Computational equivalence between quantum Turing machines and quantum circuit families

Christian Westergaard

21st November 2005

Contents

Preface

The following paper is my master's thesis in mathematics at the University of Copenhagen. The intention with my master's study was to obtain a knowledge about the theoretical foundation of quantum computing.

The field of quantum computation has developed very rapidly in the last decade. It seems safe to say, that the interest for this subject is do to some of its promising applications, e.g., Shor's prime-factoring algorithm and Grover's search algorithm. This focus, on applying the theory, is also evident in the literature which most often concentrates on the circuit model of quantum computation. This model is the most suited for describing computations and developing algorithms. But, when studying the foundational aspects of quantum computation, the Turing machine model is better qualified. Presently there are several monographs available on quantum computing, but most of these avoid the quantum Turing machine. It should be said though, that [Gruska, 1999] does have a chapter on quantum Turing machines and does treat some of the more foundational aspects of the theory. But since its publication several essential developments regarding the quantum Turing machine model have been made.

I have chosen to concentrate my study on some results obtained by Harumichi Nishimura and Masanao Ozawa regarding the computational equivalence between quantum Turing machines and quantum circuit families. The two authors have kindly supplied me with a draft of their paper [Nishimura and Ozawa, 2005a] before its publication[1]. My master's thesis is essentially a presentation of the main result in this paper.

Prerequisites

The reader is assumed to have a basic knowledge of classical computation and complexity theory. In particular one should be familiar with the polynomial-time complexity classes **P**, **ZPP**, and **BPP**, i.e., classes of languages recognized by polynomial-time Turing machines implementing exact, zero-error, and bounded-error algorithms, respectively. We will also use the concept of an encoding $\langle M \rangle$ of a Turing machine M. Furthermore one should know $\mathcal{O}$-notation. As a reference for classical complexity theory I suggest [Papadimitrio, 1994].

[1] Available since the 12th November 2005 from arXiv.org.

Introduction

At the foundation of computation theory lies the Turing machine model. In a simple way it defines the notion of algorithm, computation time, space, etc. However other models of computation, like the circuit model, are more suitable for designing algorithms and for physical realization. In order to choose freely between the two models one must establish the computational equivalence between them.

Shortly after Ethan Bernstein and Umesh Vazirani had presented their theory of quantum complexity in [Bernstein and Vazirani, 1993] based on quantum Turing machines, Andrew Yao described in [Yao, 1993] a complexity theory based on quantum circuits and gave a method of simulating computations by quantum Turing machines with quantum circuits. He used this to give a sketch of the construction of a polynomial-time universal quantum Turing machine. Bernstein and Vazirani had found another universal quantum Turing machine, but it could only efficiently simulate a special class of quantum Turing machines where the head must move at each step (two-way quantum Turing machines). Yao mentioned a forthcoming complete paper with details, but it did not appear.

By referring to the paper [Yao, 1993], it is often claimed in the literature, that the quantum Turing machine model and the quantum circuit model are computationally equivalent [2]. However, several issues were not treated. A single circuit has fixed input length, whereas a Turing machines takes an input of arbitrary length. So one must consider infinite families of circuits (one circuit for each input length). In order to compare the computational power of Turing machines and circuit families, the circuit families must satisfy some kind of uniformity condition. Or else one circuit in the family may compute faster than another. But in [Yao, 1993] families of circuits where not considered at all, and therefore a notion of uniformity was neither formulated. It could of course be, that these issues are easily treated in a way similarly as in classical complexity theory. But it turns out that, one needs a rather different notion of uniformity in quantum complexity theory.

In [Nishimura and Ozawa, 2002] a simular but different definition of simulation is formulated, than that given in [Yao, 1993]. Their definition requires that the quantum circuit simulates the quantum Turing machine for inputs of arbitrary length. They then use Yao's technique to construct a two-way efficient universal quantum Turing machine. Thus showing that, it is no restriction to only regard two-way quantum Turing machines when developing quantum complexity, as Bernstein and Vazirani did in [Bernstein and Vazirani, 1997]. They also formulate a uniformity condition which quantum circuits must satisfy and then define polynomial-time complexity classes for uniform quantum circuit families. For bounded-error computations the complexity class defined by quantum Turing machines and the class defined by quantum circuits families are shown to be the same. But they indicate that the two complexity classes do not coincide in the error-free case.

[2] See for example the discussion at the end of section 3 in [Aharonov, 1998], or the introduction of [Shor, 1994]

In the recently published paper [Nishimura and Ozawa, 2005a] it is shown that the complexity classes do coincide both for error-free and exact computations if one requires that the quantum circuit families only contain finitely many different qubit gates.

In this paper I will present quantum Turing machines and uniform quantum circuit families, and then show this equivalence. Most of the following paper consists of formalizing and defining the concepts of these two models. Many of these concepts and definitions have already been given by Nishimura and Ozawa throughout their papers, but each paper has its own objective making an inhomogeneous presentation. My work consists mainly of collecting and isolating the definitions and concepts that are necessary for proving the equivalence.

1 The quantum Turing machine model

A classical Turing machine is often described as a physical system consisting of a processor, a tape, and a head that reads and writes symbols on the tape. Something similar is also possible in the quantum case. These interpretations are some times helpful, but can also be misleading - especially in the quantum case. The presentation given here is purely mathematical and avoids using a physical interpretation. Thus many definitions and new concepts will at first be given without much reflection on their purpose.

1.1 Basics of quantum Turing machines

A *symbol set* Σ is a finite set of cardinality at least 2 with a specific element denoted by B which we refer to as the *blank symbol*.

For a symbol set Σ we define the set of functions

$$\Sigma^{\#} = \Big\{ T : \mathbb{Z} \to \Sigma \quad \Big| \quad T(m) = B \text{ except for finitely many } m \in \mathbb{Z} \Big\}$$

and refer to the functions $T \in \Sigma^{\#}$ as *tape configurations*.

For any tape configuration $T \in \Sigma^{\#}$, symbol $\tau \in \Sigma$, and integer $\xi \in \mathbb{Z}$ we define a new tape configuration T_{ξ}^{τ} by letting

$$T_{\xi}^{\tau}(m) = \begin{cases} \tau & \text{if } m = \xi, \\ T(m) & \text{if } m \neq \xi. \end{cases}$$

A *processor configuration set* Q is a finite set with two specific elements, denoted by q_0 and q_f, which we call the *initial processor configuration* and the *final processor configuration*.

A pair (Q, Σ) consisting of a processor configuration set Q and symbol set Σ is called a *Turing frame*. To each Turing frame (Q, Σ) we associate the *configuration space*

$$\mathcal{C}(Q, \Sigma) = Q \times \Sigma^{\#} \times \mathbb{Z},$$

and refer to the elements $C = (q, T, \xi)$ of $\mathcal{C}(Q, \Sigma)$ as *configurations*. The integer $\xi \in \mathbb{Z}$ is called the *head position* of C. A configuration of the form $(q_0, T, 0)$ is called an *initial configuration* and a configuration of the form (q_f, T, ξ) is called a *final configuration*.

Let (Q, Σ) be a Turing frame. The *quantum state space of* (Q, Σ), denoted by $\mathcal{H}(Q, \Sigma)$, is the Hilbert space generated by the countable set $\mathcal{C}(Q, \Sigma)$, i.e.,

$$\mathcal{H}(Q, \Sigma) = \left\{ \varphi : \mathcal{C}(Q, \Sigma) \to \mathbb{C} \quad \Big| \quad \sum_{C \in \mathcal{C}(Q, \Sigma)} |\varphi(C)|^2 < \infty \right\}$$

equipped with the inner product

$$(\varphi, \psi) = \sum_{C \in \mathcal{C}(Q,\Sigma)} \varphi(C)^\dagger \cdot \psi(C),$$

where $\varphi(C)^\dagger$ denotes the complex conjugate of $\varphi(C)$. A unit-vector in $\mathcal{H}(Q, \Sigma)$ is called a *state*.

Now is an appropriate time for a few words on notation. The reason why the inner product $(\cdot, \cdot)$ above is chosen to be conjugate linear in the first variable (not the second as usual) is that we will be using Dirac notation. Thus $|\varphi\rangle$ denotes a vector in $\mathcal{H}(Q, \Sigma)$. The inner product of vectors $|\varphi\rangle$ and $|\psi\rangle$ is written $\langle\varphi|\psi\rangle$. The functional $|\psi\rangle \mapsto \langle\varphi|\psi\rangle$ is referred to as the *dual of* $|\varphi\rangle$ and denoted by $\langle\varphi|$. The operator $|\chi\rangle \mapsto |\psi\rangle\langle\varphi|\chi\rangle$ is referred to as the *outer product of* $|\psi\rangle$ *and* $|\varphi\rangle$, and denoted by $|\psi\rangle\langle\varphi|$.

Besides using $\dagger$ to denote the complex conjugate of a number $z \in \mathbb{C}$, we will also use it on linear operators $A : \mathcal{H}(Q, \Sigma) \to \mathcal{H}(Q, \Sigma)$ to denote the adjoint operator, i.e., $A^\dagger$ denotes the adjoint operator of A.

Moreover, we adopt the following notation. For any integers $n < m$ the interval $\{n, n+1, \ldots, m-1, m\}$ will be denoted by $[n, m]_\mathbb{Z}$. The set of finite strings over a set X is denoted by X^*, and an element in X is called a *X-string*. The length of a string x is written $|x|$.

In order to have some standard representation for elements in $\mathcal{H}(Q, \Sigma)$, we define the following basis. For each element $C \in \mathcal{C}(Q, \Sigma)$ let the function $|C\rangle : \mathcal{C}(Q, \Sigma) \to \mathbb{C}$ be defined by

$$|C\rangle(C') = \begin{cases} 1 & \text{if } C' = C \\ 0 & \text{otherwise.} \end{cases}$$

Then the set

$$\Big\{ |C\rangle : \mathcal{C}(Q, \Sigma) \to \mathbb{C} \quad \Big| \quad C \in \mathcal{C}(Q, \Sigma) \Big\} \tag{1.1}$$

is obviously an orthonormal basis for $\mathcal{H}(Q, \Sigma)$. It is called the *computational basis for* (Q, Σ). Elements in the computational basis are referred to as *basis states* and when $C = (q, T, \xi)$ we just write $|q, T, \xi\rangle$ for $|C\rangle$.

A *quantum transition function* for a Turing frame (Q, Σ) is a function

$$\delta : Q \times \Sigma \times Q \times \Sigma \times [-1, 1]_\mathbb{Z} \longrightarrow \mathbb{C}.$$

A *prequantum Turing machine* is a triplet (Q, Σ, δ) such that (Q, Σ) is a Turing frame and δ is a quantum transition function.

Before we define quantum Turing machines, we introduce a particular operator determined by the quantum transition function δ. This will enable us to give a simple definition of quantum Turing machines.

Proposition 1.1. *Let $M = (Q, \Sigma, \delta)$ be a prequantum Turing machine. Then there is a unique bounded linear operator $M_\delta : \mathcal{H}(Q, \Sigma) \to \mathcal{H}(Q, \Sigma)$ which satisfies*

$$M_\delta |q, T, \xi\rangle = \sum_{(p,\tau,d) \in Q \times \Sigma \times [-1,1]_{\mathbb{Z}}} \delta(q, T(\xi), p, \tau, d) \cdot |p, T^\tau_\xi, \xi + d\rangle \tag{1.2}$$

for all $|q, T, \xi\rangle \in \mathcal{H}(Q, \Sigma)$.

Proof. (Sketch) A detailed proof of Proposition 1.1 is given in the appendix of [Nishimura and Ozawa, 2000]. It is shown that the operator norm of M_δ is bounded by $\sqrt{5}K|Q||\Sigma|^2$ where

$$K = \max_{(q,\sigma) \in Q \times \Sigma} \left(\sum_{(p,\tau,d) \in Q \times \Sigma \times [-1,1]_{\mathbb{Z}}} |\delta(q, \sigma, p, \tau, d)| \right)^{\frac{1}{2}}.$$

The idea of the proof in short is to split the operator M_δ into a sum of 5 other (more simple) operators which are each seen to be bounded by $K^2|Q|^2|\Sigma|^4$. □

The operator M_δ defined by (1.2) is called the *evolution operator* of M. Before we give the definition of a quantum Turing machine, we need to introduce the notion of efficiently computable numbers.

Informally speaking, a complex number z is polynomial-time computable if its real and imaginary parts can be approximated with accuracy of $1/2^n$ in time polynomial in n. More formally, a real number $x \in \mathbb{R}$ is *polynomial-time computable* (in $\mathbf{P}\mathbb{R}$) if there is a DTM (deterministic Turing machine) which on input 1^n computes $\phi_x \in \{m/2^n | m \in \mathbb{Z}\}$ such that $|\phi_x - x| \leq 1/2^n$, in time polynomial in n. And a complex number $z = x + iy$ is *polynomial-time computable* (in $\mathbf{P}\mathbb{C}$) if $x, y \in \mathbf{P}\mathbb{R}$.

Definition 1.2. A *quantum Turing machine* (QTM) is a prequantum Turing machine $M = (Q, \Sigma, \delta)$ such that the evolution operator M_δ of M is unitary and range$(\delta) \subseteq \mathbf{P}\mathbb{C}$.

Thus, the evolution operator must satisfy $M^\dagger_\delta M_\delta = I = M_\delta M^\dagger_\delta$, and all the values of δ must be efficiently computable. The following theorem characterizes those quantum transition functions whose evolution operators are unitary.

Theorem 1.3. *Let $M = (Q, \Sigma, \delta)$ be a prequantum Turing machine. Then, the evolution operator M_δ of M is unitary if and only if the quantum transition function δ satisfies the following conditions.*

(a) For any $(q, \sigma) \in Q \times \Sigma$,

$$\sum_{(p,\tau,d) \in Q \times \Sigma \times [-1,1]_{\mathbb{Z}}} |\delta(q, \sigma, p, \tau, d)|^2 = 1.$$

(b) For any $(q,\sigma),(q',\sigma') \in Q \times \Sigma$ *with* $(q,\sigma) \neq (q',\sigma')$,

$$\sum_{(p,\tau,d)\in Q\times\Sigma\times[-1,1]_{\mathbb{Z}}} \delta(q',\sigma',p,\tau,d)^{\dagger} \cdot \delta(q,\sigma,p,\tau,d) = 0.$$

(c) For any $(q,\sigma,\tau),(q',\sigma',\tau') \in Q \times \Sigma \times \Sigma$,

$$\sum_{(p,d)\in Q\times[0,1]_{\mathbb{Z}}} \delta(q',\sigma',p,\tau',d-1)^{\dagger} \cdot \delta(q,\sigma,p,\tau,d) = 0.$$

(d) For any $(q,\sigma,\tau),(q',\sigma',\tau') \in Q \times \Sigma \times \Sigma$,

$$\sum_{p\in Q} \delta(q',\sigma',p,\tau',-1)^{\dagger} \cdot \delta(q,\sigma,p,\tau,1) = 0.$$

Proof. (Sketch) This theorem was first stated in [Ozawa, 2000] and then proved in [Nishimura and Ozawa, 2000]. It is proved by first showing that if $M_\delta^\dagger M_\delta = I$ then, one automatically has $M_\delta M_\delta^\dagger = I$, i.e., if the evolution operator is an isometry then, it is unitary. By straight forward verification it is then seen that condition (a) holds if and only if $\langle C|M_\delta^\dagger M_\delta|C\rangle = 1$ for any $C \in \mathcal{C}(Q,\Sigma)$. Then it is shown that one automatically has $\langle C'|M_\delta^\dagger M_\delta|C\rangle = 0$ except in three different situations which arise if and only if condition (b), (c) or (d) holds. Thus showing that M_δ is an isometry if and only if conditions (a)-(d) hold. □

A QTM $M = (Q,\Sigma,\delta)$ where $\delta(p,\sigma,q,\tau,0) = 0$ for any $(p,\sigma,q,\tau) \in (Q,\Sigma)^2$ is called a *two-way* QTM. This is the restricted class of quantum Turing machines which Bernstein and Vazirani used to develop their complexity theory in [Bernstein and Vazirani, 1997].

In the next section we introduce some different projections on the quantum state space $\mathcal{H}(Q,\Sigma)$. The purpose of these projections is to extract information about the particular form of a given state $|\varphi\rangle$ (i.e., unit vector) in $\mathcal{H}(Q,\Sigma)$.

1.2 Projections on the quantum state space

Any state $|\varphi\rangle \in \mathcal{H}(Q,\Sigma)$ can be written uniquely as a finite linear combination of basis states $|C_1\rangle,\ldots,|C_n\rangle$ in the computational basis (1.1)

$$|\varphi\rangle = \sum_{i\in[1,n]_{\mathbb{Z}}} \alpha_i |C_i\rangle \tag{1.3}$$

where $\alpha_i \in \mathbb{C}\setminus\{0\}$ and $\sum_{i\in 1n}|\alpha_i|^2 = 1$. Given some state $|\varphi\rangle$, we will be interested in deducing information about the form of the $|C_i\rangle$'s in this linear combination. We do this by using projections onto different subspaces of $\mathcal{H}(Q,\Sigma)$.

The first basic projections are:

$E(\hat{q} = p)$, which is the projection onto $\mathrm{span}\{|p, T, \xi\rangle | T \in \Sigma^{\#}, \xi \in \mathbb{Z}\}$,
$E(\hat{T} = S)$, is the projection onto $\mathrm{span}\{|q, S, \xi\rangle | q \in Q, \xi \in \mathbb{Z}\}$,
$E(\hat{\xi} = \nu)$, the projection onto $\mathrm{span}\{|q, T, \nu\rangle | q \in Q, T \in \Sigma^{\#}\}$.

Note that for or any state $|\varphi\rangle \in \mathcal{H}(Q, \Sigma)$ we have

$$\sum_{(p,S,\nu)\in\mathcal{C}(Q,\Sigma)} \|E(\hat{q} = p)E(\hat{T} = S)E(\hat{\xi} = \nu)|\varphi\rangle\|^2 = 1. \tag{1.4}$$

To see this write $|\varphi\rangle$ as in (1.3). Then one sees that the sum in (1.4) is reduced to the finite sum

$$\sum_{i\in[1,n]_{\mathbb{Z}}} \|\alpha_i |C_i\rangle\|^2$$

which equals 1.

The identity (1.4) shows that for any state $|\varphi\rangle \in \mathcal{H}(Q, \Sigma)$ the function

$$(p, S, \nu) \mapsto \|E(\hat{q} = p)E(\hat{T} = S)E(\hat{\xi} = \nu)|\varphi\rangle\|^2$$

is a probability distribution on $\mathcal{C}(Q, \Sigma)$. And from this we see that for any state $|\varphi\rangle \in \mathcal{H}(Q, \Sigma)$ the functions

$$(p, \nu) \mapsto \|E(\hat{q} = p)E(\hat{\xi} = \nu)|\varphi\rangle\|^2 \quad \text{and} \quad S \mapsto \|E(\hat{T} = S)|\varphi\rangle\|^2$$

are probability distributions on $Q \times \mathbb{Z}$ and $\Sigma^{\#}$, respectively.

Assume, for example, that we are given the state $|\varphi\rangle$ and asked to determine whether it is a linear combination of initial states (see just after (1.1)). If it satisfies $\|E(\hat{q} = q_0)E(\hat{\xi} = 0)|\varphi\rangle\|^2 = 1$ then it has lost no norm by being projected onto the subspace spanned by initial states, and we can then conclude that $|\varphi\rangle$ is a linear combination of initial states. But for example if we have $\|E(\hat{q} = q_0)E(\hat{\xi} = 0)|\varphi\rangle\|^2 = 1/3$ then some of the basis states $|C_i\rangle$ in (1.3) were not initial states, and in this situation we settle by saying that $|\varphi\rangle$ is a linear combination of initial states *with probability* $1/3$.

Definition 1.4. A QTM $M = (Q, \Sigma, \delta)$ is said to be *stationary*, if for every initial state $|C_0\rangle$, there exists some $t \in \mathbb{N}$ such that $\|E(\hat{q} = q_f)E(\hat{\xi} = 0)M_\delta^t|C_0\rangle\|^2 = 1$ and for all $s < t$ we have $\|E(\hat{q} = q_f)M_\delta^s|C_0\rangle\|^2 = 0$. The positive integer t is called the *computation time* of M for state $|C_0\rangle$.

Since $M_\delta^t|C_0\rangle$ is a state in $\mathcal{H}(Q, \Sigma)$ the conditions in Definition 1.4 require that, if $M_\delta^t|C_0\rangle$ is written as in (1.3) then all the $|C_i\rangle$'s have the form $|q_f, T, 0\rangle$, and if we write $M_\delta^s|C_0\rangle$ the same way, then none of the basis states have the form $|q_f, T, \xi\rangle$.

From now on we assume that all quantum Turing machines are stationary. As shown in [Nishimura and Ozawa, 2002] (Theorem 5.8) we may consider only stationary QTMs without loss of generality to develop quantum complexity theory. By

assuming that all QTMs are stationary, we are able to talk about the computation time for any QTM.

We now introduce a special kind of tape configuration which is used to represent Σ-strings as elements in $\Sigma^{\#}$. Let $x = x_0 \cdots x_{k-1}$ where $x_m \in \Sigma$. Then we define the tape configuration $\mathrm{T}[x]$ by letting

$$\mathrm{T}[x](m) = \begin{cases} x_m & \text{if } m \in [0, k-1]_{\mathbb{Z}}, \\ B & \text{otherwise.} \end{cases}$$

We use this tape configuration to define the input/output-notion of quantum Turing machines.

Definition 1.5. Let $M = (Q, \Sigma, \delta)$ be a QTM and $x, y \in \Sigma^*$. Assume that t is the computation time of M for the initial state $|q_0, \mathrm{T}[x], 0\rangle$. Then we say that y is the *output* of M on *input* x with *probability* p, if

$$\|E(\hat{T} = \mathrm{T}[y]) M_\delta^t |q_0, \mathrm{T}[x], 0\rangle\|^2 = p. \tag{1.5}$$

Definition 1.6. A QTM $M = (Q, \Sigma, \delta)$ is said to be *polynomial-time*, if for every $x \in \Sigma^*$ the computation time t of M for $|q_0, \mathrm{T}[x], 0\rangle$ is bounded by a polynomial in the length of x.

Remark 1.7. Actually, we can assume that the computation time t is not only bounded by a polynomial, but that it equals the value of a polynomial $p(n)$ (See [Nishimura and Ozawa, 2002] Remark 2. on p.25). This enables us to talk about the computation time t for inputs of length n. And this will become useful later.

Using the two last definitions, we can now formulate the notion of recognition of languages by QTMs and define polynomial-time complexity classes.

1.3 Languages recognized by quantum Turing machines

Languages are used to represent problems to Turing machines (and other computing devices). By "problem" we mean a predicate, which can be either true or false. More formally, a predicate is a function $P : \{0,1\}^* \to \{0,1\}$. Such a function can be represented by the set $\{x \in \{0,1\}^* | P(x) = 1\}$. Subsets $L \subseteq \{0,1\}^*$ are called *languages*. If a QTM $M = (Q, \Sigma, \delta)$ satisfies that the symbol set Σ contains $\{0,1\}$ then M can take strings from a language as input. Through out this section we make the assumption that the symbol set of any QTM contains $\{0,1\}$.

We say that a QTM M *accepts* $x \in \{0,1\}^*$ *with probability* p if 1 is the output of M on input x with probability p, i.e., if M satisfies

$$\|E(\hat{T} = \mathrm{T}[1]) M_\delta^t |q_0, \mathrm{T}[x], 0\rangle\|^2 = p. \tag{1.6}$$

If M satisfies

$$\|E(\hat{T} = \mathrm{T}[0]) M_\delta^t |q_0, \mathrm{T}[x], 0\rangle\|^2 = p' \tag{1.7}$$

then we say that M *rejects* x *with probability* p'.

Let L be a language. Then we say that M *recognizes* L *with probability at least* p if the following two conditions are satisfied.

(1) M accepts x with probability at least p for any $x \in L$,

(2) M rejects x' with probability at least p for any $x' \notin L$.

Now we can define the complexity class which is the quantum analogue of **P** (the class of languages recognized by polynomial-time DTMs).

- A language L is in **EQP** if there is a polynomial-time QTM M that recognizes L with probability 1.

Before we can define the quantum analogues of **BPP** and **ZPP**, we must refine our notion of recognition slightly.

Let L be a language. Then we say that M *recognizes* L *with probability uniformly larger than* p if there is a constant $0 < \eta \leq 1 - p$ such that M recognizes L with probability at least $p + \eta$.

- A language L is in **BQP** if there is a polynomial-time QTM M that recognizes L with probability uniformly larger than $\frac{1}{2}$.

Thus, M will accept strings $x \in L$ (and reject strings $x' \notin L$) with a probability larger than 1/2. But note that, unless $p + \eta = 1$, M can still reject an $x \in L$ (or accept an $x' \notin L$) with a probability larger than 0. We can only be sure that this probability of "error" is bounded by some number less than 1/2. We rule out this possibility in the subclass defined below.

- A language L is in **ZQP** if there is a polynomial-time QTM M that recognizes L with probability uniformly larger than $\frac{1}{2}$, and M satisfies the following conditions.
  1. If M accepts x with a positive probability, then M rejects x with probability 0,
  2. If M rejects x' with a positive probability, then M accepts x' with probability 0.

From these definitions, we have obviously **EQP** $\subseteq$ **ZQP** $\subseteq$ **BQP**. In the literature these classes are often called the languages efficiently recognized by QTMs implementing deterministic, Las Vegas, and Monte Carlo algorithms, respectively. Or, Exact, Zero-error, and Bounded-error algorithms.

The goal of the next section is to define corresponding complexity classes from quantum circuit families.

2 The quantum circuit family model

Quantum circuits are often described as acyclic graphs consisting of quantum gates connected with quantum wires - similar to classical circuits. However for our purpose it is more appropriate to describe circuits as finite sequences of pairs consisting of gates and permutations. This results in a rather formal presentation. But this method is the most suited for proving the results we need.

2.1 Uniform quantum circuit families

An *index set* Λ is a subset $\{j_1, \ldots, j_n\} \subseteq \mathbb{N}$ such that $j_1 < \cdots < j_n$. The set of functions $X : \Lambda \to \{0,1\}$ is called the *binary register over* Λ, and written $\{0,1\}^\Lambda$.

Let Λ be an index set. Then the Λ-*qubit register*, denoted by $\mathcal{H}(\Lambda)$, is the 2^n-dimensional Hilbert space generated by the set $\{0,1\}^\Lambda$, i.e.,

$$\mathcal{H}(\Lambda) = \left\{\varphi : \{0,1\}^\Lambda \to \mathbb{C}\right\}$$

equipped with the inner product

$$(\varphi, \psi) = \sum_{X \in \{0,1\}^\Lambda} \varphi(X)^\dagger \cdot \psi(X).$$

In particular, $\mathcal{H}([1,n]_\mathbb{Z})$ is called the *n-qubit register*.

As in the previous section we will be using Dirac notation. Similarly we define a standard basis. For each element $X \in \{0,1\}^\Lambda$ let the function $|X\rangle : \{0,1\}^\Lambda \to \mathbb{C}$ be defined by

$$|X\rangle(X') = \begin{cases} 1 & \text{if } X' = X \\ 0 & \text{otherwise.} \end{cases}$$

Then the set

$$\left\{|X\rangle : \{0,1\}^\Lambda \to \mathbb{C} \quad \middle| \quad X \in \{0,1\}^\Lambda\right\} \tag{2.1}$$

is obviously an orthonormal basis for $\mathcal{H}(\Lambda)$. It is called the *computational basis on* Λ. As in the previous section, we will refer to unit vectors in $\mathcal{H}(\Lambda)$ as *states* and elements in the basis (2.1) as *basis states*.

Before continuing to the definition of quantum gates, it is appropriate to mention a few things regarding notation.

First, we need some convenient way of specifying the elements $|X\rangle$ in the computational basis. For this purpose we will identify mappings $X : \Lambda \to \{0,1\}$ with binary strings in the following way. Assume that the index set Λ consists of n elements. Then there is a one-to-one correspondence between n-bit strings x and elements $X \in \{0,1\}^\Lambda$. The binary string $x_1 \cdots x_i \cdots x_n$ corresponds to the function X such that $X(j_i) = x_i$ for all $i \in [1,n]_\mathbb{Z}$, and the function Y corresponds to

the binary string $Y(j_1)\cdots Y(j_i)\cdots Y(j_n)$. Thus, if $x_1\cdots x_n$ is the binary string corresponding to X then we will write $|x_1\cdots x_n\rangle$ for the basis state $|X\rangle$, and vice versa.

Second, when specifying operators on the space $\mathcal{H}(\Lambda)$, it will be convenient to view $\mathcal{H}(\Lambda)$ as the product of other quantum registers, in particular the tensor product of the qubit registers $\mathcal{H}(\{j\})$ where $j \in \Lambda$, i.e.,

$$\mathcal{H}(\Lambda) = \bigotimes_{j\in\Lambda} \mathcal{H}(\{j\}).$$

Hence, we will not distinguish between the vector $|x_1\cdots x_n\rangle$ and $|x_1\rangle \otimes \cdots \otimes |x_n\rangle$. This last vector is abbreviated $|x_1, \ldots, x_n\rangle$ - using commas.

A unitary operator $G : \mathcal{H}(\Lambda) \to \mathcal{H}(\Lambda)$ is called a Λ-*qubit gate*. In particular, a $[1,n]_{\mathbb{Z}}$-qubit gate is called an n-qubit gate. The *S-matrix* of a Λ-qubit gate G is the matrix representing the operator G in the computational basis over Λ. Usually we will identify the S-matrix of a quantum gate with the quantum gate itself.

In contrast to the situation in the previous section, we are now working in a finite dimensional vector space $\mathcal{H}(\Lambda)$. This means, among other things, that an operator G that satisfies $G^\dagger G = I$ will automatically satisfy $GG^\dagger = I$, or in other words, isometries are unitary.

We now give two important examples of n-qubit gates. These will enable us to define the notion of decomposability in a simple way.

Let π be a permutation on $[1,n]_{\mathbb{Z}}$. The *permutation operator* of π, denoted by V_π, is the n-qubit gate that maps $|x_1\cdots x_n\rangle$ into $|x_{\pi(1)}, \ldots, x_{\pi(n)}\rangle$ for any n-bit string $x_1\cdots x_n$.

Let G be an m-qubit gate and $n \in \mathbb{N}$ such that $n \geq m$. The *n-qubit extension* of G, denoted by $G[n]$, is the n-qubit gate that acts like G on the first m factors of $\mathcal{H}_{[1,n]_{\mathbb{Z}}}$ and as the identity on the last $n-m$, i.e.,

$$G[n] = G \otimes I_{[m+1,n]_{\mathbb{Z}}}$$

where $I_{[m+1,n]_{\mathbb{Z}}}$ denotes the identity operator on $\mathcal{H}([m+1,n]_{\mathbb{Z}})$.

Let $\mathcal{G}$ be a set of qubit gates on, not necessarily the same, qubit registers. Then we say that an n-qubit gate G is *decomposable into m qubit gates in $\mathcal{G}$* if there are n_i-qubit gates $G_i \in \mathcal{G}$ with $n_i \leq n$ and permutations π_i on $[1,n]_{\mathbb{Z}}$ satisfying

$$G = U_m \cdots U_1, \quad \text{where} \quad U_i = V_{\pi_i}^\dagger G_i[n] V_{\pi_i} \quad \text{for} \quad i \in [1,m]_{\mathbb{Z}}. \tag{2.2}$$

The method, by which the operator U_i above is constructed, is very useful. It allows us to operate on an arbitrary set of factors in a vector $|x_1, \ldots, x_n\rangle$. First, the n_i factors, which we want to operate on, are collected by V_{π_i} up front. Then $G_i[n]$

operates on these n_i factors and leaves the rest alone. And at last the factors are returned to there original place by $V_{\pi_i}^{\dagger}$, which works as the inverse of V_{π_i}.

The least number m of qubit gates in $\mathcal{G}$ which G can be decomposed into is called the *size of* G *for* $\mathcal{G}$. In most situations, we are only sure that the size is less than some particular m, and there fore we normally just say that G has size $\mathcal{O}(m)$ for $\mathcal{G}$, or that G is decomposable into $\mathcal{O}(m)$ qubit gates in $\mathcal{G}$.

We shall now introduce a specific set $\mathcal{G}_u$ of quantum gates that plays an important role in complexity theory. It consists of two types of 1-qubit gates and one specific 2-qubit gate.

Let $R_{1,\theta}$ and $R_{2,\theta}$ denote the 1-qubit gates whose S-matrices are given as follows.

$$R_{1,\theta} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \qquad R_{2,\theta} = \begin{pmatrix} e^{i\theta} & 0 \\ 0 & 1 \end{pmatrix}.$$

And let N denote the 2-qubit gate whose S-matrix is given by

$$N = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The gate $R_{1,\theta}$ is called the rotation gate by angle θ and $R_{2,\theta}$ is called the phase shift gate by by angle θ. N is referred to as the controlled-not gate.

Let $\mathcal{G}_u$ be the set defined by

$$\mathcal{G}_u = \left\{ R_{1,\theta}, R_{2,\theta}, N \quad \middle| \quad \theta \in \mathbf{P}\mathbb{R} \cap [0, 2\pi) \right\}. \tag{2.3}$$

Note that since there are only countably many DTMs, $\mathbf{P}\mathbb{R}$ is a countable set and thus the number of gates in $\mathcal{G}_u$ is also countable.

Theorem 2.1. *Let G be an n-qubit gate such that the entries of its S-matrix all belong to $\mathbf{P}\mathbb{C}$. Then G can be decomposed into $\mathcal{O}(2^{2n}n^3)$ qubit gates from $\mathcal{G}_u$.*

Proof. (Sketch) The proof of this theorem is given in [Nishimura and Ozawa, 2005a]. It builds on an earlier result from [Barenco et al., 1995], namely that any 2^n-dimensional near-trivial matrix M can be decomposed into $\mathcal{O}(n^3)$ matrices from $\{R_{1,\theta}, R_{2,\theta}, N | \theta \in [0, 2\pi)\}$. A near-trivial matrix U, is a matrix that has one of the following two forms.

1. U is like a unit matrix except for one diagonal element which has the form $e^{i\theta}$,
2. U is like a unit matrix except for the elements in the intersections one row and one column which form the matrix $\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$,

where $\theta \in [0, 2\pi)$. In [Nishimura and Ozawa, 2005a] it is proved that any 2^n-dimensional unitary matrix U can be decomposed into $\mathcal{O}(2^{2n})$ near-trivial matrices. By keeping account of the entries of the matrices (whether they stay in $\mathbf{P}\mathbb{C}$) through these two constructions, one obtains the theorem. □

Note that the number of gates required to decompose an n-qubit gate grows exponentially when n grows.

Definition 2.2. Let $\mathcal{G}$ be a set of qubit gates. An *n-qubit circuit K based on $\mathcal{G}$* is a finite sequence $(G_m, \pi_m), \ldots, (G_1, \pi_1)$ such that each pair (G_i, π_i) satisfies the following conditions.

(1) G_i is an n_i-qubit gate in $\mathcal{G}$ with $n_i \leq n$.

(2) π_i is a permutation on $[1, n]_{\mathbb{Z}}$.

The positive integer m is called the *size of K for $\mathcal{G}$*, and the unitary operator

$$U_m \cdots U_1, \quad \text{where} \quad U_i = V_{\pi_i}^\dagger G_i[n] V_{\pi_i} \quad \text{for} \quad i \in [1, m]_{\mathbb{Z}}$$

is called the *n-qubit gate determined by K* and denoted by $G(K)$.

Note that, by definition, $G(K)$ is decomposable into m qubit gates in $\mathcal{G}$. Hence, we can also view a qubit circuit K of size m for $\mathcal{G}$ as a description of how to decompose a qubit gate $G(K)$ into m gates in $\mathcal{G}$.

Qubit circuits are often constructed by combining other qubit circuits. Suppose that $K_1 = (G_m, \pi_m), \ldots, (G_1, \pi_1)$ and $K_2 = (G'_l, \pi'_l), \ldots, (G'_1, \pi'_1)$ are n-qubit circuits based on $\mathcal{G}$. Then

$$K_2 \circ K_1 = (G'_l, \pi'_l), \ldots, (G'_1, \pi'_1), (G_m, \pi_m), \ldots, (G_1, \pi_1)$$

is called the *concatenation* of K_1 and K_2, in particular $\underbrace{K_1 \circ \cdots \circ K_1}_{n}$ is called the concatenation of n copies of K_1 and denoted K_1^n.

The qubit circuit defined in Definition 2.2 is not yet suited for computing. The gate $G(K)$ takes a 2^n-dimensional vector and outputs a 2^n-dimensional vector. In most situations the length of the input and output is not the same. In the following definition we supply the qubit circuit with the extra structure to handle this.

Definition 2.3. Let $\mathcal{G}$ be a set of qubit gates. A *k-input m-output n-qubit circuit $\mathbb{K}$ based on $\mathcal{G}$* is a 4-tuple $(K, \Lambda_1, \Lambda_2, S)$ satisfying the following conditions.

(1) K is an n-qubit circuit based on $\mathcal{G}$.

(2) Λ_1 and Λ_2 are two subsets of $[1, n]_{\mathbb{Z}}$ satisfying $|\Lambda_1| = k$ and $|\Lambda_2| = m$, respectively.

(3) S is a function from $[1,n]_{\mathbb{Z}} \setminus \Lambda_1$ to $\{0,1\}$.

Condition (2) takes care of the input and output, i.e., the index sets Λ_1 and Λ_2 each select a specific subset of the factors in $|x_1,\ldots,x_n\rangle$ which are to be considered as input and output. The function S in condition (3) is used to set the factors which are not considered as input with some fixed value.

The *size based on* $\mathcal{G}$ of $\mathbb{K} = (K, \Lambda_1, \Lambda_2, S)$ is simply the size of K based on $\mathcal{G}$.

We are now finally able to define quantum circuit families.

Definition 2.4. Let $\mathcal{G}$ be a set of qubit gates. A *quantum circuit family (QCF) $\mathcal{K}$ based on $\mathcal{G}$* is an infinite sequence $\{\mathbb{K}_n\}_{n\in\mathbb{N}}$ such that each $\mathbb{K}_n$ is an n-input $\big(f(n)$-output $g(n)$-qubit$\big)$ circuit based on $\mathcal{G}$. The *size s based on $\mathcal{G}$* of $\mathcal{K}$ is the function $s : \mathbb{N} \to \mathbb{N}$ such $s(n)$ is the size of $\mathbb{K}_n$ based on $\mathcal{G}$. If s is bounded by some polynomial in n, $\mathcal{K}$ is said to be *polynomial-size*.

Thus, we dedicate an n-qubit circuit to each input length n.

However, we are not yet finished. In order for the quantum circuit family model to be a reasonable model of computation, we must require that quantum circuit families are constructible by polynomial time bounded DTMs. In the case with a QTM $M = (Q, \Sigma, \delta)$, this was rather simple - we just required that range$(\delta) \subseteq \mathbf{P}\mathbb{C}$. With a QCF $\mathcal{K} = \{\mathbb{K}_n\}_{n\in\mathbb{N}}$ based on $\mathcal{G}$, we must obviously require that the entries of the S-matrices of the qubit gates in $\mathcal{G}$ are all contained in $\mathbf{P}\mathbb{C}$. But this alone is not enough. Besides having to construct the qubit gates in $\mathcal{G}$ we must also be able to construct the index sets Λ_1 and Λ_2, and the function S (from Definition 2.3) of each $\mathbb{K}_n$. The first requirement, regarding the S-matrices, is overcome by only considering QCFs based on subsets of $\mathcal{G}_u$. By Theorem 2.1 any qubit gate with S-matrix entries in $\mathbf{P}\mathbb{C}$ is decomposable into qubit gates from $\mathcal{G}_u$. Hence, this restriction gives exactly what we need. In order to formalize the second requirement, regarding Λ_1, Λ_2, and S, we need to introduce the code of a qubit circuit.

In Section 1.1 we briefly defined efficiently computable numbers. A complex number $z = x + iy$ is in $\mathbf{P}\mathbb{C}$ if there are polynomial-time DTMs which approximate x and y. We shall now define the code of a number $z \in \mathbf{P}\mathbb{C}$. Let $x \in \mathbf{P}\mathbb{R}$, then we define the *code* of x, denoted by $c(x)$, to be the encoding $\langle M\rangle$ of the first polynomial-time DTM M which approximates x. We can talk about the "first" DTM because the set of DTMs is numerable. Assume that $z \in \mathbf{P}\mathbb{C}$. Then the *code* $c(z)$ of $z = x + iy$ is defined to be $\langle c(x), c(y)\rangle$.

Definition 2.5. Let $K = (G_m, \pi_m), \ldots, (G_1, \pi_1)$ be a qubit circuit based on $\mathcal{G}_u$. Then the *code* of K, denoted by $c(K)$, is defined to be the list of finite sequences of natural numbers $\langle e(G_1), \ldots, e(G_m)\rangle$, where

$$e(G_j) = \begin{cases} \langle\langle i, c(\theta)\rangle, \pi_j(1)\rangle & \text{if } G_j = R_{i,\theta} \\ \langle\langle 3, \pi_j(1)\rangle, \pi_j(2)\rangle & \text{if } G_j = N \end{cases} \qquad \text{for} \quad j \in [1,m]_{\mathbb{Z}}.$$

Let $\mathbb{K} = (K, \Lambda_1, \Lambda_2, S)$ be a k-input m-output n-qubit circuit based on $\mathcal{G}_u$, where $[1,n]_{\mathbb{Z}} \setminus \Lambda_1 = \{i_1, \ldots, i_{n-k}\}$ and $\Lambda_2 = \{j_1, \ldots, j_m\}$. The *code of* $\mathbb{K}$, denoted by $c(\mathbb{K})$, is defined to be the list of finite sequences of natural numbers,

$$c(\mathbb{K}) = \langle\langle\langle i_1, S(i_1)\rangle\rangle, \ldots, \langle i_{n-k}, S(i_{n-k})\rangle\rangle, c(K), \langle j_1, \ldots, j_m\rangle\rangle.$$

Using the code of a qubit circuit, we can now formulate the notion of uniformity.

Definition 2.6. Let $\mathcal{K} = \{\mathbb{K}_n\}_{n\in\mathbb{N}}$ be a QCF based on a subset $\mathcal{G}$ of $\mathcal{G}_u$. Then $\mathcal{K}$ is said to be *uniform*, if the function $1^n \mapsto c(\mathbb{K}_n)$ is computable by a polynomial-time DTM.

Note that if a QCF is uniform, then it must also be polynomial-size.

Definition 2.7. Let $\mathcal{K}$ be a uniform QCF. If there is a finite subset $\mathcal{G}$ of $\mathcal{G}_u$ such that $\mathcal{K}$ is based on $\mathcal{G}$, then we say that $\mathcal{K}$ is *finitely-generated*.

In what follows, when we say that a QCF is finitely-generated, then it is implicitly given that it is uniform.

In Section 2.3 we will define complexity classes from finitely-generated quantum circuit families. But before we do this, we introduce a projection on the qubit register.

2.2 The output-distribution of a quantum circuit

Assume that $\mathbb{K} = (K, \Lambda_1, \Lambda_2, S)$ is a k-input m-output n-qubit circuit, where $\Lambda_1 = \{j_1, \ldots, j_k\}$ and $\Lambda_2 = \{i_1, \ldots, i_m\}$.

For any k-bit string $x = x_1 \cdots x_k$, we define an n-bit string $u(x, \mathbb{K}) = u_1 \cdots u_n$ such that

$$u_{j_1} = x_1, \ldots, u_{j_k} = x_k \quad \text{and} \quad u_j = S(j) \text{ for all } j \in [1,n]_{\mathbb{Z}} \setminus \Lambda_1.$$

And for any basis state $|z\rangle$, with $z = z_1 \cdots z_n$, in the qubit register $\mathcal{H}([1,n]_{\mathbb{Z}})$, we define an m-bit string $v(|z\rangle, \mathbb{K}) = v_1 \cdots v_m$ such that

$$v_1 = z_{i_1}, \ldots, v_m = z_{i_m}.$$

For any m-bit string y let $E(\hat{v} = y)$ be the projection onto

$$\text{span}\Big\{ |z\rangle \in \mathcal{H}([1,n]_{\mathbb{Z}}) \quad \Big| \quad v(|z\rangle, \mathbb{K}) = y \Big\}.$$

We say that y is the *output* of $\mathbb{K}$ on *input* x with *probability* p, if

$$\|E(\hat{v} = y)G(K)|u(x, \mathbb{K})\rangle\|^2 = p.$$

For any $x \in \{0,1\}^k$ we define the function $\rho^{\mathbb{K}}(\cdot|x) : \{0,1\}^m \to [0,1]$ by

$$\rho^{\mathbb{K}}(y|x) = \|E(\hat{v} = y)G(K)|u(x, \mathbb{K})\rangle\|^2 \tag{2.4}$$

and refer to it as the *output-distribution* of $\mathbb{K}$ for input x.

2.3 Languages recognized by quantum circuit families

Let $\mathbb{K}$ be an n-input 2-output qubit circuit and $x \in \{0,1\}^n$. Then $\mathbb{K}$ is said to *accept x with probability p* if 01 is the output of $\mathbb{K}$ on input x, i.e., if $\mathbb{K}$ satisfies

$$\rho^{\mathbb{K}}(01|x) = p.$$

If $\mathbb{K}$ satisfies

$$\rho^{\mathbb{K}}(00|x) = p'$$

then we say that $\mathbb{K}$ *rejects x with probability p'*.

Let $L_n \subseteq \{0,1\}^n$. Then we say that $\mathbb{K}$ *recognizes L_n with probability at least p* if the following two conditions are satisfied.

(1) $\mathbb{K}$ accepts x with probability at least p for any $x \in L_n$,

(2) $\mathbb{K}$ rejects x' with probability at least p for any $x' \notin L_n$.

Let L be a language. A QCF $\mathcal{K} = \{\mathbb{K}_n\}_{n\in\mathbb{N}}$ is said to *recognize L with probability at least p* if $\mathbb{K}_n$ recognizes $L_n = L \cap \{0,1\}^n$ with probability at least p for any $n \in \mathbb{N}$.

- A language L is in **EFPQC** if there is a finitely-generated QCF $\mathcal{K}$ that recognizes L with probability 1.

A QCF $\mathcal{K}$ is said to *recognize L with probability uniformly larger p* if there is a constant $0 < \eta < 1 - p$ such that $\mathcal{K}$ recognizes L with probability at least $p + \eta$.

- A language L is in **BFPQC** if there is a finitely-generated QCF $\mathcal{K}$ that recognizes L with probability uniformly larger than $\frac{1}{2}$.

- A language L is in **ZFPQC** if there is a finitely-generated QCF $\mathcal{K} = \{\mathbb{K}_n\}_{n\in\mathbb{N}}$ that recognizes L with probability uniformly larger than $\frac{1}{2}$, and satisfies

$$\rho^{\mathbb{K}_{|x|}}(00|x) = 0 \quad \text{or} \quad \rho^{\mathbb{K}_{|x|}}(01|x) = 0 \quad \text{for any} \quad x \in \{0,1\}^*. \tag{2.5}$$

From these definitions, we have obviously **EFPQC** $\subseteq$ **ZFPQC** $\subseteq$ **BFPQC**.

The goal of the next section is to show that these complexity classes are the same as those defined in Section 1.3.

3 Computational equivalence

In the following section we define the notion of simulation between QTMs and QCFs, and then use this to show that polynomial-time QTMs and finitely-generated QCFs recognize the same class of languages.

3.1 Encoding and simulation of quantum Turing machines

Before we can introduce the notion simulation of QTMs by QCFs, we define another projection on the quantum state space $\mathcal{H}(Q,\Sigma)$ of a QTM $M=(Q,\Sigma,\delta)$.

For any $\sigma\in\Sigma$ and $m\in\mathbb{Z}$ let $E(\hat{T}(m)=\sigma)$ denote the projection onto $\mathrm{span}\{|q,T,\xi\rangle | q\in Q, T(m)=\sigma, \xi\in\mathbb{Z}\}$. Then for any $t\in\mathbb{N}$ and Σ-string x we define the probability distribution $\rho_t^M(\cdot|x)$ on Σ^{2t+1} by

$$\rho_t^M(y|x)=\|E(\hat{T}(-t)=y_1)\cdots E(\hat{T}(t)=y_{2t+1})M_\delta^t|q_0,\mathrm{T}[x],0\rangle\|^2.$$

This distribution serves a similar purpose as the output-distribution of a quantum circuit.

The qubit circuit $\mathbb{K}_n$ in a QCF $\mathcal{K}=\{\mathbb{K}_n\}_{n\in\mathbb{N}}$, takes binary strings of length n as input, whereas the QTM $M=(Q,\Sigma,\delta)$ takes Σ-strings of arbitrary length as input. To handle this we define an encoding function which maps Σ-strings into binary strings of length n.

Let $e:\Sigma\to\{0,1\}^\lambda$, where $\lambda=\lceil\log|\Sigma|\rceil$, be an injection computable by a polynomial-time DTM. For any $t\in\mathbb{N}$ we define the *encoding function* $e_t:\Sigma^*\to\{0,1\}^{(2t+1)\lambda}$ by

$$e_t(x_1\cdots x_k)=\begin{cases}\underbrace{e(B)\cdots e(B)}_{t}\,e(x_1)\cdots e(x_k)\,\underbrace{e(B)\cdots e(B)}_{t+1-k} & \text{if } t+1\geq k,\\ \underbrace{e(B)\cdots e(B)}_{t}\,e(x_1)\cdots e(x_{t+1}) & \text{if } t+1<k.\end{cases}$$

Next, we define a decoding on the image of e_t. Let $d:e(\Sigma)\to\Sigma$ be a function computable by a polynomial-time DTM such that $d\cdot e=\mathrm{id}$. For any $t\in\mathbb{N}$ we define the *decoding function* $d_t:e_t(\Sigma^*)\to\Sigma^{2t+1}$ by

$$d_t(z_1\cdots z_{2t+1})=d(z_1)\cdots d(z_{2t+1}).$$

Definition 3.1. Let $M=(Q,\Sigma,\delta)$ be a QTM and t some positive integer. A $(2t+1)\lambda$-input $(2t+1)\lambda$-output qubit circuit $\mathbb{K}$ is said to *t-simulate* M (under the encoding e_t and decoding d_t), if for any $x\in\Sigma^*$ and $y\in\Sigma^{2t+1}$, we have

$$\rho_t^M(y|x)=\rho^{\mathbb{K}}(d_t^{-1}(y)|e_t(x)).$$

A QCF $\mathcal{K}=\{\mathbb{K}_n\}_{n\in\mathbb{N}}$ is said to *simulate* M, if for any $n\in\mathbb{N}$ the qubit circuit $\mathbb{K}_n$ $t(n)$-simulates M, where $t(n)$ is the computation time of M for inputs of length n (See Remark 1.7).

In [Yao, 1993] a similar, but different definition of simulation was given. The difference is that the Σ-string x must have length $\lceil n/\lceil \log|\Sigma|\rceil\rceil$ - there is no encoding function e_t to shorten off or supplement x. In the literature, this kind of simulation is often referred to as (n,t)-simulation.

Theorem 3.2. *Let $M = (Q, \Sigma, \delta)$ be a polynomial-time QTM. Then, there is a finitely-generated QCF $\mathcal{K} = \{\mathbb{K}_n\}_{n\in\mathbb{N}}$ that simulates M.*

Proof. Let $t(n)$ be the computation time of M on input of length n. First, we fix n and construct a qubit circuit $\mathbb{K}$ which t-simulates M. Henceforth, let $t = t(n)$. The qubit circuit $\mathbb{K} = (K, \Lambda_1, \Lambda_2, S)$ will be a $(2t+1)\lambda$-input $(2t+1)\lambda$-output $(l_0 + (2t+1)(\lambda+2))$-qubit circuit, where $\lambda = \lceil \log|\Sigma|\rceil$ and $l_0 = \lceil \log|Q|\rceil$. For the sake of simplicity, let $k = 2t+1$ and $l = (\lambda+2)$. Thus making $\mathbb{K}$ a $k\lambda$-input $k\lambda$-output (l_0+kl)-qubit circuit. The first l_0 factors of $\mathcal{H}_{[1,l_0+kl]_\mathbb{Z}}$ will be used to represent the processor configuration q. The next l factors will represent the symbol $T(-t)$ and whether the head position ξ is $-t$ or not. The next l represent $T(-t+1)$ and if $\xi = -t+1$, and so on. In what follows, $p, q, \ldots$ denote binary strings representing elements of Q, the symbols $\sigma, \tau, \ldots$ denote binary strings representing elements of Σ, and $s = \bar{0}, \bar{1}, \bar{2}$ denote $00, 10, 11$, respectively. Then we denote the computational basis state $|q\sigma_1 s_1 \sigma_2 s_2 \cdots \sigma_k s_k\rangle$ on $[1, l_0+kl]_\mathbb{Z}$ by $|q; \sigma_1 s_1; \sigma_2 s_2; \cdots; \sigma_k s_k\rangle$.

We start by defining two qubit gates G_1 and G_2 which we will base our circuit $\mathbb{K}$ on. G_1 is an (l_0+3l)-qubit gate satisfying the following conditions.

(i) $G_1|w_{p,\sigma_1,\sigma,\sigma_3}\rangle = |v_{p,\sigma_1,\sigma,\sigma_3}\rangle$ where

$$\begin{aligned}
|w_{p,\sigma_1,\sigma,\sigma_3}\rangle &= |p; \sigma_1\bar{0}; \sigma\bar{1}; \sigma_3\bar{0}\rangle, \\
|v_{p,\sigma_1,\sigma,\sigma_3}\rangle &= \sum_{(q,\tau)\in Q\times\Sigma} \delta(p,\sigma,q,\tau,-1)|q; \sigma_1\bar{2}; \tau\bar{0}; \sigma_3\bar{0}\rangle \\
&\quad + \sum_{(q,\tau)\in Q\times\Sigma} \delta(p,\sigma,q,\tau,0)|q; \sigma_1\bar{0}; \tau\bar{2}; \sigma_3\bar{0}\rangle \\
&\quad + \sum_{(q,\tau)\in Q\times\Sigma} \delta(p,\sigma,q,\tau,1)|q; \sigma_1\bar{0}; \tau\bar{0}; \sigma_3\bar{2}\rangle.
\end{aligned}$$

for any $(p, \sigma_1, \sigma, \sigma_3) \in Q \times \Sigma^3$.

(ii) $G_1|h\rangle = |h\rangle$ for each $|h\rangle$ in the subspace H of $\mathcal{H}_{[1,l_0+3l]_\mathbb{Z}}$ spanned by the three types of vectors:

$$\begin{aligned}
|h^1_{q,\sigma_1,\sigma_2,\sigma_3}\rangle &= |q; \sigma_1 s_1; \sigma_2 s_2; \sigma_3 s_3\rangle \text{ where } s_1 \neq \bar{1} \text{ and } s_1, s_2, s_3 \neq \bar{2}; \\
|h^2_{p,\sigma,\sigma_2,\sigma_3}\rangle &= \sum_{(q,\tau)\in Q\times\Sigma} \delta(p,\sigma,q,\tau,0)|q; \tau\bar{2}; \sigma_2\bar{0}; \sigma_3\bar{0}\rangle + \\
&\quad \sum_{(q,\tau)\in Q\times\Sigma} \delta(p,\sigma,q,\tau,1)|q; \tau\bar{0}; \sigma_2\bar{2}; \sigma_3\bar{0}\rangle; \\
|h^3_{p,\sigma,\tau,\sigma_1,\sigma_2,\sigma_3}\rangle &= \sum_{q\in Q} \delta(p,\sigma,q,\tau,1)|q; \sigma_1\bar{2}; \sigma_2\bar{0}; \sigma_3\bar{0}\rangle.
\end{aligned}$$

The motivation behind these conditions will become obvious later. In order to see that there actually exists a qubit gate satisfying these conditions, let $W = \{|w_{p,\sigma,\sigma_1,\sigma_3}\rangle | (p,\sigma,\sigma_1,\sigma_3) \in Q \times \Sigma^3\}$ and $V = \{|v_{p,\sigma,\sigma_1,\sigma_3}\rangle | (p,\sigma,\sigma_1,\sigma_3) \in Q \times \Sigma^3\}$. Then, obviously W is an orthonormal system. By using condition (a) of Theorem 1.3 one can, by straight forward verification, show that each vector in V has unit norm. And by using condition (b) of Theorem 1.3 one sees that all the vectors in V are orthogonal to each other. Or in other words, V is also an orthonormal system. Obviously, the first type of vector $|h^1_{q,\sigma_1,\sigma_2,\sigma_3}\rangle$ in H is orthogonal to vectors in V. By using condition (c) of Theorem 1.3 one can, by straight forward verification, show that the second type of vector $|h^2_{q,\sigma,\sigma_2,\sigma_3}\rangle$ in H is orthogonal to V. And using condition (d) of Theorem 1.3 one sees that $|h^3_{p,\sigma,\tau,\sigma_1,\sigma_2,\sigma_3}\rangle$ is also orthogonal V. Thus, $H \perp V$. Obviously we have $H \perp W$ and $W \perp V$. So, condition (i) requires that the operator G_1 maps the orthonormal subset W of $\mathcal{H}_{l_0+3l}$ bijectively into the other orthonormal subset V which is orthogonal to W. And condition (ii) requires that G_1 is the identity on the subspace H which is orthogonal to both W and V. These requirements are consistent with unitarity, and therefore a unitary operator G_1 satisfying these conditions exists.

Let G_2 be an $(l_0 + kl)$-qubit gate which does nothing except for mapping all $s_i = \bar{2}$'s to $s_i = \bar{1}$'s. and vice versa.

The qubit circuit $\mathbb{K} = (K, \Lambda_1, \Lambda_2, S)$ will be based on the set $\mathcal{G} = \{G_1, G_2\}$. Let K_1 be the $(l_0 + kl)$-qubit circuit

$$\underbrace{(G_2, \mathrm{id}), (G_1, \pi_{k-1}), \ldots, (G_1, \pi_2), (G_1, \mathrm{id})}_{k-1}$$

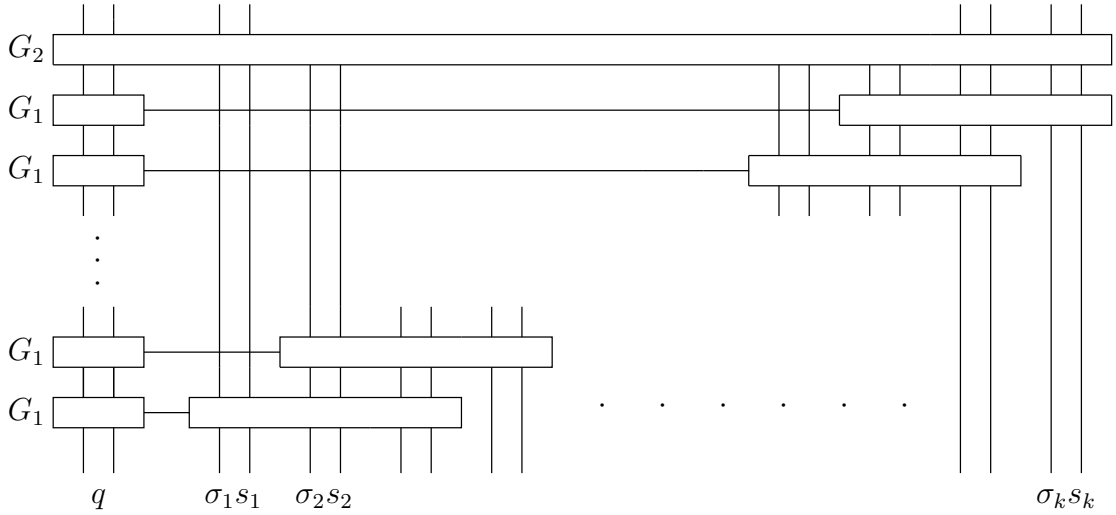
where π_i is the permutation such that V_{π_i} maps

$$|q; \sigma_1 s_1; \cdots; \sigma_{i-1} s_{i-1}; \sigma_i s_i; \sigma_{i+1} s_{i+1}; \cdots; \sigma_k s_k\rangle$$

to

$$|q; \sigma_{i-1} s_{i-1}; \sigma_i s_i; \sigma_{i+1} s_{i+1}; \sigma_1 s_1; \cdots; \sigma_{i-2} s_{i-2}; \sigma_{i+2} s_{i+2}; \cdots \sigma_k s_k\rangle,$$

i.e., where $\sigma_{i-1} s_{i-1}; \sigma_i s_i; \sigma_{i+1} s_{i+1}$ is pushed up in front. The following diagram illustrates the structure of K_1.



The qubit circuit K in $\mathbb{K}$ is defined to be the concatenation of t copies of K_1. Note that the qubit gate $G(K)$ determined by K will then be equal to the gate $(G(K_1))^t$.

Next, we define the sets Λ_1, Λ_2, and the function S. Λ_1, Λ_2 are both to be the subset of $[1, l_0+kl]_{\mathbb{Z}}$ that corresponds to the indexing of the σ_i's in $|q; \sigma_1 s_1; \sigma_2 s_2; \cdots ; \sigma_i s_i; \cdots ; \sigma_k s_k\rangle$. Or more formally, the set

$$\{l_0 + 1, \ldots, l_0 + \lambda; l_0 + \lambda + 3 \ldots; l_0 + 2\lambda + 3; \ldots\}.$$

And finally, the the function S from $[1, l_0 + kl]_{\mathbb{Z}} \setminus \Lambda_1$ to $\{0, 1\}$. It maps the first l_0 bits to q_0 - the binary string representing the initial processor configuration in Q. And sets all the s_i's to $\bar{0}$ except σ_{t+1} which is set to $\bar{1}$. Hence, we get

$$|u(e_t(x), \mathbb{K})\rangle = |q_0; \underbrace{B\bar{0}; \cdots ; B\bar{0}}_{t}; x_0\bar{1}; x_1\bar{0}; \cdots ; x_m\bar{0}\rangle$$

In order to see that $\mathbb{K}$ t-simulates M, we must show that for any $x \in \Sigma^*$ and $y \in \Sigma^{2t+1}$ we have

$$\rho_t^M(y|x) = \rho^{\mathbb{K}}(d_t^{-1}(y)|e_t(x))$$

or

$$\|E(\hat{T}(-t) = y_1) \cdots E(\hat{T}(t) = y_{2t+1}) M_\delta^t |q_0, \mathrm{T}[x], 0\rangle\| =$$

$$\|E(\hat{v} = d_t^{-1}(y))(G(K_1))^t |u(e_t(x), \mathbb{K})\rangle\|.$$

That this holds, will become obvious by applying M_δ and $G(K_1)$ just once to the vectors $|q_0, \mathrm{T}[x], 0\rangle$ and $|u(e_t(x), \mathbb{K})\rangle$, respectively:

From the definition of M_δ (Proposition 1.1) we get

$$M_\delta|q_0, T[x_1 \cdots x_m], 0\rangle = \sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, -1)|q, \mathrm{T}[\tau x_1 \cdots x_m], -1\rangle + \sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, 0)|q, \mathrm{T}[\tau x_1 \cdots x_m], 0\rangle + \sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, 1)|q, \mathrm{T}[\tau x_1 \cdots x_m], 1\rangle.$$

And from the definition of K_1 we first get

$$G(K_1)|q_0; B\bar{0}; \cdots; B\bar{0}; x_0\bar{1}; x_1\bar{0}; \cdots; x_t\bar{0}\rangle = \sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, -1)|q; B\bar{0}; \cdots; B\bar{2}; \tau\bar{0}; x_1\bar{0}; \cdots; x_t\bar{0}\rangle + \sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, 0)|q; B\bar{0}; \cdots; B\bar{0}; \tau\bar{2}; x_1\bar{0}; \cdots; x_t\bar{0}\rangle + \sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, 1)|q; B\bar{0}; \cdots; B\bar{0}; \tau\bar{0}; x_1\bar{2}; \cdots; x_t\bar{0}\rangle,$$

which is then transformed into

$$\sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, -1)|q; B\bar{0}; \cdots; B\bar{1}; \tau\bar{0}; x_1\bar{0}; \cdots; x_t\bar{0}\rangle + \sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, 0)|q; B\bar{0}; \cdots; B\bar{0}; \tau\bar{1}; x_1\bar{0}; \cdots; x_t\bar{0}\rangle + \sum_{(q,\tau)\in Q\times\Sigma} \delta(q_0, x_0, q, \tau, 1)|q; B\bar{0}; \cdots; B\bar{0}; \tau\bar{0}; x_1\bar{1}; \cdots; x_t\bar{0}\rangle.$$

Since the δ-values are exactly the same, one sees that $\mathbb{K}$ will t-simulate M.

In order to get a qubit circuit $\mathbb{K}_n$ based on $\mathcal{G}_u$ that $t(n)$-simulates M, we decompose the gates G_1 and G_2 in $\mathbb{K}$ into gates from $\mathcal{G}_u$. According to Theorem 2.1 G_1 can be decomposed into a finite number of gates from $\mathcal{G}_u$. G_1 is an $(l_0 + 3l)$-qubit gate, so this number does not vary with n. We can not use this decomposition on G_2 because it is an $(l_0+(2t+1)l)$-qubit gate, and hence the number of gates used would increase exponentially in n. However, one can easily see that G_2 can be decomposed into $(2t + 1)$ controlled-not gates N. In the construction of $\mathbb{K}$ we used $t(2t - 1)$ copies of G_1 and t copies of G_2, so the resulting qubit circuit $\mathbb{K}_n$ has size $\mathcal{O}(t^2)$. M is polynomial-time so the QCF $\mathcal{K} = \{\mathbb{K}_n\}_{n\in\mathbb{N}}$ is polynomial-size. From the way we constructed $\mathbb{K}$ and thereby $\mathbb{K}_n$, one sees that the code $c(\mathbb{K}_n)$ is computable by polynomial-time DTM, and hence $\mathcal{K}$ is uniform. Since the number of different gates used to decompose G_1 and G_2 does not vary with n, $\mathcal{K}$ is a finitely-generated QCF. □

3.2 Simulation of quantum circuit families and equivalence

In this section we present the converse result of the previous section, namely that any finitely-generated QCF can be simulated by a polynomial-time QTM. However, only a superficial sketch of the proof will be given, since a detailed one requires a much more extensive treatment of QTMs than has been given in Section 1.

Definition 3.3. Let $\mathbb{K} = (K, \Lambda_1, \Lambda_2, S)$ be an n-input k-qubit circuit. A QTM $M = (Q, \Sigma, \delta)$ is said to *carry out* $\mathbb{K}$, if for any n-bit string x and k-bit string y, we have

$$y \text{ is the output of } M \text{ on input } x \text{ with probability } |\langle y|G(K)|u(x,\mathbb{K})\rangle|^2. \tag{3.1}$$

Let $\mathcal{K} = \{\mathbb{K}_n\}_{n\in\mathbb{N}}$ be QCF. A QTM $M = (Q, \Sigma, \delta)$ is said to *simulate* $\mathcal{K}$, if M carries out $\mathbb{K}_n$ for any $n \in \mathbb{N}$.

Note that if Λ_2, in the above definition, is the whole interval $[1, k]_{\mathbb{Z}}$, i.e., if $\mathbb{K}$ is a k-output k-qubit circuit, then the probability $|\langle y|G(K)|u(x,\mathbb{K})\rangle|^2$ is equal to $\rho^{\mathbb{K}}(y|x)$ - see (2.4).

Theorem 3.4. *Let $\mathcal{K} = \{\mathbb{K}_n\}_{n\in\mathbb{N}}$ be a finitely-generated QCF. Then, there is a polynomial-time QTM $M = (Q, \Sigma, \delta)$ that simulates $\mathcal{K}$.*

Proof. (Sketch) This theorem is essentially the same as Theorem 5.1 in the paper [Nishimura and Ozawa, 2002], which states the same, but for $\mathcal{G}_{\mathcal{R}}$*-uniform* QCFs instead of finitely-generated QCFs. $\mathcal{G}_{\mathcal{R}}$ is a particular finite set of qubit gates, and $\mathcal{G}_{\mathcal{R}}$*-uniform QCFs* are simply uniform QCFs based on $\mathcal{G}_{\mathcal{R}}$. In order to construct the QTM M, one needs the so called programming primitives for QTMs, developed in [Bernstein and Vazirani, 1997]. A proper presentation of these, is beyond the scope of this paper. However, the main idea in the construction is to first compute the code $c(\mathbb{K}_n)$ from the input string 1^n using a polynomial-time DTM. And then to carry out $\mathbb{K}_n$ using the code and QTMs which each carry out the gates that $\mathbb{K}_n$ is based on. The reason why all these different tasks can be composed together into a single QTM M, is do to the fact that $\mathcal{K}$ is based on, only a finite number of different gates. □

The following corollary is obtained directly from Theorem 3.2 and Theorem 3.4.

Corollary 3.5. *The class of languages* **EFPQC** *(resp.* **ZFPQC** *and* **BFPQC***) recognized with certainty (resp. with zero-error and bounded-error) by finitely-generated QCFs coincides with the corresponding complexity class* **EQP** *(resp.* **ZQP** *and* **BQP***) for polynomial-time QTMs.*

4 Concluding remarks

In the previous section we have seen that finitely-generated QCFs are equivalent to polynomial-time QTMs when implementing, not only bounded-error algorithms but, also zero-error and exact algorithms. This suggests that we should base the circuit model of quantum computation on the set of finitely-generated QCFs. In this section we briefly consider some other possibilities.

4.1 Comparison with other models

A type of QCFs called *polynomial-time uniformly generated* were introduced in the paper [Kitaev and Watrous, 2000]. These circuit families can briefly be described as uniform QCFs (in the sense of Definition 2.6) based on a specific set of 5 qubit gates (called the Shor-basis). This type of QCF is used to define quantum computation in the text book [Kitaev et al., 2002][3]. In the bounded-error case, the set of polynomial-time uniformly generated QCFs recognizes the same languages as polynomial-time QTMs do (i.e., **BQP**). But in [Nishimura, 2003] it has been shown that by fixing the set of qubit gates, which the QCFs are based on, one seriously reduces the class of languages which they recognize when implementing exact or zero-error algorithms. For instance, in the exact case, the polynomial-time uniformly generated QCFs only recognize the class **P** (languages recognized by polynomial-time DTMs) - not **EQP**. Thus, a circuit model based on polynomial-time uniformly generated QCFs is only suited for bounded-error computation.

One may also consider a circuit model based on QCFs that are just uniform, i.e., without any limitation on the number of different qubit gates. Again, when implementing bounded-error algorithms the set of uniform QCFs recognizes precisely **BQP** - as it should[4]. But, in the exact and zero-error case, it turns out that they recognize to much. In [Mosca and Zalka, 2003] it has been shown how to construct a uniform QCF which exactly implements the quantum Fourier transform of any order[5]. On the contrary it is proved in [Nishimura and Ozawa, 2005b] that there exists no finitely-generated QCF which implements the quantum Fourier transform of any order without error. Since finitely-generated QCFs and polynomial-time QTMs are equivalent in this case, we see that uniform QCFs are more powerful than they ought to be in the zero-error and exact case.

Finitely-generated QCFs are thus an intermediate of polynomial-time uniformly generated QCFs and uniform QCFs, such that their computational power exactly matches that of polynomial-time QTMs.

[3] See Definition 8.5 and Section 9.4

[4] Theorem 5.2 in [Nishimura and Ozawa, 2002]

[5] See [Nishimura and Ozawa, 2005b] for a definition.

4.2 Conclusion

Nishimura and Ozawa have been able to prove many fundamental results regarding both the Turing machine model and circuit model of quantum computation. This has been due to the strict mathematical formalism they have introduced enabling them to give a thorough definition of concepts that are normaly vaguely formulated.

References

Dorit Aharonov. Quantum computation. In *Annual Reviews of Computational Physics VI*, 1998. URL http://arxiv.org/abs/quant-ph/9812037.

A. Barenco, C. H. Bennett, D. DiVicenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Phsical Review A*, 52:3457–3467, 1995. URL http://arxiv.org/abs/quant-ph/9503016.

Ethan Bernstein and Umesh Vazirani. Quantum complexity theory (preliminary abstract). In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 11–20, 1993.

Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997. URL http://citeseer.ist.psu.edu/bernstein97quantum.html.

Josef Gruska. *Quantum Computing*. McGraw-Hill, 1999.

A. Yu. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. AMS, 2002.

A. Yu. Kitaev and John Watrous. Parallelization, amplification, and exponential time simulation of quantum interactive proof systems. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 608–617, 2000.

Michele Mosca and Christof Zalka. Exact quantum fourier transforms and discrete logarithm algorithms. 2003. URL http://arxiv.org/abs/quant-ph/0301093.

Harumichi Nishimura. Quantum computation with restricted amplitudes. *International Journal of Foundations of Computer Science*, 14(5):853–870, 2003.

Harumichi Nishimura and Masanao Ozawa. Local transition functions of quantum turing machines. *Theoretical Informatics and Application*, 34:379–402, 2000. URL http://arxiv.org/abs/quant-ph/9811069.

Harumichi Nishimura and Masanao Ozawa. Computational complexity of unifom quantum circuit families and quantum turing machines. *Theoretical Computer Science*, 276:147–181, 2002. URL http://arxiv.org/abs/quant-ph/9906095.

Harumichi Nishimura and Masanao Ozawa. Perfect computational equivalence between quantum turing machines and finitely generated uniform quantum circuit families. 2005a. URL http://arxiv.org/abs/quant-ph/0511117.

Harumichi Nishimura and Masanao Ozawa. Uniformity of quantum circuit families for error-free algorithms. *Theoretical Computer Science*, 332:487–496, 2005b.

Masanao Ozawa. Quantum turing machines: Local transition, preparation, measurement, and halting. In *Quantum Communication, Computing and Measurement 2*, pages 233–241, 2000. URL http://arxiv.org/abs/quant-ph/9809038.

Christos H. Papadimitrio. *Computational Complexity*. Addison-Wesley, 1994.

Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th IEEE FOCS*, pages 124–134, 1994. URL http://citeseer.ist.psu.edu/14533.html.

Andrew Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of the 34th IEEE FOCS*, pages 352–361, 1993. URL http://citeseer.ist.psu.edu/yao93quantum.html.