but apply $\sqrt{\text{NOT}}$ again, the whole procedure will transform 0 into 1 and 1 into 0 in a deterministic way. This looks strange, but remember, that we do not perform the measurement after the first application of the $\sqrt{\text{NOT}}$ gate, but we transfer the result to the second gate—and the measurement is performed *after* the second application of $\sqrt{\text{NOT}}$. Of course, if we measured the state of the system after the first $\sqrt{\text{NOT}}$, the state of the quantum system would collapse (become either $|0\rangle$ or $|1\rangle$), so the input of the second gate would be either $|0\rangle$ or $|1\rangle$, and the second measurement would yield either 0 or 1 with the same probability.

As was already mentioned before, in the general case the measurement does not give any information about the state of the system *before* the measurement. So, if we perform a quantum computation and afterwards perform the measurement of the system, we usually will not be able to tell, what was the state of the system *before* the measurement. That means, that in the general case during the measurement we lose the information that was obtained in course of the computation. But in some interesting cases, some additional information about the evolution of the system will make it possible to deduce the final state of the system from the result of the measurement—and in these cases we will be able to harness quantum mechanics in order to produce effective information processing procedures.

Two simple examples may illuminate the point:

(a) Consider one qubit, which is known *in advance* to be in one of the two states from the computational basis (i.e. it is either $|0\rangle$ or $|1\rangle$, but we do not know, which one). Of course, in that case the outcome of the measurement gives us information about the internal state.

(b) Consider a 2-qubit quantum register, which is known *in advance* to be in one of the two states:

$$\Phi_0 : 1/\sqrt{2}(|00\rangle + |01\rangle),$$
$$\Phi_1 : 1/\sqrt{2}(|10\rangle + |11\rangle).$$

If we perform the measurement on the *first* qubit, we will obtain the complete information about the state of the *whole* register, as we could obtain 0 only when the state of the register is Φ_0 (and analogously, *only* Φ_1 can result in obtaining 1). That means, that in some cases the outcome of the measurement allows us to "extract" information about the state of the system before the measurement. We can make use of this fact in quantum algorithms, the simplest of which is Deutsch's algorithm.

2.4. Examples of quantum algorithms[30]

Stated in colloquial terms, we have to decide, whether a coin is a genuine coin or not (in which case it has two tails or two heads). Of course, classically we have to look at the coin twice. The mathematical counterpart is as follows: for a given function $f\colon \{0,1\} \to \{0,1\}$, we have to find out, whether the function is constant ($f(0) = f(1)$—this corresponds to the fake coin), or whether it is balanced ($f(0) \neq f(1)$—a genuine coin). How many function evaluations are necessary in order to find out, whether f is constant or balanced? Classically—of course two: we have to evaluate both the values $f(0)$ and $f(1)$, and then compare them. In the quantum world we can do this in one quantum step.

Consider a quantum "black box" U_f, which acts on a two-qubit register in the following way:

$$U_f\colon |x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle,$$

where $\oplus$ is addition *modulo* 2 (i.e. $0 \oplus 0 = 0$; $0 \oplus 1 = 1$; $1 \oplus 0 = 1$; $1 \oplus 1 = 0$).[31] It turns out, that it is possible to find out the value $f(0) \oplus f(1)$ applying the procedure U_f only once.[32]

The Deutsch algorithm proceeds as follows (to simplify matters I will omit all the coefficients like $1/\sqrt{2}$, ½ *etc.*):

1. We prepare two qubits in initial states $|0\rangle$ and $|0\rangle - |1\rangle$, so the state of the register can be written down as the product $|0\rangle(|0\rangle - |1\rangle)$. We apply the Hadamard gate to the first qubit, and identity (i.e. do nothing) to the second qubit:[33]

$$H\colon |0\rangle(|0\rangle - |1\rangle) \to (|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$$

[30] We present the simplest possible algorithm here. The interested reader can find more technical details concerning quantum algorithms e.g. in [Nielsen Chuang 2000], [Hirvensalo 2001].

[31] Formally, we apply the identity operator to the first qubit; evaluate $f(x)$ on the first qubit and add the result *modulo* 2 to the value of the second qubit.

[32] If f is balanced, than $f(0) \oplus f(1) = 1$, if it is constant, $f(0) \oplus f(1) = 0$. So we do not have to compute the particular values $f(0)$ and $f(1)$ in order to answer the question. In a sense, we are only using the minimal information available.

[33] This means, that after this operation the state of the first qubit is $|0\rangle + |1\rangle$, and the second $|0\rangle - |1\rangle$. The operation should formally be written as $H \otimes \mathrm{Id}$, but we omit these details.

2. Now we apply the quantum gate U_f. A straightforward computation shows, that:

$$U_f\colon (|0\rangle + |1\rangle)(|0\rangle - |1\rangle) \to ((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)(|0\rangle - |1\rangle).$$

The state of the first qubit is therefore

$$(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle$$

(the state of the second qubit is $|0\rangle - |1\rangle$, but that is of no further relevance to our problem). There are two possibilities:

(i) If $f(0) = f(1)$, then the state of the first qubit is either $|0\rangle + |1\rangle$ (this happens when $f(0) = f(1) = 0$) or $-(|0\rangle + |1\rangle)$ (this happens when $f(0) = f(1) = 1$).

(ii) If $f(0) \neq f(1)$, then the state of the first qubit is either $|0\rangle - |1\rangle$ or $-(|0\rangle - |1\rangle)$.

Now we apply the Hadamard gate again to the first qubit. The result is:

- if (i) was the case: $|0\rangle$ or $-|0\rangle$,
- if (ii) was the case: $|1\rangle$ or $-|1\rangle$.

If we now perform the measurement on the first qubit, we obtain either 0 or 1, and in this particular case we are able to deduce, whether (i) or (ii) took place. This is because we knew in advance, that the qubit had to be in one of four particular states before the final measurement. Observe, that in this algorithm we evaluated the function f *only once*. The measurement is performed *after* the algorithm terminates—otherwise we would cause the measured qubit to collapse and therefore destroy the computation.

The Deutsch algorithm is interesting, but it seems somehow artificial, and the increase in speed (1 call of the function f instead of 2 calls) is not very spectacular. But it has an interesting generalization: the Deutsch-Jozsa algorithm. Here we have a function $f\colon \{0,1\}^n \to \{0,1\}$ (so any 0-1 sequence of the length n is mapped either on 0 or 1). We know in advance, that f is either constant (all the values of f are 0 or all the values are 1), or balanced (which means, that f takes as many times the value 0 as the value 1). In the classical algorithm we have to call

f approximately 2^{n-1} times.[34] But the Deutsch-Jozsa algorithm solves this problem in a polynomial time.

2.5. The killer application?

The by far most impressive example of a quantum algorithm is Shor's algorithm,[35] which can be called one of the cornerstones of quantum computation theory. Shor's algorithm shows, how a problem believed to be hard becomes easy by referring to quantum mechanics.

The algorithm deals with the factorization problem, which belongs to the class **NP**: given a solution we can check within a polynomial time whether it is fact is the desired solution, but factoring a number into primes is a complicated task (try this with e.g. 1062347—which is not a very big number). Every known classical algorithm for factoring requires an exponentially increasing number of steps. This fact is exploited in cryptographic protocols: the security of these protocols relies on the assumption, that factoring is intractable. We will not go into the technical details of Shor's algorithm—it consists of a classical and a quantum part. In the classical part we exploit some number-theoretic results (concerning finding prime factors by determining a period of a certain function). The problem of factorization reduces to the problem of finding the period of a certain periodic function. This problem can be solved efficiently by Shor's quantum algorithm. In particular, Shor's algorithm provides a theoretical possibility of cracking the RSA code, the security of which rests on the assumption, that factoring is hard.

3. Final remarks

Quantum computers could be very powerful—so why there are none of them available? Not going into technical details, quantum registers are extremely fragile. One of the most technical problems is to prevent the surrounding environment from interacting with the qubit registers. In

[34] Of course, if the function is balanced and we are lucky, we obtain two different values in the first two evaluations, but it can also happen, that we obtain 2^{n-1} times the same value, and have to make one more evaluation in order to decide, whether f is constant or balanced.

[35] It was presented in 1994 ([Shor 1994]) and sparked a tremendous interest even outside the physics community.

a sense, we have to encapsulate the quantum computer and prevent it from losing the information in the environment (a decoherence, which would destroy the computation). That means, that there are formidable technical problem to be overcome before a quantum computer can be build.[36]

But in spite of these practical problems, I think that the area of quantum computing has a profound impact on our understanding of some classical philosophical and methodological notions. These issues will be discussed in the subsequent paper. In particular, I will discuss the philosophical impact which this theory has on philosophy of mathematics, and—in particular—I will examine the thesis, that the best explanation of the status of mathematical knowledge compatible with the advances in QCT if offered by the quasi-empiricist stance (which incorporates mathematical knowledge into our "web of belief", including also scientific knowledge).

Acknowledgments. This paper was supported by the KNF grant N N101094136.

References

Aharonov, D., 1998, "Quantum computing", *Annual Review of Computational Physics* VI, Singapore: World Scientific. Available on-line: http://arxiv.org/PS_cache/quant-ph/pdf/9812/9812037v1.pdf

Davis, M., 2006, "Why there is no such discipline as hypercomputation", *Applied Mathematics and Computation* 178, 1: 4-7 (Special Issue on Hypercomputation).

Deutsch D., A. Ekert, and R. Lupacchini, 2000, "Machines, Logic and Quantum Physics", *The Bulletin of Symbolic Logic* 6, 3: 265–283.

Feynman, R.P., 1982, "Simulating physics with computers", *International Journal of Theoretical Physics* 21: 467–488.

Garey, M. R., and D. S. Johnson, 1979, *Computers and Intractability. A Guide to the Theory of NP-completeness*, New York, WH Freeman.

Hirvensalo, M., 2001, *Quantum Computing*, Springer-Verlag.

Hogarth, M., 1994 "Non-Turing computers and non-Turing computability", *PSA* 94, 1: 126–138.

[36] A presentation of these technical matters can be found in [Stolze, Suter 2004].

Hopcroft, J. E., and J.D. Ullman, 1979, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley.

Laraudogoitia, J. P., 1996, "A beautiful supertask", *Mind* 105: 81-83.

Nielsen, M. A., and I. L. Chuang, 2000, *Quantum Computation and Quantum Information*, Cambridge University Press.

Pitowsky, I., 1990, "The physical Church thesis and physical computational complexity", *Iyyun* 39: 81–99.

Shor, P., 1994, "Algorithms for quantum computation. Discrete logarithms and factoring", pp. 124–134 in: *Proc. 35th Annual Symposium on Foundations of Computer Science*.

Stannett, M., 2006, "The case for hypercomputation", *Applied Mathematics and Computation* 178, 1 (Special Issue on Hypercomputation).

Stolze, J., and D. Suter, 2004, *Quantum Computing. A Short Course from Theory to Experiment*, Wiley-VCH, Weinheim.

KRZYSZTOF WÓJTOWICZ
Department of Logic
Warsaw University
ul. Krakowskie Przedmieście 3
00-927 Warszawa, Poland
wojtow@uw.edu.pl