Interference as a Computational Resource: A Tutorial

Mika Hirvensalo
Department of Mathematics and Statistics
University of Turku
mikhirve@utu.fi

September 25, 2017

1 Introduction

Interference is obviously familiar, at least to some extent, to anyone having watched the waves propagating on a water surface. Falling rain drops create a sequence of nested circular, expanding waves. When two such wave patterns meet, sometimes the wave crests amplify each other, but sometimes the wave crest and hollow cancel each other, forming patterns which a single wave propagation never creates; see Figure 1. This is exactly how the interference in wave propagation is understood: The waves may amplify or weaken each other.

It may however require good luck or patience and careful design to observe and photograph clear, large, and systematic interference patterns on the water surface: The water surface should be calm except some individual points, where the creation of the circular waves should happen in constant time intervals.

The water waves are not the only ones where interference can be observed, rather very much on the contrary. A very useful and perhaps the well-known audio interference effect is the beat: the volume of the sound varying like tremolo if the sound sources are of slightly different frequency. For instance, a guitar tuner will press the lowest (E) string against the fifth fret and leave the second lowest (A) string free. Then picking both strings simultaneously will produce an audible tremolo effect if the guitar is not well tuned.

On the other hand, observation of light interference appears practically much more difficult. Since the days of Huygens and Newton, the scien-

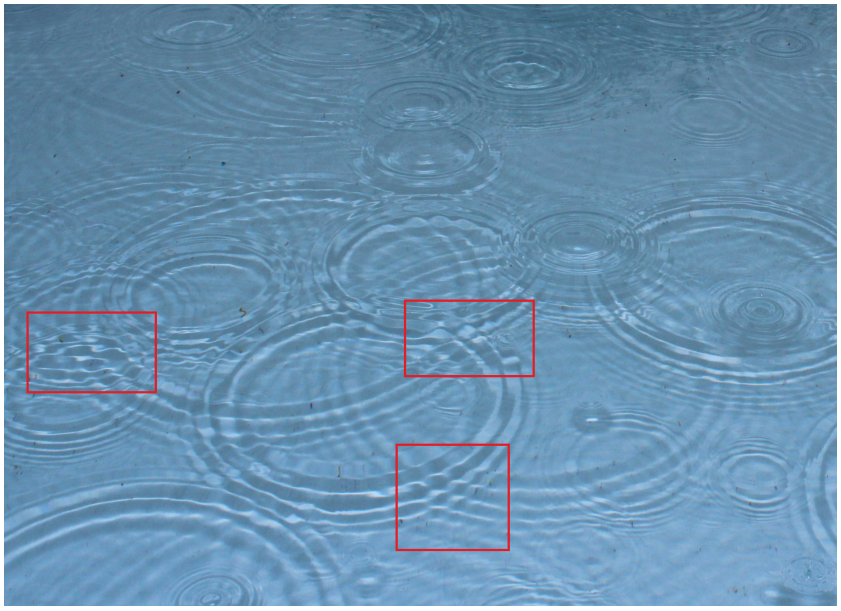

Figure 1: Water surface photographed in the rain. Waves crossing each other may either cancel or amplify each other, creating interference patterns. Some most visually notable interference patterns in this photograph are surrounded in a red box.

tific community had been debating over the nature of light: Huygens supported the undulatory explanation, whereas Newton favoured the corpuscular model. Certainly, a thin oil film on a water surface shone with all rainbow colours, but those days it was unknown that the origin of that beautiful visual phenomenon is light wave interference. Moreover, the commonplace experience has shown that two light sources such as candles never create any visible interference pattern like those observed in water.

English polymath Thomas Young was however very optimistic and believed that the lack of visually observed light interference patterns was due to the turbulent or mixed nature of the candle light or daylight. In the very early 19th century, Young devised an ingenious experiment demonstrating the undulatory nature of light. Young's idea was to imitate waves propagating on a water surface, and for that purpose, he needed to create two coherent light sources, hoping that the light emitted by the sources will create an interference pattern on the screen.

More than two hundred years ago, without modern day laser technology,

Young resolved the problem of creating coherent light for two light sources by letting the light pass through a small hole in the first screen, and then inserting an intermediate screen with two holes in between the first and the final one, see Figure 2. It should be noticed also that in early 19th century, there was no clear idea about the electromagnetic fields yet, but the idea of *aether*, a nonvisible substance filling the universe was already launched. Hence it might have been rather natural to assume that the aether could indeed be the substance where the light propagates as an undulatory motion.

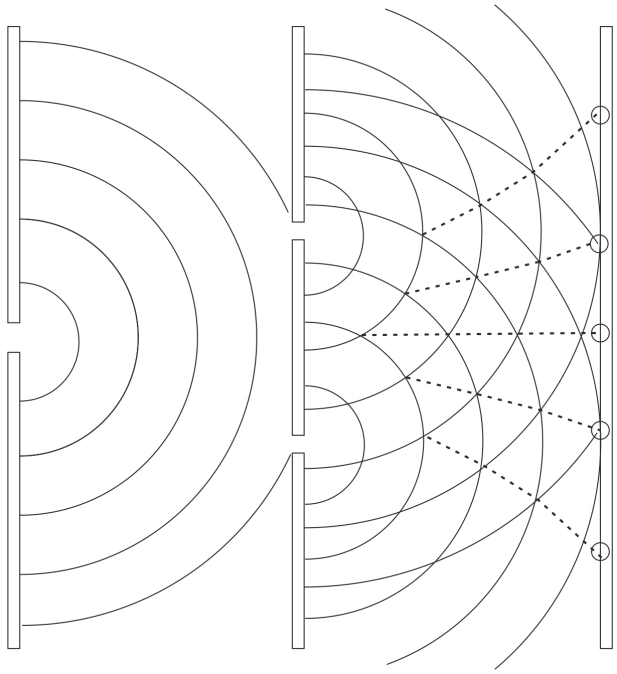

Figure 2: Young's interference experiment. The light source is located on the left, and one small hole in the first screen gives rise to coherent light waves which propagate to the next screen with two holes. The rightmost screen on the right displays the interference pattern. The arcs stand for the (light) wave crests, and the spots on the rightmost screen denote the most luminous areas in the interference pattern.

With such an ingenious experimental set-up, Young was the first to demonstrate that the light indeed has at least some inherently undulatory properties [29]. Young was even able to calculate the approximate wavelengths of different colours by using the estimate below Figure 3.

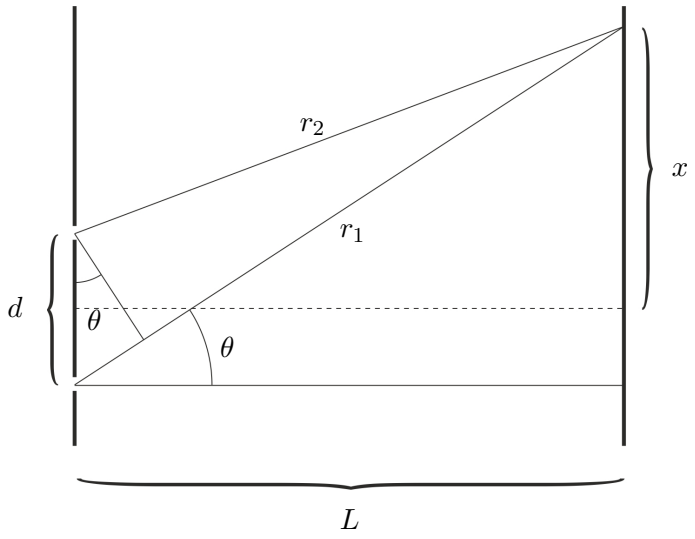

Figure 3: Two coherent light waves starting from the holes in the left screen interfere maximally when the wave crests meet exactly at the right screen; meaning that $r_1 - r_2 = m\lambda$, where λ is the wavelength of the light and m is an integer. If now d and x both are small compared to L, then $r_1 - r_2 \approx d \sin\theta$. Basing on the assumptions, θ is small, and then also $\frac{x}{L} \approx \tan\theta \approx \sin\theta$. It follows that $m\lambda \approx d\frac{x}{L}$. Using this approximation, Young could calculate the wavelengths of different colours of the visible light by measuring the light spot distances in the interference pattern.

These examples above – water, sound, and light waves – are by no means the only examples of physical wave interference. Water of course can be replaced with any other material with suitable molecular connections: Sound waves propagate as well in the air, water, oil, and even in iron rails. With a suitable experimental design, the interference can be apparently observed in any physical undulatory phenomenon.

Relatively soon after Young's experiment, the scientific community accepted the undulatory nature of the light. We cannot live the history again even partly with different premises, so we cannot be sure what would have become of Maxwell without the prior understanding of light as some kind of wave propagation. The only thing we can be sure about is that Maxwell mathematically demonstrated how the experimentally observed properties of electrical and magnetic phenomena can be combined into a general theory, which predicts a wave propagation: An alternation in the electric field,

implies an alternation in the magnetic field, which again implies an alternation in the magnetic field etc., and this is exactly how the electromagnetic waves propagate.

Maxwell was well aware of the wave-like properties of light and using the measurement-based estimates of that time for electric and magnetic constants, Maxwell was able to calculate the speed of electromagnetic radiation approximately as 310000 km/s, close enough to the speed of light measured then. Maxwell was then confident enough to draw the conclusion: Visible light and previously found infrared and ultraviolet seem to be only one narrow, special part of the wide range of electromagnetic wave propagation. During the 19th and 20th century, this picture has been further strengthened: an enormous variety of electromagnetic waves nowadays known as gamma rays, X-rays, ultraviolet, visible light, infrared, microwaves, and radio waves, all those were discovered in a relatively short time after Maxwell published his theory.

In the turn of 19th and 20th century, the physical sciences were being developed very enthusiastically: First, the mathematical machinery was improved for better presentation of already known physical structures, but also new explanations were found: It should be noted that those days, the view of atoms and molecules as the structural elements of the matter was not accepted by all notable scientists. However, the atom and molecule -based explanation of the matter and statistical thermodynamics supported each other.

Planck's 1900 interpretation of the black body radiation [21] supported the long forsaken corpuscular theory of electromagnetic radiation (the visible light as a special case), and Einstein's theory of the photoelectric effect eventually demonstrated that the understanding of light is incomplete, so there was evidently a need for a more sophisticated theory of the physical world. Einstein himself was indeed a driving force of the novel *relativity theory*, but via the explanation of the photoelectric effect, he become also one propelling force of *quantum physics*, which we will shortly focus on more precisely.

Quantum mechanics was created in the early 20th century because the classical mechanics was not able to explain newly discovered phenomena in the physical world. Quantum mechanical description of the physical world involves the idea of wave-particle dualism: All very small physical objects describable as particles may be portrayed as waves, equally well.

Even though quantum mechanics was already a well-established theory in 1930's when computer science sought its foundations through the seminal works of Church, Turing, and Kleene, it seems that the physical nature

of computers was not considered for many decades. One of the first observations – if not the very first one – of the implications of the quantum mechanics to the computational complexity was made by a most famous physicist, Nobel Prize winner Richard P. Feynman, who proposed in his seminal article [8] that a quantum physical system of R particles cannot be simulated by an ordinary computer without an *exponential* slowdown in the speed of the simulation. On the other hand, the simulation of a system of R particles in classical physics is possible with only a polynomial slowdown. The main reason for this is that the mathematical *description size* of a particle system is linear in R in classical physics but exponential in R according to quantum physics. As Feynman himself expressed:

> But the full description of quantum mechanics for a large system with R particles is given by a function $\psi(x_1, x_2, \ldots, x_R, t)$ which we call the amplitude to find the particles x_1, ..., x_R, and therefore, because it has too many variables, it *cannot be simulated* with a normal computer with a number of elements proportional to R or proportional to N. [8]

Number N in the previous citation refers to the accuracy of the simulation: "the number of points in the space", as Feynman formulates. In the same article, Feynman considered the problem of "negative probabilities", and returned to the same issue a couple of years later [9]. Feynman's approach may be earliest formulations to understand the role of interference in the probabilities induced by quantum mechanics.

In my opinion, it is justified to say that the quantum computing era started in 1982, and that the starting force was indeed the aforesaid Feynman's article. It is certainly true that Paul Benioff [4] and Yury Manin [17] both introduced their quantum mechanical model of computing, but Feynman appears to be the first one to suggest that the *computational complexity* may depend on the underlying physical model – at least for some computational tasks.

Following Feynman's idea and using quantum mechanical systems for bearing the information and carrying out the computation, it is possible to design algorithms that benefit from the interference: the undesired computational paths may cancel each other, whereas the desired ones may amplify. This phenomenon is generally believed to be the very source of the power of quantum computing.

In this tutorial, we are not going to refute the aforesaid picture about the power source of quantum computing. Instead, we are going to highlight some notable interference patterns used in famous quantum algorithms, but also to point out that the phenomenon itself – interference, and especially its destructive version – has been used as a computational resource even

before the quantum computing era.

Required preliminaries: This tutorial is not intended to be an introduction to quantum computing. Even though the basics on representing quantum information are reviewed in Section 3, it would be very useful – but not compulsory – to have some, even vague knowledge of quantum computing. If no such background exists, then a knowledge of basic linear algebra and an open mind to extend it to complex numbers is required. Also, some bachelor-level knowledge of group theory will be useful.

There will be examples requiring some knowledge of number theory and automata theory, but only the latter is essential when reading the last section. For that, the basic knowledge of deterministic and nondeterministic automata will be required, and the knowledge of probabilistic automata will be an advantage.

As the computational complexity also plays an important role here, we must assume some prior knowledge of Turing machines. Deterministic, nondetermistic, and time-bounded models are useful when defining the complexity classes. It is however not so important to know the quantum models of Turing machines, since the effect of interference can be studied via more elementary models as well.

2 Some remarks on computational complexity

The expression "computational resource" in the title can certainly be understood in many ways, but here we will mainly discuss the following two issues: First, how the quantum interference can be used to obtain time-efficient algorithms for some particular computational tasks and what can be obtained. Second, how the interference has been used to find *classical* devices for a computational task seemingly challenging to approach directly.

The second issue is treated in the final chapter, but for the first one, it is necessary to discuss the computational complexity briefly. For a concise introduction on the topic, see Papadimitriou's book [19].

It is rather difficult to trace the roots of nondeterminism, since the phenomenon itself has multiple connotations. Newtonian mechanics is strictly deterministic and nondeterminism therein arises only from the imperfect knowledge of the initial state and conditions of the system, hence this nondeterminism is of stochastic nature, which in principle could be removed or at least reduced by increasing the knowledge of the system. Apparently, inherent nondeterminism in the physical sciences arose rather late, but at the latest in the advent of quantum mechanics.

In mathematical terms, nondeterminism is not very hard to handle: replacing functions with a more general notion of relations will do the job. As the theory of computation is traditionally described in mathematical terms, the extension from deterministic to nondeterministic is quite trivial, but the question about the motivation remains still open: Why to introduce nondeterministic computation?

There are multiple reasons for studying nondeterministic computation, and as a mathematician, I like to emphasize the usefulness of nondeterminism as a theoretical tool for obtaining results which were far more difficult without nondeterminism.

Example 1. The *reversal* of regular language is regular [28], [7]. This is very easy to verify by using a nondeterministic automaton, but is more complicated by relying on deterministic constructions only.

Another, rather fundamental reason to study nondeterministic computing is that there is a variety of computational problems for which an efficient deterministic solution is not known, but if some slight uncertainty in the answer is allowed, then the solution becomes efficient.

Example 2. *Prime numbers* are important in public-key cryptography. The prime number theorem [12] says that $\pi(x)$, the number of primes in the interval $[2, x]$ satisfies

$$\lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\ln x}} = 1,$$

so asymptotically $\pi(x) \sim \frac{x}{\ln x}$. It follows that picking randomly $\ln x$ numbers around x you should be able to find a prime number about the magnitude of x with a non-negligible probability. It is not known how to find such a prime efficiently and deterministically, with probability of 1. And yet another story is how to *test* the primality efficiently.

Example 3. Given a prime number p, a *quadratic nonresidue* modulo p can be probabilistically found just by choosing any element $\mathbb{Z}_p^* = \{1, 2, \ldots p-1\}$. As half of the elements in $\mathbb{Z}_p^*$ are quadratic nonresidues, any choice is correct with 50% probability. Even the testing is efficient, but it is not known how to find a quadratic nonresidue modulo p efficiently, if no error probability is tolerated.

Another reason, different from the above, to study nondeterministic computing is the modelling of imperfect machines. An ideal computational machine always produces a unique successor from its current state, but in the

presence of physical imperfections, also other successors (with some probabilities) are possible.

From the viewpoint of this tutorial, the most important reason for introducing nondeterministic computing models is the nondeterministic nature of quantum mechanics itself. When encoding the bits in quantum mechanical systems and performing the computation based on those, it is impossible to avoid nondeterminism.

Nondeterministic computation can generally be depicted as a *computation tree*, where each node corresponds to a configuration of the computing machine. The root stands for the initial configuration, and the descendants of any node n correspond to the configurations reachable from n. Even assuming that the computation from the root to each leaf is efficient (polynomial time), the general problem in comparing any nondeterministic and deterministic computation is that in the former, the number of the leaves may be exponential in the length of the computation, whereas a deterministic procedure gives only one single outcome. See Figure 4 for a general view on nondeterministic computing.

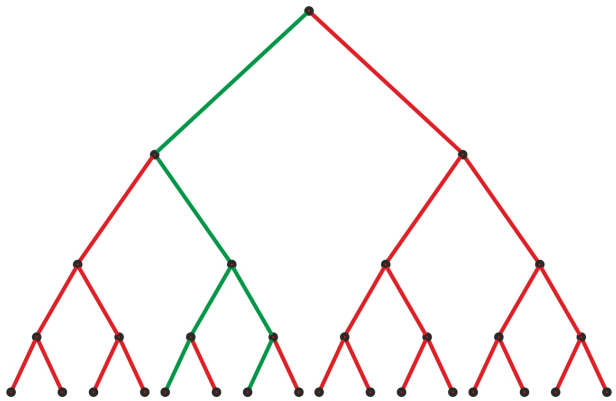

Figure 4: Computation starts at the root, and each node may have multiple successors. The leaves correspond to the final configurations, and it may be possible that only a tiny fraction of the computational paths is successful (green in this figure).

The above considerations will raise a plenty of interesting complexity classes; here we will only mention some of the most relevant for this tutorial. A reader willing to learn more about the complexity classes is recommended to consult Papadimitriou [19], if a formal and detailed representation is

required, or Aaronson [1] for informal and even entertaining presentation.

The most usual way to define the complexity classes goes through Turing machine formalism, which is accepted as the basic model of computing. While clumsy for any practical programming, the Turing machines still have the advantage of a well-defined notions of the time and space needed for computation. Moreover, the Turing machine formalism is very tolerant: constant factors will not affect the complexity classes, and nondeterministic and probabilistic models generalizations are straightforward [19].

Almost all the below complexity classes require somewhat standardized models of computations. When defining the nondeterministic time complexity classes, it may be important to know that all computational paths are equally long. This may be achieved by running two Turing machines simultaneously, one designed for resolving the given problem, and the other one acting as a clock [19].

A *decision problem* is a computational problem with (yes,no)-answer, such as "is the given number a quadratic nonresidue modulo p?" or "is the given Boolean expression satisfiable?"

A generalization of the decision problems are the *function problems* [19], where, instead of simple decision (yes,no) (or equally well $\{0,1\}$), some non-binary value is required. For instance, given an integer n, it is a decision problem to find out whether n is composite or not. On the other hand, it is a *function problem* to discover the least factor of n larger than 1.

In many cases, the function problems can be converted into a sequence of decision problems. For instance, when looking for the least factor $d > 1$ of integer n, we may ask for the bits of d sequentially: least, second least, etc.

In this tutorial, we will speak about the following complexity classes:

- **P** (Polynomial time): The class of decision problems solvable deterministically in polynomial time with respect to the input size.
- **PP** (Probabilistic Polynomial time): The class of decision problems solvable nondeterministically in polynomial time so that more than half computational paths give the correct answer for "yes"-instances, but at most half computational paths accept a "no"-instance.
- **NP** (Nondeterministic Polynomial time): The class of decision problems solvable nondeterministically in polynomial time so for any "yes"-instance, there is at least one accepting computational path, and for "no"-instances there are none.[1]

[1] Equivalently, **NP** is the class of the decision problems that can be verified in polyno-

- **BPP** (Bounded-error Probabilistic Polynomial time): The class of decision problems solvable nondeterministically in polynomial time so that at least $\frac{2}{3}$ computational paths give the correct answer.
- **BQP** (Bounded-error Quantum probabilistic Polynomial time): The class of decision problems solvable by quantum Turing machine in polynomial time so that at least $\frac{2}{3}$ computational paths give the correct answer.
- **PSPACE** (Polynomial Space): The class of decision problems solvable deterministically in polynomial space, i.e. in $O(n^k)$ space for some k.
- **EXPSPACE** (Exponential Space): The class of decision problems solvable deterministically in exponential space, i.e., in $O(2^{n^k})$ space for some k.

In the continuation, we will refer to these complexity classes when discussing the effect and role of the interference. It is also possible to show that the classes defined via the proportion of the computational paths such as **NP**, **PP**, and **BPP** can be equally well defined by using nondeterministic machines using transition probabilities and defining the acceptance probabilities accordingly, see [19]. According to such definition,

- **P** stands for polynomial time computing with correctness probability 1.
- **PP** is asymmetric: For any "yes"-instance, the acceptance probability must be greater than $\frac{1}{2}$, but all "no"-instances must be rejected with a probability of greater than *or equal* to $\frac{1}{2}$.
- **NP** is also asymmetric: For any "yes"-instance, the correctness probability must be positive, but for any "no"-instance, it should be 1.
- **BPP** stands for polynomial time computing with correctness probability at least $\frac{2}{3}$.
- **BQP** stands for polynomial time quantum computing with correctness probability at least $\frac{2}{3}$.

Remark 1. According to the traditional view (from 1970's until now) of the computational complexity, polynomial time computing is practically feasible, whereas computational time exceeding all polynomials is considered unfeasible in practice.

mial time, when a certificate is given. See Papadimitriou [19] for details.

It is very much possible to criticize this viewpoint: n^{20} is a polynomial which initially grows very much faster than the non-polynomial $n^{\ln\ln\ln n}$. Yet, the latter is preferable time complexity for any practical computation whatsoever. In fact, the latter function exceeds the former only for values of n which are far beyond everyday human experience. Even to write such numbers in decimal is impossible as the needed digits will outnumber the elementary particles in the known universe.

Here we will not focus on this kind of criticism but ignore the slowly growing non-polynomial functions. Instead, we will sharpen the question: Which versions of the polynomial time computation, **P**, **NP**, **PP**, and **BPP** (or even **BQP**) should be considered "practically feasible?" The usual reply is: **BPP** (and therefore also **P**) for classical and **BQP** for quantum computers.

There is a very good reason why the class **PP** algorithms are not considered practically feasible. It becomes clear when thinking about the following example: Consider the Boolean satisfiability problem and resolve it by just guessing an assignment for all the variables of the input expression. Then check whether the assignment was satisfying, and in the positive case, reply "yes". However, in the negative case, reply "yes" or "no" just by flipping a fair coin.

Clearly this is a polynomial time nondeterministic procedure giving answer "yes" with a probability greater than $\frac{1}{2}$ if the input expression is satisfiable, but only with a probability of $\frac{1}{2}$ in the opposite case.

Such a procedure can hardly be regarded practical, since the "yes" and "no" answers are very hard to distinguish. In fact, it will take $\Omega(2^n)$ attempts on a probabilistic system to tell the difference between the answer probability $\frac{1}{2}$, and $\frac{1}{2}+\frac{1}{2^n}$. Hence **PP** cannot be considered as a "practical" model of polynomial time computing. For the same reason, **NP** cannot be seen "practical", either.

On the other hand, class **BPP** (or **BQP**) has the advantage of bounded-error property. Now, the Chernoff bound (see Papadimitriou [19]) implies that in order to reduce the error probability smaller than ϵ, it is sufficient to repeat the computation a fixed number N_ϵ times and finally decide by the majority vote. Hence $\frac{2}{3}$ is not crucial as the correctness probability, but any probability strictly greater than $\frac{1}{2}$ will do.

Hence it is widely agreed that **BPP** (or **BQP** in the quantum case) is the synonym for practically feasible computation.

There are definition-based easy-to-see inclusions such as $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PP}$ and $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP}$, but the complexity theory as known today

is unfortunately too weak to provide answers to questions such as $\mathbf{P} = \mathbf{BPP}$?, $\mathbf{P} = \mathbf{NP}$?, or $\mathbf{BPP} = \mathbf{BQP}$?, these are in fact the most challenging questions of the complexity theory. Although there is no proof even in the sight, there are good reasons to believe that the first equality is true but the second and the third one are not [1].

3 Mathematical model of quantum information

The mathematical representation of quantum mechanics developed by Dirac, Heisenberg, Pauli, Schrödinger, etc. was already in 1930's completed into a Hilbert space formalism mainly by John von Neumann [18]. In this representation, we are not going to explore the development or the mathematical structure of quantum mechanics in details. Instead, we refer to Hirvensalo [13] or [16] for the general Hilbert space structure of quantum mechanics and present here a simple version of *closed quantum systems*. A reader well aware of the mathematical representation of quantum systems may as well skip this section.

It should be noticed that the finite-dimensional model we are going to introduce is for the *discrete* quantum systems with only finitely many (perfectly) distinguishable observable values, but for practical computational and information processing purposes, this model is sufficient anyway.

For the sake of completeness, some words should be said about the computational models with an unbounded memory, as well. It is definitely true that limiting the memory will also limit the potentials of the computation, but for the purposes of this study, we need not to introduce any other than the trivial restrictions: For computational process lasting of T time steps, $O(T)$ space is certainly sufficient, and in many cases, we may be satisfied with even narrower space restrictions.

Hence it is not reasonable to introduce the infinite versions of Hilbert spaces, as for our computational purposes (computational time polynomial in the input size), we may be happy with the finite-dimensional picture, as well.

Description. An n-level quantum system refers to a physical system with n perfectly distinguishable physically observable values. The mathematical description of such a system consists of the following parts:

- An n-dimensional Hilbert space H_n. Any n-dimensional Hilbert space is isomorphic to $\mathbb{C}^n$, wherein the (Hermitian) inner product for vectors

$\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$ is defined as

$$\langle \boldsymbol{x} \mid \boldsymbol{y} \rangle = x_1^* y_1 + \ldots + x_n^* y_n,$$

where x^* stands for complex conjugation. This inner product is obviously linear with respect to the second argument $\boldsymbol{y}$, nondegenerate (meaning that $\langle \boldsymbol{x} \mid \boldsymbol{x} \rangle = 0$ only if $\boldsymbol{x} = \boldsymbol{0}$), and conjugate symmetric, meaning that $\langle \boldsymbol{x} \mid \boldsymbol{y} \rangle^* = \langle \boldsymbol{y} \mid \boldsymbol{x} \rangle$. The inner product in H_n also induces *the norm* by $||\boldsymbol{x}|| = \sqrt{\langle \boldsymbol{x} \mid \boldsymbol{x} \rangle}$.

- The *(pure) states* of an n-level quantum system are identified with the unit-length vectors in H_n. That is, if $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is an orthonormal basis of H_n, then any $\boldsymbol{x} = c_1 \boldsymbol{x}_1 + \ldots + c_n \boldsymbol{x}_n$ with $|c_1|^2 + \ldots + |c_n|^2 = 1$ is a (pure) state.

- Rather typically, there is a preferable basis, which we will call a *computational basis*, whose elements may be associated with some numerical values 0, 1, …, $n-1$. By convention, we may use the Dirac *ket* notation $\boldsymbol{x}_1 = |\, 0\rangle$, …, $\boldsymbol{x}_n = |\, n-1\rangle$ to stand for the preferred orthogonal basis.

- A special case of the aforesaid is *quantum bit*, *qubit* for short, which is, by definition, a two-level quantum system depicted in H_2. The Hilbert space associated to such a system may be equipped with orthonormal basis $\{|\, 0\rangle, |\, 1\rangle\}$, and a general state of such a system is a vector $c_0\, |0\rangle + c_1\, |1\rangle$, where $|c_0|^2 + |c_1|^2 = 1$.

- A physical *observable* with a maximal number of different values consists of an orthonormal basis $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$, together with the observable values $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$, each λ_i associated with $\boldsymbol{y}_i$.

- *The minimal interpretation* of the quantum mechanics is an axiom connecting the mathematical model to the physically observable reality. In this simple model, the minimal interpretation says that if the quantum system is in state $\boldsymbol{x}$, then for a given observable $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$ with associated values $\lambda_1, \ldots, \lambda_n$, the probability of observing values λ_i is given by $|c_i|^2$, where α_i is the coefficient of $\boldsymbol{y}_i$ in representation

$$\boldsymbol{x} = c_1 \boldsymbol{y}_1 + \ldots + c_n \boldsymbol{y}_n.$$

- *Closed time evolution.* In a closed quantum system, a state $\boldsymbol{x}$ evolves into $U\boldsymbol{x}$, were U is a *unitary mapping*, meaning that $UU^* = U^*U = I$,

where U^* stands for the complex conjugate of the transpose of the matrix representing U. Mapping U depends on the physical circumstances surrounding the studied system, and on the time interval between the starting and the observation moment. In the context of quantum computing, it is usually assumed that there is a fixed set of unitary mappings $U_1, \ldots, U_k$ that can be applied to quantum states, and these mappings are called *quantum gates*.

- *Compound quantum systems* are described by using the tensor product construction. If H_1 and H_2 are the Hilbert spaces of the partial systems, then the Hilbert space of the compound system is $H_1 \otimes H_2$. The basis of the compound system can be formed by taking all tensor product of the subsystems basis elements. In the context of quantum computing, the tensor sign $\otimes$ is usually omitted, and for instance $|\,0\rangle\otimes|\,0\rangle$, $|\,0\rangle\otimes|\,1\rangle$, $|\,1\rangle\otimes|\,0\rangle$, and $|\,1\rangle\otimes|\,1\rangle$ are written more shortly as $|\,0\rangle\,|\,0\rangle$, $|\,0\rangle\,|\,1\rangle$, $|\,1\rangle\,|\,0\rangle$, and $|\,1\rangle\,|\,1\rangle$ or even more shortly as $|\,00\rangle$, $|\,01\rangle$, $|\,10\rangle$, and $|\,11\rangle$. In particular, this means that the system of two quantum bits is described by using four-dimensional Hilbert space having $B = \{|\,00\rangle, |\,01\rangle, |\,10\rangle, |\,11\rangle\}$ as an orthonormal basis. A general (pure) state of two quantum bits is then described as

$$c_{00}\,|\,00\rangle + c_{01}\,|\,01\rangle + c_{10}\,|\,10\rangle + c_{11}\,|\,11\rangle,$$

where c_{ij} are complex numbers satisfying $|c_{00}|^2+|c_{01}|^2+|c_{10}|^2+|c_{11}|^2 = 1$.

- The above generalizes to multiple quantum systems. Especially, the mathematical description of the system n quantum bits is a 2^n-dimensional Hilbert space with orthonormal basis $\{|\,\boldsymbol{x}\rangle \mid \boldsymbol{x} \in \{0,1\}^n\}$. A general (pure) state of n qubit system then looks like

$$c_{0\ldots00}\,|\,0\ldots00\rangle + c_{0\ldots01}\,|\,0\ldots01\rangle + \ldots + c_{1\ldots11}\,|\,1\ldots11\rangle, \qquad (1)$$

where $c_{\boldsymbol{x}}$ are complex numbers satisfying $\sum_{\boldsymbol{x}\in\{0,1\}^n} |c_{\boldsymbol{x}}|^2 = 1$.

Recall that according to Feynman [8], the difficulty of simulating quantum systems efficiently is due to the fact that a full quantum mechanical description of an R-particle system requires an exponential amount of complex numbers. In the description of a n-qubit state (1) this is explicitly shown: The mathematical description of a (pure) state of a system of n quantum bits requires 2^n complex numbers.

Example 4. A quantum bit is described in a two-dimensional Hilbert space $H_2 = \mathbb{C}^2$. We may choose coordinate representations $|\,0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, (resp. $|\,1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$) and associate logical value 0 (resp. 1) to the basis vector $|\,0\rangle$ (resp. $|\,1\rangle$).

Now any quantum bit can be presented as a linear combination (which is called *superposition* in the context of quantum bits)

$$\boldsymbol{q} = c_0\,|\,0\rangle + c_1\,|\,1\rangle, \tag{2}$$

where $|c_0|^2 + |c_1|^2 = 1$.

If we now fix an observable $\{|\,0\rangle, |\,1\rangle\}$ with values $0 \leftrightarrow |\,0\rangle$, and $1 \leftrightarrow |\,1\rangle$, then the probablity of observing bit value 0 (resp. 1) is $|c_0|^2$ (resp. $|c_1|^2$).

Definition 1. Let

$$W = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

It is plain to verify that W is unitary. W is called *Walsh transform* or *Hadamard-Walsh* transform, and is an example of a nontrivial unary quantum gate.

A simple calculation also shows that $W\,|\,0\rangle = \frac{1}{\sqrt{2}}(|\,0\rangle + |\,1\rangle)$, and that $W\,|\,1\rangle = \frac{1}{\sqrt{1}}(|\,0\rangle - |\,1\rangle)$. Hence the probability of observing 0 (in the computational basis) in state $W\,|\,0\rangle$ is $\left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$. Then of course the probability of observing 1 in state $W\,|\,0\rangle$ is $1 - \frac{1}{2} = \frac{1}{2}$, which also could be derived analogously. For the continuation, it is useful to already now observe that if $x \in \{0,1\}$, then

$$\begin{aligned} W\,|x\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x\,|1\rangle) = \frac{1}{\sqrt{2}}((-1)^{0\cdot x}\,|0\rangle + (-1)^{1\cdot x}\,|1\rangle) \\ &= \frac{1}{\sqrt{2}}\Big(\sum_{y\in\{0,1\}} (-1)^{y\cdot x}\,|y\rangle\Big). \end{aligned} \tag{3}$$

4 Interference in Quantum Computing

4.1 General Principles

Interference of computational paths cannot exist unless there are more than one computational paths to interfere with each other, and the multiple paths

can occur only when nondeterminism takes place. Another obvious requirement for interference is that at least the final outcomes of the different computational paths can be somehow combined. That is, we should have the computational paths labelled with objects from some additive structure. Due to the mathematical formalism of quantum mechanics, this structure in quantum computing is naturally selected as $\mathbb{C}$, and the labels are called *amplitudes*. When connecting the computing model to the physical reality, there should evidently be some rule $\alpha \mapsto \mathbb{P}(\alpha)$ that associates a probability to label α. The minimal interpretation of quantum mechanics naturally suggests the Born probability rule $\alpha \to |\alpha|^2$.

In fact, we have not yet defined what *interference* exactly means, so let us give here at least an informal description: Assume that the configuration c can be nondetermistically reached from the initial one via $k \geq 2$ computational paths with labels (amplitudes) $\alpha_1, \ldots, \alpha_k$. Then the total weight of the configuration c simply becomes $\alpha_1 + \ldots + \alpha_k$. We say that interference is *constructive*, if $\mathbb{P}(\alpha_1 + \ldots + \alpha_k) \geq \mathbb{P}(\alpha_1) + \ldots + \mathbb{P}(\alpha_k)$, and *destructive*, if $\mathbb{P}(\alpha_1 + \ldots + \alpha_2) < \mathbb{P}(\alpha_1) + \ldots + \mathbb{P}(\alpha_k)$. In the context of quantum computing, constructive interference is characterized with the property $|\alpha_1 + \ldots + \alpha_k|^2 \geq |\alpha_1|^2 + \ldots + |\alpha_k|^2$. In the case of destructive interference, we may say that the computational paths *cancel* each other (at least partially) and in the case of constructive interference, the paths amplify each other.

It should be noticed that in the computational models with limited number of configurations such as finite automata, some configurations are likely to reappear multiple times in computational paths, but the reoccurrence may happen even when if number of potential configurations is finite.

Remark 2. In a classical probabilistic model for computing, the "labels" are simply the probabilities, and the rule for associating the probability to a label is the identity mapping $\mathbb{P} : p \mapsto p$. Moreover, as the probabilities are all nonnegative, all interference in the classical probabilistic computing is always constructive, meaning that the computational paths cannot cancel.

The time development of even a single quantum bit may result in a nontrivial interference pattern, as we will see shortly. Later we will see some examples on how interesting interference patterns can be found in quantum computing more generally.

A very simple example of interference is given by the single-qubit state $|\,0\rangle$ when Hadamard-Walsh transform W is applied twice on it. Figure 5 contains a scheme of such computation.

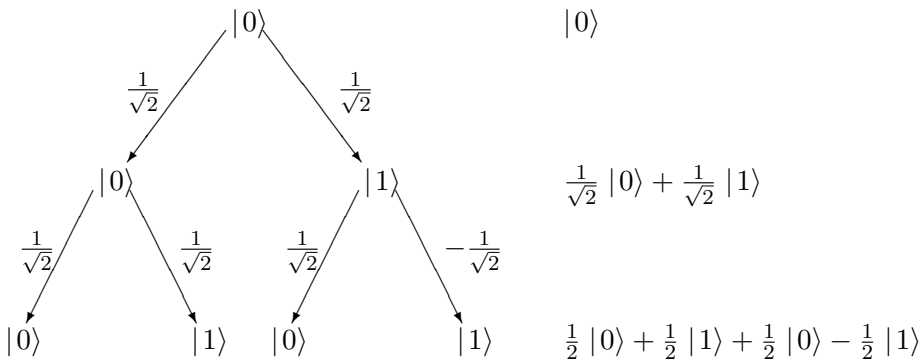

Figure 5: Hadamard-Walsh-gate twice. The left-hand side depicts how the application of W operates on states, whereas the corresponding states are written on the right side.

The top row of the figure contains the initial state $|\,0\rangle$ with amplitude 1. When applied once, W "splits" the state $|\,0\rangle$ into successors $|\,0\rangle$ and $|\,1\rangle$; both are produced with the amplitude $\frac{1}{\sqrt{2}}$. This is depicted in the figure's middle row. The second application of W "splits" $|\,0\rangle$ as before, but $|\,1\rangle$ is split slightly differently; now $|\,1\rangle$ is produced with amplitude $-\frac{1}{\sqrt{2}}$. The bottom row of the figure describes the final state. Now, the amplitudes in the bottom row can be computed by following the path from top to bottom and multiplying all the amplitudes along the path. For example, the amplitude of the leftmost $|\,0\rangle$ in the bottom row is $\frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}} = \frac{1}{2}$, whereas the amplitude of the right-most $|\,1\rangle$ is $\frac{1}{\sqrt{2}} \cdot (-\frac{1}{\sqrt{2}}) = -\frac{1}{2}$.

Starting from root $|\,0\rangle$, there are four computational paths, each of length 2, having $|\,0\rangle$, $|\,1\rangle$, $|\,0\rangle$, and $|\,1\rangle$ as ending configurations, and $\frac{1}{2}$, $\frac{1}{2}$, $\frac{1}{2}$, and $-\frac{1}{2}$ as their amplitudes.

Now that there are multiple occurrences of the same final configuration ($|\,0\rangle$ and $|\,1\rangle$), interference is expected. In fact, the final amplitude of $|\,0\rangle$ will become $\frac{1}{2}+\frac{1}{2} = 1$, and that of $|\,1\rangle$ will be $\frac{1}{2}-\frac{1}{2} = 0$. The computational paths with $|\,0\rangle$ as the final configuration hence interfere constructively, and those with $|\,1\rangle$ destructively. In this example, the 2nd and the 4th computational paths completely cancel each other.

Experience has shown that it may be rather challenging to design a strategy for quantum computing (i.e. a quantum algorithm) that would use the interference in a clever way, producing the desired result more efficiently than a classical strategy (algorithm).

There are (at least) four generally known ways for introducing desirable interference in quantum computing:

- Quantum Fourier Transform
- Grover search
- Quantum Random Walks
- Adiabatic quantum computing

4.2 Examples

To keep this tutorial in reasonable limits, we must again select one topic, and as historically Quantum Fourier Transform was the first strategy for controlling interference in quantum algorithms, we will here focus on that. A simple one-qubit version of that was used by David Deutsch in 1985 to demonstrate that quantum computers may be able to resolve some computational problems more efficiently than the classical ones [5]. Subsequently, in 1992, Deutsch and Josza extended the Deutsch' original version into a multiqubit promise problem [6], and Simon gave another generalization of their algorithm in 1994 [25]. As a simple variant of Quantum Fourier Transform can be expressed very easily by using Hadamard-Walsh transformation, we will shortly discuss it in more details.

Year 1994 should be considered as a very important landmark for quantum computing. It was 1994 when Peter Shor introduced his quantum algorithm for factoring integers and computing discrete logarithms. As Shor, himself pointed out, his procedures are generalization of that of Simon [23], and it is justified to say that Shor's algorithms were those that effectively raised quantum computing from a very marginal phenomenon into a greater public knowledge.

Fourier analysis, and its generalization *harmonic analysis* is certainly a most important mathematical machinery discovered since the birth of integral and differential calculus. Harmonic analysis and its applications can be used to decompose signals into linear combinations of monochromatic signals, a fundamental tool in signal processing. Unfortunately, the Fourier analysis falls beyond the scope of this tutorial, but we will simply present some examples most relevant to this tutorial.

Example 5. Let $\mathbb{Z}_2 = \{0, 1\}$ be the additive group of two elements. Clearly all functions $\mathbb{Z}_2 \to \mathbb{C}$ can be represented as $f(x) = \alpha_0 x + \alpha_1(1 - x) = \alpha_1 + (\alpha_0 - \alpha_1)x$, and noticing that $x = \frac{1}{2}(1 - (-1)^x)$ for $x \in \{0, 1\}$, we see

that in fact, each function $f : \mathbb{Z}_2 \to \mathbb{C}$ can be expressed in terms of two basis functions $B_0(x) = \frac{1}{\sqrt{2}}(-1)^{0\cdot x}$ and $B_1(x) = \frac{1}{\sqrt{2}}(-1)^{1\cdot x}$ (the reason for the coefficient $\frac{1}{\sqrt{2}}$ becomes evident soon). For any given

$$f = f_0 B_0 + f_1 B_1 \tag{4}$$

we then define $\widehat{f}(0) = f_0$ and $\widehat{f}(1) = f_1$, and substituting $x = 0$ and $x = 1$ in equation (4) gives us a matrix equality

$$\begin{pmatrix} f(0) \\ f(1) \end{pmatrix} = \underbrace{\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}}_{W} \begin{pmatrix} \widehat{f}(0) \\ \widehat{f}(1) \end{pmatrix}. \tag{5}$$

A direct computation shows that the Walsh matrix W is an involution (i.e. $W^2 = I$)[2], and hence the equation (5) implies

$$\begin{pmatrix} \widehat{f}(0) \\ \widehat{f}(1) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} f(0) \\ f(1) \end{pmatrix}. \tag{6}$$

The above two matrix equations (5) and (6) demonstrate a beautiful symmetry between the coefficients $(\widehat{f}(0), \widehat{f}(1))$ and $(f(0), f(1))$, and the symmetry is just one reason for using representation (4). This transformation $f \mapsto \widehat{f}$ is the simplest nontrivial example of a Fourier Transform and will be extended shortly.

4.3 Characters of Finite Abelian Groups Briefly

We will now focus on the generalizations of the basis functions B_0 and B_1 for groups larger than $\mathbb{Z}_2$. Ignoring the factor $\frac{1}{\sqrt{2}}$, functions like $\chi_0(x) = (-1)^{0\cdot x}$ and $\chi_1(x) = (-1)^{1\cdot x}$ are a special case of functions $f : G \to \mathbb{C}$ known as *characters*.

Definition 2. For a finite abelian group G, function $\chi : G \to \mathbb{C} \setminus \{0\}$ is *a character*, if $\chi(x + y) = \chi(x)\chi(y)$ for all $x, y \in G$.

The characters are very useful objects in the group theory, and their properties are well understood. Many basic properties are easy to derive, for instance $\chi(0) = \chi(0 + 0) = \chi(0)\chi(0)$, so $\chi(0) = 1$ is the only option. Moreover, $1 = \chi(0) = \chi(x-x) = \chi(x)\chi(-x)$, showing that $\chi(-x) = \chi(x)^{-1}$.

[2] In fact, the computation tree in Figure 5 shows that $W^2 \,|\, 0\rangle = |\, 0\rangle$, and it is equally easy to see that $W^2 \,|\, 1\rangle = |\, 1\rangle$

More importantly, as G is a finite group, $|G|\,x = 0$ for each group element, which implies that $\chi(x)^{|G|} = \chi(|G|\,x) = \chi(0) = 1$. Hence the character values lie in the unit circle and consequently $\chi(-x) = \chi(x)^{-1} = \chi(x)^*$.

We are not going to represent here the character theory in details, some most important features related to quantum computing can be found in [13]. There is however one structural property certainly worth mentioning, and towards it we give the following definition:

Definition 3. The *Hermitian inner product* for functions $f : G \mapsto \mathbb{C}$ is defined as

$$\langle f_1 \mid f_2 \rangle = \sum_{g \in G} f_1^*(g) f_2(g).$$

One of the most important property of the characters is introduced in the following lemma.

Lemma 1 (The orthogonality of the characters)**.** *Let χ_1 and χ_2 be characters of a finite abelian group G. Then*

$$\langle \chi_1 \mid \chi_2 \rangle = \begin{cases} |G|\,, & \text{if } \chi_1 = \chi_2 \\ 0, & \text{if } \chi_1 \neq \chi_2 \end{cases} .$$

We are not going the present the proof here, it can be found in [13]. Nevertheless, it is worth emphasizing that as the characters are orthogonal, they must also be linearly independent. As the dimension of the vector space of functions $G \to \mathbb{C}$ is clearly $|G|$,[3] then the characters form an (orthonormal) basis of that space, if there are $|G|$ distinct characters. It turns out that for each abelian group, this is indeed the case [13].

For any finite abelian group G, it is rather customary to label its characters by the group elements: $\{\chi_g \mid g \in G\}$ is the set of characters, and especially χ_0 stands for the trivial character. In fact, if we define the *product* of the characters as $(\chi_1\chi_2)(g) = \chi_1(g)\chi_2(g)$, it turns out that the characters form a group isomorphic to the original one!

Now that the characters χ_g form an orthogonal basis for function space $G \to \mathbb{C}$, we can introduce a renormalization factor $\frac{1}{\sqrt{G}}$ to obtain even an orthonormal set of basis functions $B_g = \frac{1}{\sqrt{|G|}}\chi_g$.

Then any function $G \to \mathbb{C}$ can be expressed in the form

$$f = \sum_{g \in G} \widehat{f}_g B_g. \tag{7}$$

[3] Characteristic functions f_g defined as $f_g(g') = 1$ if $g = g'$ and 0 otherwise clearly form a basis (called the *natural basis*) for the function space $G \to \mathbb{C}$.

The following definition clearly extends the representation of Example 5, which was the prelude for the simplest nontrivial Fourier transform.

Definition 4. The (discrete) Fourier transform of a function f in equation (7) is a function $G \to \mathbb{C}$ defined as

$$\widehat{f}(g) = \widehat{f}_g.$$

Using the orthonormality of functions B_g, it is not too difficult to discover the coefficient $\widehat{f}$ in equation (7). In fact, it follows directly that

$$\langle B_h \mid f \rangle = \sum_{g \in G} \widehat{f}_g \langle B_h \mid B_g \rangle = \widehat{f}_h,$$

so

$$\widehat{f}(g) = \widehat{f}_g = \langle B_g \mid f \rangle = \sum_{h \in G} B_g^*(h) f(h) = \frac{1}{\sqrt{|G|}} \sum_{h \in G} f(h) \chi_g^*(h).$$

In general, the above Fourier Transform is referred as to Discrete Fourier Transform, but here we will omit the attribute "discrete".

4.4 Special cases

The group

$$\mathbb{Z}_2^n = \mathbb{Z}_2 \times \ldots \times \mathbb{Z}_2$$

is an n-fold direct product of $\mathbb{Z}_2$. Hence it is not surprising that the characters of $\mathbb{Z}_2^n$ are not difficult to express. In fact, it turns out that the n-fold product of characters of $\mathbb{Z}_2$ are indeed the characters of $\mathbb{Z}_2^n$. More precisely:

Lemma 2. *Let $\boldsymbol{y} \in \mathbb{Z}_2^n$. Then the function*

$$\chi \boldsymbol{y}(\boldsymbol{x}) = (-1)^{\boldsymbol{x} \cdot \boldsymbol{y}} = (-1)^{x_1 y_1 + \ldots + x_n y_n}$$

is a character of $\mathbb{Z}_2^n$.

All the aforesaid imply the following:

Corollary 1. *The Fourier transform $\widehat{f}$ of a function $f : \mathbb{Z}_2^n \to \mathbb{C}$ can be expressed as*

$$\widehat{f}(\boldsymbol{x}) = \frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{y} \in \mathbb{Z}_2^n} (-1)^{\boldsymbol{y} \cdot \boldsymbol{x}} f(\boldsymbol{y}) \tag{8}$$

Notice that the equation (6) can be understood in the following way: for any function satisfying $|f(0)|^2 + |f(1)|^2 = 1$,

$$f(0)\,|0\rangle + f(1)\,|1\rangle \tag{9}$$

is an eligible qubit state, and

$$W(f(0)\,|0\rangle + f(1)\,|1\rangle) = \widehat{f}(0)\,|0\rangle + \widehat{f}(1)\,|1\rangle, \tag{10}$$

which is to say that W performs the Fourier transform on the coefficients.

The generalization of the above formula (10) is straightforward: Instead of the group $\mathbb{Z}_2$, we study its n-fold Cartesian product $\mathbb{Z}_2^n$, representable with n qubits, and observe that

$$\begin{aligned} & (W \otimes \ldots \otimes W)\,|x_1 \ldots x_n\rangle \\ = \;& ((-1)^{0\cdot x_1}\,|0\rangle + (-1)^{1\cdot x_1}\,|1\rangle) \ldots ((-1)^{0\cdot x_n}\,|0\rangle + (-1)^{1\cdot x_n}\,|1\rangle) \\ = \;& \frac{1}{\sqrt{2^n}} \sum_{(y_1,\ldots,y_n)\in\{0,1\}^n} (-1)^{x_1\cdot y_1+\ldots+x_n\cdot y_n}\,|y_1 \ldots y_n\rangle \end{aligned} \tag{11}$$

Denoting $W^{\otimes n} = W \otimes \ldots \otimes W$, (11) can be written more compactly as

$$W^{\otimes n}\,|\boldsymbol{x}\rangle = \frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{y}\in\mathbb{Z}_2^n} (-1)^{\boldsymbol{x}\cdot\boldsymbol{y}}\,|\boldsymbol{y}\rangle. \tag{12}$$

In order to get a better picture on how the Walsh transform $W^{\otimes n}$, which is a special case of Fourier transform, is related to quantum computing, let us study any function $f : \mathbb{Z}_2^n \to \mathbb{C}$ satisfying

$$\sum_{\boldsymbol{x}\in\mathbb{Z}_2^n} |f(\boldsymbol{x})|^2 = 1.$$

In fact, if f is not identically zero, this condition can always be forced by multiplying f with a constant. Because of this condition, vector

$$\sum_{\boldsymbol{x}\in\mathbb{Z}_2^n} f(\boldsymbol{x})\,|\boldsymbol{x}\rangle \tag{13}$$

in H_{2^n} has unit norm and is hence perfectly eligible n-qubit pure state. Now, the effect of $W^{\otimes n}$ on state (13) can be discovered by straightforward

computation:

$$\begin{aligned} & W^{\otimes n} \sum_{\boldsymbol{x}\in\mathbb{Z}_2^n} f(\boldsymbol{x})\,|\,\boldsymbol{x}\rangle = \sum_{\boldsymbol{x}\in\mathbb{Z}_2^n} f(\boldsymbol{x}) W^{\otimes n}\,|\,\boldsymbol{x}\rangle \\ = & \sum_{\boldsymbol{x}\in\mathbb{Z}_2^n} f(\boldsymbol{x}) \frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{y}\in\mathbb{Z}_2^n} (-1)^{\boldsymbol{x}\cdot\boldsymbol{y}}\,|\,\boldsymbol{y}\rangle \\ = & \sum_{\boldsymbol{y}\in\mathbb{Z}_2^n} \frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{x}\in\mathbb{Z}_2^n} f(\boldsymbol{x})(-1)^{\boldsymbol{x}\cdot\boldsymbol{y}}\,|\,\boldsymbol{y}\rangle \\ = & \sum_{\boldsymbol{y}\in\mathbb{Z}_2^n} \widehat{f}(\boldsymbol{y})\,|\,\boldsymbol{y}\rangle = \sum_{\boldsymbol{x}\in\mathbb{Z}_2^n} \widehat{f}(\boldsymbol{x})\,|\,\boldsymbol{x}\rangle, \end{aligned}$$

meaning that the n-fold Walsh transformation $W^{\otimes n}$ actually performs the Fourier transform on the coefficients of superposition (13).

One notable feature here is that the complex coefficients (function f values) form a vector of length 2^n, but the Fourier transform on these coefficients can be performed by using only n quantum operations (operation W on each qubit).

On the other hand, the classical complexity of computing the Fourier transform of a sequence $(f(\boldsymbol{x}))_{\boldsymbol{x}\in\mathbb{Z}_2^n}$ of $N = 2^n$ complex numbers can be estimated as follows: A direct computation of a single value using formula (8) requires roughly $N = 2^n$ additions and multiplications. As there are $N = 2^n$ values to be calculated, we can conclude that the Fourier transform computed in this straightforward way needs $O(N^2)$ operations.

However, there is a known significant improvement: A subgroup $G = \{\boldsymbol{x} \in \mathbb{Z}_2^n \mid \boldsymbol{x}_1 = 0\}$ clearly has cardinality 2^{n-1}, and its coset $(1, 0, \ldots, 0) + \mathbb{Z}_2^n$ together with the subgroup cover the whole group. It turns out that using this covering, it is possible to compute the Fourier transform on $\mathbb{Z}_2^n$ by computing two Fourier transforms in $\mathbb{Z}_2^{n-1}$ and then combining them in a proper way, see [13] for this special case. This is an example of the well-known method of "divide and conquer" in computer science, in this case leading into the computational complexity of $O(nN) = O(N \log_2 N)$. The aforesaid algorithm for Fourier transform is known as the *Fast Fourier Transform* (FFT), and is the cornerstone of all modern signal processing: Fourier transforms are used in (almost) all contexts in signal transmission, data compression and analysis, etc.

Aforesaid classical complexity $O(N \log_2 N)$ of FFT is clearly outperformed by that of QFT having complexity $O(\log_2 N)$, but on the other hand, it is also true that QFT and FFT are incomparable in the sense that the former is performed on the *amplitudes* of the physical qubits, whereas

in the latter, the bits themselves present the numbers on which FFT is performed on. However, using the quantum version, we can see how the following problem can be resolved by using only a few quantum operations.

Deutsch-Jozsa problem: Given a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ which is promised to be either *constant* (meaning that either $f(\mathbb{Z}_2^n) = \{0\}$ or $f(\mathbb{Z}_2^n) = \{1\}$) or *balanced* (meaning that $f(\boldsymbol{x}) = 0$ for exactly half of $\boldsymbol{x} \in \mathbb{Z}_n$). Determine if f is constant or balanced.

In connection to the Deutsch-Jozsa problem it is assumed that the function f is so-called *black box function*, given not explicitly. This means that there is no given algorithm to compute f, but the value of f is obtained as an *oracle call* in a single computational step.

From the classical point of view, it is then obvious that the problem cannot be solved *with certainty* calling function f fewer than $2^{n-1}+1$ times: By evaluating the function on half of the elements of $\mathbb{Z}_2^n$, and always seeing e.g. 0 as the value, we still cannot be sure that the function f is constant, but there is still room for the possibility that on the other half of $\mathbb{Z}_2^n$ function f gets value 1.

It should be carefully noticed that the above argumentation relies strongly on the black box nature of f: If an algorithm for computing f is available, then it is possible, at least in principle, that the solution for Deutch-Jozsa problem can be derived from the algorithm with essentially less effort than evaluating f on 2^{n-1} elements of $\mathbb{Z}_2^n$.

Another notable point is that this argumentation is about only zero-error computing. A probabilistic classical solution is also easy: Just randomly pick two elements $\boldsymbol{x}_1$, $\boldsymbol{x}_2 \in \mathbb{Z}_2^n$ and evaluate f on them. If $f(\boldsymbol{x}_1) = f(\boldsymbol{x}_2)$, then decide that f is constant, otherwise that f is balanced.

However, whereas the classical zero-error solution appears to require at least $2^{n-1}+1$ evaluations of f, there is a quantum solution with only one single evaluation of f. It is, quite naturally, assumed that the evaluation of a black box function obeys linearity as any other quantum gate does. It is assumed that f is implemented via a quantum operator Q_f acting on $n+1$ qubits as $Q_f \,|\boldsymbol{x}\rangle \,|b\rangle = |\boldsymbol{x}\rangle \,|f(\boldsymbol{x}) \otimes b\rangle$.

Quantum algorithm for the Deutsch-Jozsa problem

1. Start with $n+1$ qubit state $|0\ldots0\rangle \,|1\rangle$
2. Apply Hadamard-Walsh transform W to all qubits to obtain state

$$\frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} |\boldsymbol{x}\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

3. Call function f (i.e., apply Q_f) on the above superposition to obtain

$$\begin{aligned} & \frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} |\boldsymbol{x}\rangle \frac{1}{\sqrt{2}} (|f(\boldsymbol{x})\rangle - |f(\boldsymbol{x}) \oplus 1\rangle \\ = & \frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} |\boldsymbol{x}\rangle (-1)^{f(\boldsymbol{x})} \frac{1}{\sqrt{2}} (|\mathbf{0}\rangle - |\mathbf{1}\rangle) \end{aligned}$$

4. Apply W to each qubit to obtain

$$\begin{aligned} & \frac{1}{2^n} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} (-1)^{f(\boldsymbol{x})} \sum_{\boldsymbol{y} \in \mathbb{Z}_2^n} (-1)^{\boldsymbol{x} \cdot \boldsymbol{y}} |\boldsymbol{y}\rangle |1\rangle \\ = & \frac{1}{2^n} \sum_{\boldsymbol{y} \in \mathbb{Z}_2^n} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} (-1)^{f(\boldsymbol{x})} (-1)^{\boldsymbol{x} \cdot \boldsymbol{y}} |\boldsymbol{y}\rangle |1\rangle \end{aligned}$$

5. Observe the first n qubits. The probability that $\boldsymbol{y} = \mathbf{0}$ will be observed is given by

$$\mathbb{P}(\mathbf{0}) = \left| \frac{1}{2^n} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} (-1)^{f(\boldsymbol{x})} \right|^2 = \begin{cases} 0, & \text{if } f \text{ is balanced} \\ 1, & \text{if } f \text{ is constant} \end{cases}$$

In terms of interference, the above quantum algorithm can be analyzed as follows: Step 1 begins at a deterministic state $|0 \ldots 01\rangle$, which is in step 2 divided into 2^{n+1} different computational paths, each having a configuration of form $|\boldsymbol{x}\rangle |0\rangle$ or $|\boldsymbol{x}\rangle |1\rangle$. All paths have amplitude $\frac{1}{\sqrt{2^{n+1}}}$.

Step 3 then does not split the computational paths further, but only affects on the amplitudes so that the value of $f(\boldsymbol{x})$ is encoded to the signs of the amplitudes.

Step 4 again splits each existing computational path into 2^{n+1} paths. As shown in step 5, the interference can be calculated exactly and the result is that in the case of constant function, the computational paths ending up to $|\mathbf{0}\rangle$ interfere constructively and the other ones destructively, so that in the constant case, it is only possible to observe $\mathbf{0}$ as the first n qubit values. On the other hand, in the balanced case, the situation is exactly the opposite, all paths ending to $|\mathbf{0}\rangle$ cancel each other perfectly and something different from $\mathbf{0}$ will be observed.

In [5] Deutsch considered a special case of the Deutsch-Jozsa problem with $n = 1$. In that case, each function $f : \mathbb{Z}_2 \to \{0, 1\}$ is indeed either constant or balanced, and that was most likely the first sound evidence that

quantum computing can provide some advantage over the classical computing.

As mentioned before, The Deutsch-Jozsa problem has an efficient classical solution, if a probabilistic solution is allowed. On the other hand, Daniel Simon [25] demonstrated in 1994 that there are problems (with promise and black-box) that allow significantly more efficient solutions on quantum computers, even if the stochastic nature of the output is allowed.

Simon's problem: Given a black box function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ with the promised property that there is some $\boldsymbol{s} \in \mathbb{Z}_2^n$ so that $f(\boldsymbol{x}) = f(\boldsymbol{y})$ if and only if $\boldsymbol{x} = \boldsymbol{y}$ or $\boldsymbol{y} = \boldsymbol{s} + \boldsymbol{x}$. The problem is to find out this particular element $\boldsymbol{s} \in \mathbb{Z}_2^n$.

The quantum solution of Simon's problem uses the quantum interference mostly in the same way as Deutsch-Jozsa solution does. Here we assume that function f is evaluated via a black-box quantum operator Q_f operating of $2n$ qubits: $Q_f \left|\boldsymbol{x}\right\rangle \left|\boldsymbol{0}\right\rangle = \left|\boldsymbol{x}\right\rangle \left|f(\boldsymbol{x})\right\rangle$. Also, we denote $\left|\boldsymbol{0}\right\rangle = \left|00\ldots0\right\rangle$.

Simon's algorithm

1. Start with $2n$ qubit state $\left|\boldsymbol{0}\right\rangle \left|\boldsymbol{0}\right\rangle$.
2. Apply $W^{\otimes n}$ on the first n qubits to obtain
$$\frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} \left|\boldsymbol{x}\right\rangle \left|\boldsymbol{0}\right\rangle.$$
3. Call function f (apply operation Q_f) to obtain
$$\frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} \left|\boldsymbol{x}\right\rangle \left|f(\boldsymbol{x})\right\rangle$$
4. Apply $W^{\otimes n}$ on the first n qubits to get
$$\frac{1}{2^n} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^n} \sum_{\boldsymbol{y} \in \mathbb{Z}_2^n} (-1)^{\boldsymbol{x} \cdot \boldsymbol{y}} \left|\boldsymbol{y}\right\rangle \left|f(\boldsymbol{x})\right\rangle$$
5. Observe all $2n$ qubits simultaneously. By the assumption, there will be two identical values $f(\boldsymbol{x}) = f(\boldsymbol{x} + \boldsymbol{s})$ and hence an interference

pattern

$$\begin{aligned} & (-1)^{\boldsymbol{x}\cdot\boldsymbol{y}} \,|\boldsymbol{y}\rangle\,|f(\boldsymbol{x})\rangle + (-1)^{(\boldsymbol{x}+\boldsymbol{s})\cdot\boldsymbol{y}} \,|\boldsymbol{y}\rangle\,|f(\boldsymbol{x}+\boldsymbol{s})\rangle \\ = \;& ((-1)^{\boldsymbol{x}\cdot\boldsymbol{y}} + (-1)^{(\boldsymbol{x}+\boldsymbol{s})\cdot\boldsymbol{y}})\,|\boldsymbol{y}\rangle\,|f(\boldsymbol{x})\rangle \\ = \;& (-1)^{\boldsymbol{x}\cdot\boldsymbol{y}}(1+(-1)^{\boldsymbol{s}\cdot\boldsymbol{y}})\,|\boldsymbol{y}\rangle\,|f(\boldsymbol{x})\rangle \\ = \;& \begin{cases} 0, & \text{if } \boldsymbol{s}\cdot\boldsymbol{y}=1 \\ 2(-1)^{\boldsymbol{x}\cdot\boldsymbol{y}}\,|\boldsymbol{y}\rangle\,|f(\boldsymbol{x})\rangle, & \text{if } \boldsymbol{s}\cdot\boldsymbol{y}=0 \end{cases} \end{aligned}$$

takes place. This means that in the first n qubits, we can observe only elements $\boldsymbol{y} \in \mathbb{Z}_2^n$ satisfying $\boldsymbol{y}\cdot\boldsymbol{s} = 0$.

6. Repeating the procedure over again, we will eventually obtain $n-1$ linearly independent elements $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n-1}$ all satisfying $\boldsymbol{y}_i \cdot \boldsymbol{s} = 0$. Using those elements, $\boldsymbol{s}$ can be recovered by using Gauss-Jordan procedure.

A more careful analysis of the above procedure shows that $O(n)$ applications of Q_f (queries) are enough to discover element $\boldsymbol{s}$ with a high probability. On the other hand, it can be shown that $\Omega(\sqrt{2^n})$ queries are needed for any classical black-box algorithm to solve the Simon's problem with a bounded error probability [24].

Whereas the Deutch-Jozsa problem separates zero-error classical and quantum computing, the Simon's problem introduces analogous separation for *bounded error* computing. Simon's problem actually shows that there is a (promise) problem efficiently solvable with quantum computers, whereas a classical efficient solution is impossible.

However, it should be noticed that we are again talking about the promise problems over black-box functions. As discussed before, the proofs using a black box model however do not imply results in unrelativized setting.

The question whether the QFT can be computed efficiently in large groups has at least two aspects: Can the group elements be efficiently represented in some quantum system as $|\,g_1\rangle, \ldots, |\,g_n\rangle$. Typically, we assume that this can be done (even though not yet technologically), but the second question about the Fourier transform is more important:

Assume that for a finite Abelian group G, there is an orthonormal basis $\{|\,g\rangle \mid g \in G\}$ and a function $f : G \to \mathbb{C}$ satisfying

$$\sum_{g\in G} |f(g)|^2 = 1.$$

Then

$$\sum_{g \in G} f(g) \, |g\rangle \tag{14}$$

is an eligible quantum state in an $|G|$-dimensional vector space. Is it possible to have a sequence of simple quantum operations transforming (14) into

$$\sum_{g \in G} \widehat{f}(g) \, |g\rangle \tag{15}$$

so, that the number of operations would be polynomial in the group descriptions size?

The answer is known in some cases. As we saw before, $\mathbb{Z}_2^n = \mathbb{Z}_2 \times \ldots \times \mathbb{Z}_2$, and QFT on $\mathbb{Z}_2^n$ can be performed by performing it on each two-element subgroup $\mathbb{Z}_2$, and this can be generalized to any product representation: If $G = G_1 \times G_2$, the Quantum Fourier transform on G can be obtained by computing it on G_1 and G_2 separately.

Moreover, the case when a cyclic group C_n has cardinality $n = n_1 n_2$, where $\gcd(n_1, n_2) = 1$ can be handled: It is known how to implement the QFT in C_n if it can be implemented in C_{n_1} and C_{n_2} [13].

Group $\mathbb{Z}_{2^n}$ plays an important role in Shor's factoring algorithm. In fact, the "quantum part" of Shor's factoring algorithm consists of finding out the smallest period of function $n \mapsto a^n \pmod N$, and this problem could be actually most naturally be described as a *Hidden subgroup problem* in $\mathbb{Z}$. On the other hand, $\mathbb{Z}$ is an infinite group, so a computationally feasible solution must be searched from elsewhere, and it turns out that $\mathbb{Z}_{2^n}$ as a finite approximation of $\mathbb{Z}$ will do.

Any element of $\mathbb{Z}_{2^n}$ can be presented as a string of n (qu)bits, but notice the difference: $\mathbb{Z}_{2^n}$ is a cyclic group, whereas $\mathbb{Z}_2^n$ is not if $n > 1$. Moreover, $\mathbb{Z}_{2^n}$ has no cyclic subgroups with coprime cardinalities, so something else different from the aforesaid recursive procedures must, and has been discovered. It is known how to implement the QFT in $\mathbb{Z}_{2^n}$ with $O(n^2)$ quantum operations [13].

5 Simulating quantum interference classically

Recall again that Richard Feynman proposed that it may be computationally expensive to simulate quantum mechanical system classically, the reason being that a system of n particles has a mathematical description size exponential in n.

In this section, we will study the simulation problem of quantum mechanics in terms of the classical complexity theory. In principle, Quantum Turing machine will be the underlying model, but in the case of polynomial time computing, we can in practice restrict to $N = n^k$ qubits ($N = n^k$ is the absolute upper bound on the qubits needed for the computation). In the continuation, we will use these notations and restrictions.

We can hence view the quantum computing as a process where the previous configuration (vector in H_{2^N}) is multiplied with the unitary computational operator U until the final configuration is reached at step t.

Theorem 1. BQP $\subseteq$ EXPSPACE

Proof sketch. Let us assume that a Quantum Turing machine Q takes a binary input word $w \in \{0,1\}^n$ and the **BQP** computation of Q lasts for $T = n^k$ steps. Clearly Q uses no more than $N = n^k$ qubits. It follows that the mathematical representation of the computation can be accommodated in a 2^{n^k}-dimensional Hilbert space, which means that each configuration $|\boldsymbol{x}\rangle$ can be presented as a sequence of 2^{n^k} complex numbers.

Simulation of a computational step uses the current state vector, and for each nonzero amplitude α corresponding to configuration C, finds out which are the (finitely many) successors of C. If, say $C \xrightarrow{\alpha_i} C_i$, then the new amplitude for C_i is $\alpha\alpha_i$, but it should be noted that there may be more than one configuration yielding C_i as a successor. If so, all equal C_is are combined and their amplitudes are summed (this is interference!).

Another point of view of the aforesaid is that each configuration of Q can be represented as an infinite-dimensional row vector $|\boldsymbol{x}\rangle$ (having only at most 2^{n^k} nonzero coordinates), which is multiplied from the left by the transition matrix of U of Q (see [2] for more details), which has only finitely many nonzero entries on each row and column.

Independent from the viewpoint, it is straightforward to verify that the (classical) simulation can be performed using $O(2^{n^k})$ space. □

Remark 3. In the above proof sketch, the bounded error property of **BQP** was not used at all, but the polynomial time length was essential. An analogous conclusion hence holds for any polynomial time quantum computing.

Remark 4. In the above sketch, the space needed to store the complex numbers representing the amplitudes was not considered at all. Neither was the space or time needed to compute the algebraic operations. If fact, it was not even specified which kind of amplitudes are permissible for a Quantum Turing machine Q. All these questions are studied in details in [2], hence

we will not pay very much attention to those questions here, but will rather present also the forthcoming proof sketches ignoring the technical details surrounding the complex numbers.

Theorem 2. BQP $\subseteq$ PSPACE

Proof sketch. We will here use the same notations as in the previous proof. Keeping the whole state of n^k qubits in the classical computer memory requires by definition 2^{n^k} complex numbers to remember. On the other hand, the simulation of a single computational path (of length at most n^k) can be performed easily, just by multiplying the amplitudes along the path, starting from the root (representing the initial configuration) and ending at a leaf (corresponding to a final configuration). Thus, it is possible to compute any leaf amplitude in polynomial space, and even in polynomial time.

Even though computing the final amplitude takes only a polynomial time (and hence space) for any final configuration, it is nevertheless true that there may be exponentially many computational paths, some leading to accepting and some to rejecting final configuration. For the acceptance probability, we should add the amplitudes with equal final configurations (this is interference!) and eventually discover the sum of squared absolute values of the accepting configurations.

Now the standard strategy for computing the total acceptance probability is to reuse space for computing each final amplitude for each computational path. A more elaborate analysis will demonstrate that this is indeed enough: computing the total acceptance probability is possible to do in **PSPACE** by always reusing the space. □

A somewhat more precise result can be obtained by using the strategy of [2] or [1].

Theorem 3. BQP $\subseteq$ PP

Proof sketch. Without loss of generality, we may assume that there is only one accepting final configuration, which can be achieved via paths P_1, ..., P_r with amplitudes α_1, ..., α_r. In fact, the proof would be essentially the same for multiple final accepting configurations, only the technical details would be more complicated.

Now the acceptance probability is given by

$$\Big|\sum_{i=1}^{r}\alpha_i\Big|^2 = \Big(\sum_{i=1}^{r}\alpha_i\Big)\Big(\sum_{j=1}^{r}\alpha_j^*\Big) = \sum_{i=1}^{r}\sum_{j=1}^{r}\alpha_i\alpha_j^*$$

$$= \sum_{i=1}^{r}|\alpha_i|^2 + \sum_{i>j} 2\,\mathrm{Re}(\alpha_i\alpha_j^*),$$

a sum where each α_i can be computed in polynomial time. In order to convert the evaluation of that sum into an **PP** algorithm M, we just need to let M guess, instead of a single guess in **BQP** machine, two computational paths, one leading to amplitude α_i and the other one to α_j in the above formula.

After guessing the paths, compute α_i and α_j (can be done in polynomial time). Then compute $|\alpha_i|^2$, if $i = j$, and $2\,\mathrm{Re}(\alpha_i\alpha_j^*)$ if $i \neq j$, and proceed as follows: if $i = j$, create roughly $C \cdot \alpha_i^2$ accepting successor paths. If $i \neq j$ but $2\,\mathrm{Re}(\alpha_i\alpha_j^*) > 0$, create roughly $C \cdot 2\,\mathrm{Re}(\alpha_i\alpha_j^*)$ accepting paths, but if $2\,\mathrm{Re}(\alpha_i\alpha_j^*) < 0$, create about $C \cdot 2\,\mathrm{Re}(\alpha_i\alpha_j^*)$ rejecting paths.

Here C is a constant depending on the precision of the numbers occurring in the definition of a **PP** machine, the number of the paths must be certainly an integer. For a detailed proof, see [2]. □

Remark 5. The latest theorem shows that the polynomial time quantum computing in **BQP** can be simulated in polynomial time classical model **PP**, but as noted previously, **PP** cannot be regarded as "practical computing". Especially, the bounded error -property of **BQP** is usually lost in the simulation.

Remark 6. Scott Aaronson has pointed out [1] that the **PP** simulation of **BQP** following the computational paths is analogous to the Feynman path integrals. The analogue is certainly evident, but the details are not so straightforward.

6 Finite automata

Finite automata, especially their probabilistic versions are the main subject of this final chapter. As a general information source on regular languages and finite automata, we refer to Eilenberg [7] and Yu [28]. For probabilistic generalization of the finite automata, Rabin [22], Turakainen [26] and Paz [20] are recommended, and their quantum versions can be studied from Hirvensalo [14], [15], or Ambainis and Yakaryılmaz [3].

Finite automata are very natural mathematical tools for presenting space-bounded computation. However, in my opinion, the beauty of the automata theory lies in its elegant mathematical properties: closure under Boolean operations and Kleene star, multiple alternative viewpoints such as rational formal power series, finite syntactic monoids (recognizability), etc., see Yu[28].

As a natural generalization (and a variant) of deterministic (and nondeterministic) finite automata, Rabin introduced in 1963 probabilistic finite automata and stochastic languages Rabin [22]. Because we are anyway going to extend the notion of stochastic automata, it is useful here to start from the general case and then introduce the special one.

Definition 5. A *Generalized finite automaton* over $\mathbb{R}$, also known as $\mathbb{R}$-*weighted finite automaton* with n states over an alphabet Σ is a triplet $(\boldsymbol{x}, \{M_a \mid a \in \Sigma\}, \boldsymbol{y})$, where $\boldsymbol{x} \in \mathbb{R}^n$ is the *initial (column) vector*, each $M_a \in \mathbb{R}^{n \times n}$ is an $n \times n$ *transition matrix*, and $\boldsymbol{y}$ is the *final (row) vector* in $\mathbb{R}^n$.

To shorten the notations in the following definitions, we introduce some technical notions meanwhile.

Definition 6. The *reverse (or mirror image)* of a word $w = a_1 a_2 \dots a_n$ is defined as $w^R = a_n \dots a_2 a_1$.

Definition 7. If matrices A_i are indexed by letters a_i, then for any word $w = a_1 a_2 \dots a_n$, matrix A_w is defined as $A_w = A_{a_1} A_{a_2} \dots A_{a_n}$. In other words, mapping $w \to A_w$ is a morphism from the semigroup of the words into the semigroup of matrices. This morphism is extended to the monoid of words by assigning the identity matrix to the empty word.

Definition 8. The *value* $f_A(w)$ computed by a generalized automaton A on input word $w = w_1 \dots w_n$ is given by

$$f_A(w) = \boldsymbol{y}^T M_{w^R} \boldsymbol{x}. \tag{16}$$

On our way to stochastic automata, we need to introduce some more definitions and brief lemmata.

Definition 9. A *stochastic vector* is a column vector with nonnegative coordinates which sum up to 1.

Definition 10. A matrix A is a stochastic, if all its columns are stochastic vectors.

The following lemma is straightforward, but essential in depicting the mathematical form of stochastic computing.

Lemma 3. *If $\boldsymbol{x}$ is a stochastic vector and A and B stochastic matrices, then $A\boldsymbol{x}$ is a stochastic vector and AB is a stochastic matrix.*

Definition 11. A *stochastic* (also called *probabilistic*) finite automaton $P = (\boldsymbol{y}, \{M_a \mid a \in \Sigma\}, \boldsymbol{x})$ with n states over an alphabet Σ is a generalized finite automaton where each transition matrix, the initial vector $\boldsymbol{x}$ is stochastic, and the final vector $\boldsymbol{y}$ is in $\{0,1\}^n$.

The following lemma is an obvious consequence of the definitions:

Lemma 4. *Let P be a stochastic finite automaton as above. Then its value function satisfies*

$$f_P(w) = \boldsymbol{y}^T M_{w^R} \boldsymbol{x} \in [0,1] \tag{17}$$

for each $w \in \Sigma^*$.

Remark 7. According to the previous lemma, a stochastic (probabilistic) automaton assigns to each input word w a number $f_P(w) \in [0,1]$. This number can be interpreted as a probability to accept the input word, and this interpretation gives raise to the following definition.

Definition 12. Let P be a stochastic automaton and $\lambda \in [0,1]$. We will call λ as the cut-point, and the stochastic *cut-point language* is defined as

$$L_{>\lambda}(P) = \{w \in \Sigma^* \mid f_P(w) > \lambda\}. \tag{18}$$

The set of stochastic cut-point languages will be denoted as $\mathbf{S}^{>}$ hereafter and called stochastic languages.

Definition 13 (Stochastic and generalized languages)**.** If symbol "$>$" in Equation (18) of Definition 12 is replaced with $<$, $=$, $\geq$, etc., we then denote the corresponding language classes by $\mathbf{S}^{<}$, $\mathbf{S}^{=}$, $\mathbf{S}^{\geq}$, etc. If the automaton model is relaxed to allow a generalized automaton, we denote the corresponding language classes as $\mathbf{G}^{<}$, $\mathbf{G}^{=}$, $\mathbf{G}^{\geq}$, etc.

When restricting to rational entries (matrices, initial and final vectors, and the cut-point), we will use subindex $\mathbb{Q}$ and write $\mathbf{S}^{>}_{\mathbb{Q}}$, $\mathbf{G}^{>}_{\mathbb{Q}}$, etc.

As mentioned before, the study of stochastic languages started roughly 1963, when Rabin published his article [22]. Using a cardinality argument, Rabin demonstrated that there must be stochastic languages which are not rational. Rabin also noticed an important property of the *bounded error* automata formally defined below:

Definition 14. Let P and $L_{>\lambda}(P)$ be as in Definition 12. The cut-point λ is called *isolated*, if there exists an $\epsilon > 0$ so that for each $w \in \Sigma^*$ either $f_P(w) \geq \lambda + \epsilon$ or $f_P(w) \leq \lambda - \epsilon$. Such an automaton P is called a *bounded error* stochastic automaton, and languages defined by such automata are called bounded-error stochastic languages.

Remark 8. The bounded error property for automata is analogous to **BPP** property of the Turing Machines: If the acceptance probability can be arbitrarily close to λ, then it may require exponentially many attempts to practically separate cases $w \in L$ and $w \notin L$. But whereas we do not know whether the inclusion $\mathbf{P} \subseteq \mathbf{BPP}$ is strict (it is supposed not to be, though), we know that the bounded-error probabilistic automata do not bring any additional computing power: Quite remarkably, Rabin demonstrated that any *bounded-error* stochastic language is regular [22].

Since Rabin's seminal article, it seems that the next few years brought quite little progress in the theory of stochastic languages, but this may be understood via the Rabin's aforesaid result forcing all bounded-error stochastic languages regular. Even though Rabin demonstrated that nonregular stochastic languages exists, no explicit example was given.

After Rabin's result, there were some progress on probabilistic automata, but the next steps relevant to this study were achieved only in the end of 1960's when Paavo Turakainen presented his works [26] and [27], where an explicit construction of a nonregular stochastic language was presented.

Theorem 4 (Turakainen [27])**.** *Language* $\{a^n b^n \mid n \in \mathbb{N}\}$ *is stochastic.*

Notice that the pumping lemma [28] implies that the language $\{a^n b^n \mid n \in \mathbb{N}\}$ cannot be regular, and by Rabin's result, it is clear that this language cannot be a bounded-error stochastic, either.

The rest of this section is dedicated to Turakainen's method to prove the above claim. It will be argued that in the final step, Turakainen used the interference of computational paths, but also that prior to that, he presented a general method for simulating interference in general automata using stochastic automata. As we shall see, the price of such simulation is analogous to **PP** simulation of **BQP**: The bounded-error property of the cutpoint will be lost.

Remark 9. In 1981, Rūsiņš Freivalds demonstrated that the language $\{a^n b^n \mid n \in \mathbb{N}\}$ can be recognized by a probabilistic *two-way finite automaton* with a bounded-error probability [10]. Freivald's construction was

completely different from that of Turakainen, and requires exponential running time. Even though Freivald's construction is very interesting, it does not make use of interference, and hence is not included in this tutorial.

Hereafter I will argue that this construction by Turakainen, showing that $\{a^n b^n \mid n \in \mathbb{N}\}$ is stochastic, was the earliest example of using interference in computational processes. To be honest, I will only argue that Turakainen's proof *uses* interference, and if a reader recognizes any earlier usage of interference in algorithm construction, I warmly invite him/her to let me know of such occasion.

As the interference requires both positive and negative "amplitudes" which don't occur in probabilistic automata, the model must be first extended to allow also negative numbers, this is what we called *generalized automata* in the beginning of this section, likewise Turakainen did in [26].

The first steps towards a general result are very simple.

Lemma 5. *If f_A and f_B are functions $\Sigma^* \to \mathbb{R}$ computed by generalized automata A and B, then so are $f_A f_B$ and $f_A + f_B$.*

Proof sketch. The tensor product construction ensures that function $f_A f_B$ can be computed by a generalized automaton. Similarly, the direct sum construction ensures that $f_A + f_B$ can be computed by a generalized automaton, Paz [20], Hirvensalo [15]. □

Lemma 6. *Let G be a generalized automaton and $\lambda \in \mathbb{R}$. Then it is possible to construct a generalized automaton G' so that $L_{>\lambda}(G) = L_{>0}(G')$.*

Proof. As the constant function $\Sigma^* \to \mathbb{R}$, $w \mapsto -\lambda$ can be easily computed by a one-state generalized automaton, the claim follows directly from the previous lemma. □

Lemma 7. *For any generalized automaton G, it is possible to construct another generalized automaton G' which has its initial vector of form $(1, 0, \ldots, 0)$, its final vector of form $(0, 0, \ldots, 1)$ and $f_G(w) = f_{G'}(w)$.*

Proof sketch. If $\boldsymbol{x}$ and $\boldsymbol{y}$ are the initial and final vectors of G, then replacing each matrix A of G with

$$\begin{pmatrix} 0 & 0 & 0 \\ A\boldsymbol{x} & A & 0 \\ \boldsymbol{y}^T A\boldsymbol{x} & \boldsymbol{y}^T A & 0 \end{pmatrix}$$

will do. □

Since each probabilistic automaton is a special case of $\mathbb{R}$-weighted automaton, it is clear that $\mathbf{S}^{>} \subseteq \mathbf{G}^{>}$. Turakainen's great contribution was to show that also the contrary is true.

Theorem 5 (Turakainen [26])**.**

$$\mathbf{G}^{>} = \mathbf{S}^{>}.$$

Proof. Let L be a cut-point language of an n-state generalized automaton $G = (\boldsymbol{x}, \{A_i \mid i \in \Sigma\}, \boldsymbol{y})$. Due to the previous lemmata, we can assume that the cut-point equals to 0, and that the initial and final vectors are chosen as $\boldsymbol{x} = (1, 0, \ldots, 0)$, and $\boldsymbol{y} = (0, \ldots, 1)$. It remains to be shown that there is a stochastic automaton S and $\lambda \in [0, 1]$ so that $L = L_{>\lambda}(S)$.

Step 1. *Annihilating row and column sums.* Let B_i be an $(n+2) \times (n+2)$-matrix defined by augmenting A_i at each size by one row or column:

$$B_i = \begin{pmatrix} 0 & 0 & 0 \\ \boldsymbol{b}_i & A_i & 0 \\ a_i & \boldsymbol{c}_i & 0 \end{pmatrix},$$

where $\boldsymbol{c}_i \in \mathbb{R}^n$, $\boldsymbol{b}_i^T \in \mathbb{R}^n$, and $a_i \in \mathbb{R}$ are chosen so that the row and column sums of B_i are zero. Clearly $\boldsymbol{b}_i$, $\boldsymbol{c}_i$, a_i are determined uniquely. It is also straightforward to see that for any $w \in \Sigma^+$,

$$B_w = \begin{pmatrix} 0 & 0 & 0 \\ \boldsymbol{b}_w & A_w & 0 \\ a_w & \boldsymbol{c}_w & 0 \end{pmatrix},$$

where $\boldsymbol{b}_w$, $\boldsymbol{c}_w$, and a_w make the row and column sums of B_w zero. Introducing two extra rows and columns here correspond2 to adding two states to the automaton.

More importantly, letting $\boldsymbol{x}_1 = (0, \boldsymbol{x}, 0) \in \mathbb{R}^{n+2}$ and $\boldsymbol{y}_1 = (0, \boldsymbol{y}, 0) \in \mathbb{R}^{n+2}$ we can see that

$$\boldsymbol{y}_1^T B_w \boldsymbol{x}_1 = \boldsymbol{y}^T A_w \boldsymbol{x}.$$

By the assumption, both $\boldsymbol{x_1}$ and $\boldsymbol{y}_1$ are of special type: Exactly one coordinate equals to 1, the other ones are 0.

Step 2. *Forcing entries positive.* Let $\mathbb{E}$ be an $(n+2) \times (n+2)$ matrix having all elements equal to 1, Choose $c > 0$ large enough so that each element of each

$$C_i = B_i + c\mathbb{E},$$

is nonnegative. Now that the columns and rows of each B_i sum up to 0, it follows that $B_i\mathbb{E} = \mathbb{E}B_i = \mathbf{0}$, where $\mathbf{0}$ stands for the zero matrix. As also $\mathbb{E}^k = (n+2)^{k-1}\mathbb{E}$, it follows that

$$C_w = B_w + c^{|w|}(n+2)^{|w|-1}\mathbb{E}.$$

As the row and the column sums of each B_i is zero, and those of $c\mathbb{E}$ is $c(n+2)$, it follows that the row and column sums of each C_i is $c(n+2)$.

Step 3. *Renormalization.* Let

$$D_i = \frac{1}{c(n+2)}C_i.$$

It follows immediately that each entry of D_i is nonnegative, and that the row and column sums of each D_i equals to 1, which is another way to say that each D_i is *doubly stochastic*.

An easy calculation then shows that

$$D_w = \frac{1}{(c(n+2))^{|w|}}B_w + \frac{1}{n+2}\mathbb{E},$$

and it follows that

$$\begin{aligned} \boldsymbol{y}_1^T D_w \boldsymbol{x}_1 &= \frac{1}{(c(n+2))^{|w|}}\boldsymbol{y}_1^T B_w \boldsymbol{x}_1 + \frac{1}{n+2}\boldsymbol{y}_1^T \mathbb{E}\boldsymbol{x}_1 \\ &= \frac{1}{(c(n+2))^{|w|}}\boldsymbol{y}^T A_w \boldsymbol{x} + \frac{1}{n+2}, \end{aligned} \tag{19}$$

so $\boldsymbol{y}_1^T D_w \boldsymbol{x}_1 > \frac{1}{n+2} \Leftrightarrow \boldsymbol{y}^T A_w \boldsymbol{x} > 0$. The claim follows now by choosing $S = (\boldsymbol{x}_1, \{D_i \mid i \in \Sigma\}, \boldsymbol{y}_1)$ and $\lambda = \frac{1}{n+2}$. □

Remark 10. The above construction by Turakainen maps a set $\{A_i \mid i \in \Sigma\}$ of arbitrary matrices into a set $\{D_i \mid i \in \Sigma\}$ of stochastic matrices. Without any constraints, such a mapping would be straightforward to find, but Turakainen's construction ingeniously preserves multiplicative structure of the original set well enough. On the other hand, as the first term of Equation (19) tends to zero when $|w| \to \infty$, it is clear that the boundedness property of the threshold (if originally existed) is not preserved.

Nevertheless, this construction simulates interference (if such exists in the original generalized automaton) by using a stochastic automaton and is analogous to proving that **BQP** $\subseteq$ **PP**.

In order to prove Theorem 4, it will be useful to present another auxiliary result.

Lemma 8 (Turakainen [27])**.** $\mathbf{G}^{=}_{\mathbb{Q}} \subseteq \mathbf{G}^{>}_{\mathbb{Q}} \subseteq \mathbf{G}^{>} = \mathbf{S}^{>}$.

Proof sketch. The last equality is that of Theorem 5, and the second last inclusion is trivial. For the first inclusion, let $L \in \mathbf{G}^{=}_{\mathbb{Q}}$ and let also $A = (\boldsymbol{x}, \{A_i \mid i \in \Sigma\}, \boldsymbol{y})$ a generalized automaton with rational matrix entries so that

$$L = \{w \in \Sigma^* \mid \boldsymbol{y}^T A_w \boldsymbol{x} = \lambda\}.$$

Using the construction of Lemma 5, it is possible to obtain a generalized rational-entry automaton B so that

$$L = \{w \in \Sigma^* \mid \boldsymbol{y}^T B_w \boldsymbol{x} = 0\}.$$

Now choose M as the least common multiple of all denominators of the numbers in the description of automaton B to obtain a generalized integer-entry automaton C so that

$$L = \{w \in \Sigma^* \mid \boldsymbol{y}^T C_w \boldsymbol{x} = 0\}.$$

As now $\boldsymbol{y}^T C_w \boldsymbol{x} \in \mathbb{Z}$, we can also write

$$L = \{w \in \Sigma^* \mid (\boldsymbol{y}^T C_w \boldsymbol{x})^2 < 1\},$$

and by Lemma 5 there is a generalized automaton $D = (\boldsymbol{x}_1, \{D_i \in \Sigma\}, \boldsymbol{y}_1)$ so that $(\boldsymbol{y}^T C_w \boldsymbol{x})^2 = \boldsymbol{y}_1^T D_w \boldsymbol{x}_1$. It follows that

$$L = \{w \in \Sigma^* \mid \boldsymbol{y}_1^T D_w (-\boldsymbol{x}_1) > -1\},$$

which proves that $L \in \mathbf{G}^{>}_{\mathbb{Q}}$. □

The above lemmata show that in order to prove Turakainen's theorem (Theorem 4), it is sufficient to show that the language $L = \{a^n b^n \mid n \in \mathbb{N}\}$ (or a closely enough related language) is in $\mathbf{G}^{=}_{\mathbb{Q}}$.

For that purpose, Turakainen presented the automaton in Figure 6. The automaton in Figure 6 consists of weighted versions of two copies of a three-state automaton recognizing regular language a^+b^+ plus one sink state s. Each copy contains one initial state (p_1 and q_1, both with weight $\frac{1}{2}$) and one final state (p_3 and q_3, with weights 2 and -2, respectively).

Because of the automaton structure, it is clear that only words of form a^+b^+ can survive from an initial state to a final state, so let's consider an input word $w = a^{n+1}b^{k+1}$. The weight contribution of the upper copy is then $\frac{1}{2}(\frac{1}{2})^{n+1}(\frac{1}{4})^k(\frac{3}{4}) \cdot 2$, and the contribution of the lower copy is $\frac{1}{2}(\frac{1}{4})^n \cdot \frac{3}{4} \cdot (\frac{1}{2})^{k+1}(-2)$.

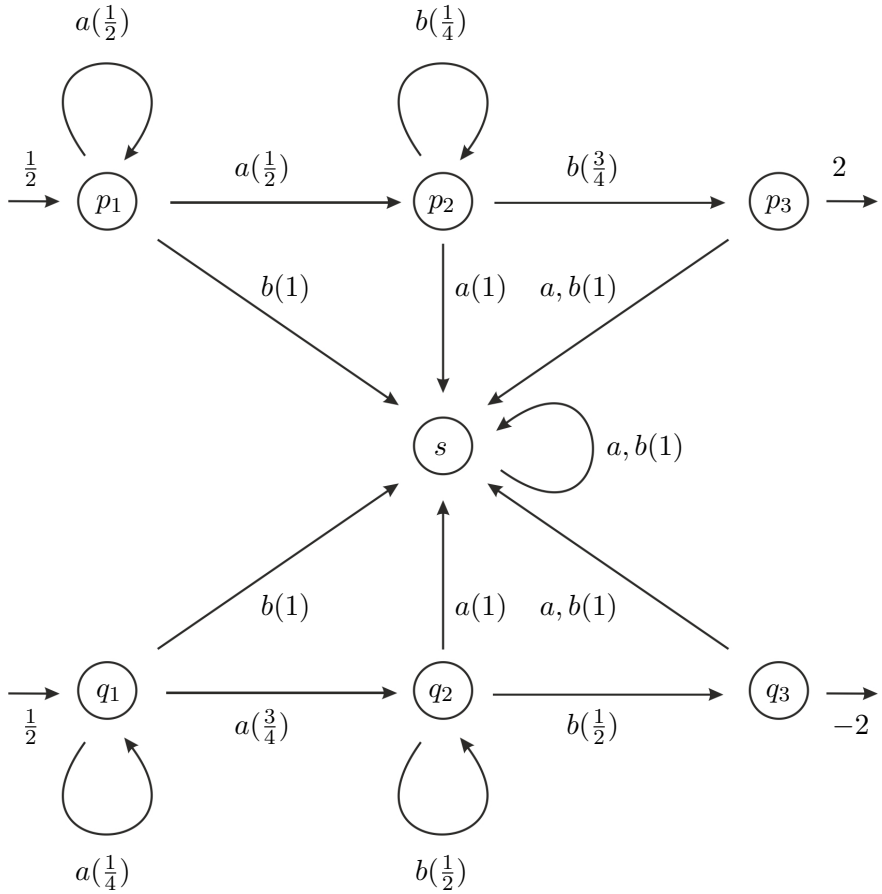

Figure 6: Turakainen's $\mathbf{G}^{=}_{\mathbb{Q}}$ automaton for $L = \{a^n b^n \mid n \geq 1\} \cup (\{a, b\}^* \setminus a^+ b^+)$ consists of two weighted versions of automaton recognizing language $a^+ b^+$. The weights of the transitions are in the brackets, the initial (resp. final) states are indicated by incoming (resp. outcoming arrows). Their weights are directly associated with the arrows.

As the final weights are of different signs, the weights of different computational paths can cancel each other: This is exactly where destructive interference occurs in automaton 6. Quite likely a reader used to quantum computing models and seeking the best analogy would probably had used initial weights $\frac{1}{2}$ and $-\frac{1}{2}$, and final weights both equal to 1, but the effect is the same anyway. Now, adding up the contributions of the upper and lower rows, we can conclude that the total weight associated to the word $a^{n+1} b^{k+1}$

is

$$\begin{aligned} & \frac{1}{2}\Big(\frac{1}{2}\Big)^{n+1}\Big(\frac{1}{4}\Big)^{k}\Big(\frac{3}{4}\Big)\cdot 2 + \frac{1}{2}\Big(\frac{1}{4}\Big)^{n}\cdot\frac{3}{4}\cdot\Big(\frac{1}{2}\Big)^{k+1}(-2) \\ = \;& \frac{3}{4}\Big(\frac{1}{2}\Big)^{n+k+1}\Big(\Big(\frac{1}{2}\Big)^{k}-\Big(\frac{1}{2}\Big)^{n}\Big). \end{aligned}$$

It is hence clear that the language of the $\mathbf{G}^{=}_{\mathbb{Q}}$-automaton of Figure 6 with $\lambda = 0$ is exactly $\{a^n b^n \mid n \geq 1\} \cup \big(\{a,b\}^* \setminus a^+b^+\big)$, meaning that the accepted words are those of form $a^n b^n$ with $n \geq 1$, and those that are not of form a^+b^+ (they fall into the sink state, hence their total weight will be 0 as well). The crucial point here is that inside the language a^+b^+, only words of form $a^n b^n$ have the property that their computational paths cancel each other. To put it in another way: A finite automaton has limited memory for computing, but in the generalized automaton of Figure 6, the interference "handles" the counting.

As mentioned above, the language L of the automaton in Figure 6 is not exactly the desired $\{a^n b^n \mid n \in \mathbb{N}\}$, but $\{a^n b^n \mid n \in \mathbb{N}\} \cup L_1$, where $L_1 = \{a,b\}^* \setminus a^+b^+$ consists of words not of form a^+b^+. However, clearly

$$\{a^n b^n \mid n \in \mathbb{N}\} = L \cap L_1,$$

and because of the aforesaid constructions, we know that $L \in \mathbf{G}^{=}_{\mathbb{Q}} \subseteq \mathbf{S}^{>}_{\mathbb{Q}}$.

The proof of Theorem 4 is now almost complete, and the very last simple lemma will conclude it:

Lemma 9. *The intersection of stochastic language $L_1 \in \mathbf{S}^{>}$ and regular language L_2 is stochastic.*

Proof. Let A be a stochastic automaton so that $L_1 = L_{>\lambda}(A) = \{w \mid f_A(w) > \lambda\}$ and B a deterministic automaton (a subcase of stochastic automaton). Now the function computed by B satisfies $f_B(w) \in \{0,1\}$, and by Lemma 5, there is a generalized automaton C so that $f_C(w) = f_A(w) f_B(w)$. □

References

[1] Scott Aaronson: *Quantum Computing Since Democritus*. Cambridge University Press (2013).

[2] Leonard M. Adleman, Jonathan Demarrais, and Ming-Deh A. Huang: *Quantum Computability*. SIAM Journal on Computing 26:5, pp. 1524–1540 (1997).

[3] Andris Ambainis and Abuzer Yakaryılmaz: *Automata and Quantum Computing*, a manuscript. `https://arxiv.org/abs/1507.01988`, read 03 Jul 2017.

[4] Paul Benioff: *The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines*. Journal of Statistical Physics 22:5, pp. 563–591 (1980).

[5] David Deutsch: *Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer*. Proceedings of the Royal Society of London A. 400: 97–117 (1985).

[6] David Deutsch and Richard Jozsa: *Rapid solutions of problems by quantum computation*. Proceedings of the Royal Society of London A. 439: 553–558 (1992).

[7] Samuel Eilenberg: *Automata, Languages, and Machines*. Academic Press (1974).

[8] Richard P. Feynman: *Simulating physics with computers*. International Journal of Theoretical Physics 21, 467–488 (1982).

[9] Richard P. Feynman: *Negative probability*. In Basil J. Hiley and D. Peat (eds.): *Quantum Implications: Essays in Honour of David Bohm*, pp. 235–248 (1987).

[10] Rūsiņš Freivalds: *Probabilistic Two-Way Machines*. Proceedings on Mathematical Foundations of Computer Science, LNCS 188, pp. 33–45 (1981).

[11] John Gill *Computational complexity of probabilistic Turing machines*. SIAM Journal on Computing 6 (4), pp. 675695 (1977).

[12] Godfrey H. Hardy, Edward M. Wright: *An introduction to the theory of numbers*, 5th ed. (1979).

[13] Mika Hirvensalo: *Quantum Computing*, 2nd edition. Springer (2004).

[14] Mika Hirvensalo: *Various Aspects of Finite Quantum Automata*. LNCS 5257 (Proceedings of DLT 2008), pp. 21–33 (2008).

[15] Mika Hirvensalo: *Quantum Automata with Open Time Evolution. International Journal of Natural Computing Research* 1, 70–85 (2010).

[16] Mika Hirvensalo: *Mathematics for Quantum Information Processing*. In G. Rozenberg, T. Bck, J. Kok (eds.): *Handbook of Natural Computing*. Springer (2012).

[17] Yuri Manin: *Computable and Uncomputable* (in Russian). Sovetskoye Radio, Moscow (1980).

[18] John von Neumann: Mathematische Grundlagen der Quantenmechanik. Springer (1932).

[19] Christos H. Papadimitriou: *Computational Complexity*. Pearson (1994).

[20] Azaria Paz: *Introduction to Probabilistic Automata*. Academic Press (1971).

[21] Max Planck: Annalen der Physik 1, 69, 1900; Verhandlg. dtsch. phys. Ges., 2, 202; Verhandlg. dtsch. phys. Ges. 2, 237; Annalen der Physik 4, 553, 1901.

[22] Michael O. Rabin: *Probabilistic automata*. Information and Control 6, pp. 230–245 (1963).

[23] Peter W. Shor: *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*. Proceedings of the 35th Annual Symposium on Foundations of Computer Science 20–22, IEEE Computer Society Press, pp. 124-134 (1994).

[24] Victor Shoup: *Lower Bounds for Discrete Logarithms and Related Problems*. Lecture Notes in Computer Science 1233, pp. 256–266 (1997).

[25] Daniel Simon: *On the Power of Quantum Computation*. Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, pp. 116–123 (1994).

[26] Paavo Turakainen: *On Probabilistic Automata and their Generalizations*. Annales Academiae Scientiarum Fennicae. Series A 429 (1969).

[27] Paavo Turakainen: *On languages representable in rational probabilistic automata*. Annales Academiae Scientiarum Fennicae. Series A 439 (1969).

[28] Sheng Yu: *Regular Languages*. In: G. Rozenberg and A. Salomaa: *Handbook of Formal Languages*, vol 1, Springer (1997).

[29] Thomas Young: *The Bakerian Lecture: On the Theory of Light and Colours*. Philosophical Transactions of the Royal Society of London. 92: 12-48 (1802).