Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Information and Computation 203 (2005) 145–162
Information and Computation
www.elsevier.com/locate/ic

On the computational power of probabilistic and quantum branching program

Farid Ablayev [a,1], Aida Gainutdinova [b,2], Marek Karpinski [c,3], Cristopher Moore [d,*,4], Christopher Pollett [e]

[a] *Department of Theoretical Cybernetics, Kazan State University, Russia*
[b] *Department of Theoretical Cybernetics, Kazan State University, Russia*
[c] *Department of Computer Science, University of Bonn, Germany*
[d] *Computer Science Department, University of New Mexico, Albuquerque and the Santa Fe Institute, USA*
[e] *Department of Computer Science, San Jose State University, USA*

Received 14 February 2004; revised 22 April 2004
Available online 26 September 2005

Abstract

In this paper, we show that one-qubit polynomial time computations are as powerful as NC^1 circuits. More generally, we define syntactic models for quantum and stochastic branching programs of bounded width and prove upper and lower bounds on their power. We show that any NC^1 language can be accepted exactly by a width-2 quantum branching program of polynomial length, in contrast to the classical case where width 5 is necessary unless $NC^1 = ACC$. This separates width-2 quantum programs from width-2 doubly stochastic programs as we show the latter cannot compute the middle bit of multiplication. Finally, we show that bounded-width quantum and stochastic programs can be simulated by classical programs of larger but bounded width, and thus are in NC^1. For read-oncequantum branching programs (QBPs), we give a symmetric Boolean

[*] Corresponding author. Fax: +1 505 982 0565.
*E-mail:*ablayev@ksu.ru (F.Ablayev), aida@ksu.ru (A. Gainutdinova), marek@cs.uni-bonn.de (M. Karpinski), moore@cs.unm.edu (C. Moore), pollett@cs.sjsu.edu (C. Pollet).
[1] Work done in part while visiting Institute of Advanced Study and Max-Planck Institute for Mathematics, supported in part by Russia Fund for Basic Research Grant 03-01-00769.
[2] Supported in part by Russia Fund for Basic Research Grant 03-01-00769.
[3] Supported in part by DFG grants, and IST Grant 14036 (RAND-APX).
[4] Supported by NSF Grants PHY-0200909, CCR-0220070, and EIA-0218563.

0890-5401/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2005.04.003

function which is computable by a read-once QBP with $O(\log n)$ width, but not by a deterministic read-once BP with $o(n)$ width, or by a classical randomized read-once BP with $o(n)$ width which is "stable" in the sense that its transitions depend on the value of the queried variable but do not vary from step to step. Finally, we present a general lower bound on the width of read-once QBPs, showing that our $O(\log n)$ upper bound for this symmetric function is almost tight.

1. Preliminaries

Interest in quantum computation has steadily increased since Shor's discovery of a polynomial time quantum algorithm for factoring [19]. A number of models of quantum computation have been considered, including quantum versions of Turing machines, simple automata, circuits, and decision trees. The goal of much of this research has been to understand in what ways quantum algorithms do and do not offer a speed-up over the classical case, and to understand what classical techniques for proving upper and lower complexity bounds transfer to the quantum setting.

Branching programs have proven useful in a variety of domains, such as hardware verification, model checking, and other CAD applications; see, for example, the book by Wegener [23]. In addition, branching programs are a convenient model for non-uniform computation with varying restrictions. Even oblivious branching programs of constant width—the non-uniform equivalent of finite-state automata—are surprisingly powerful. Indeed, Barrington [6] showed that branching programs of width 5 are already as powerful as circuits of logarithmic depth.

Moreover, branching programs are a very natural model for comparing the power of quantum computation with classical computation, both deterministic and randomized. Randomized branching programs (with restrictions on variables testing) have been intensively investigated since 1996 [3]. In [3], the first example of a function is presented which is exponentially cheaper for a randomized OBDD than for a deterministic OBDD; see [23] for more information. Recently, several models of quantum branching programs have been proposed [1,2,5,14,22].

In this paper, we define and consider models of stochastic and quantum branching programs. For this syntactic model, we present several results for quantum branching programs of bounded width [1]. We show that width-2 syntactic quantum programs are more powerful than width-2 doubly stochastic programs, and are as strong as deterministic branching programs of width 5. Specifically, we show that polynomial-length, width-2 syntactic quantum branching programs can recognize any language in NC^1 exactly. Note that such programs are equivalent to a non-uniform automata whose only storage device is a single qubit!

On the other hand, we show that polynomial-length, width-2 doubly stochastic programs cannot compute the middle bit of the multiplication function $MULT$. This is a consequence of the more general fact (proved in [4]) that $MULT$ is exponentially hard for randomized OBDDs and that 2×2 doubly stochastic matrixes commute. Note that Yao [24] showed that width-2 deterministic programs require superpolynomial length to compute the majority function.

Next, we show that bounded-error syntactic quantum and stochastic programs can be simulated by deterministic programs of the same length and larger, but still bounded, width. Therefore, the class of languages recognized by these programs coincides with (non-uniform) NC^1. This also implies that, for bounded-width quantum programs, exact acceptance is just as strong as acceptance with bounded error.

To give some flavour of what our syntactic model is, consider the usual computation of a branching program. When we query a variable, we do one of two actions depending on its values. If we query the variable again we expect to see the same value. An *inconsistent* state is one that is reached by querying a variable more than once and seeing different values at different times. In analogy with syntactic read-k-times classical branching programs [8], our syntactic quantum and stochastic programs have a bounded-error promise for their acceptance probabilities that holds even for inconsistent final states. This restriction is admittedly somewhat artificial, but it allows us to put upper bounds on the classical complexity of the languages recognized by such branching programs using simple metric arguments. Proving similar upper bounds without this syntactic restriction is an interesting open question.

We use the techniques of our result to show that (syntactic) polynomial-length width-2 stochastic programs that accept with probability $1/2 + \epsilon$ cannot compute the majority function if $\epsilon > 1/4$. In addition, we show that polynomial-length stochastic programs with width 2 and $\epsilon > 1/8$, width 3 and $\epsilon > 1/3$, or width 4 and $\epsilon > 3/8$ can only recognize languages in ACC.

Finally, we investigate the properties of read-once quantum branching programs, which were defined in [2]. We show that, just as in the classical deterministic case, ordered read-once permutation branching programs of exponential width (and hence QBPs) can recognize arbitrary languages. Next we exhibit a symmetric Boolean function which is computable by a read-once QBP with $O(\log n)$ width, but which requires $\Omega(n)$ width for deterministic BPs, and for randomized classical BPs with a certain restriction on their transitions. Finally, we present a general lower bound on the width of read-once QBPs, and show that our $O(\log n)$ upper bound for this symmetric function is almost tight.

2. Branching programs

We begin by discussing the classical model of branching programs and then generalize it to the quantum setting. A good source of information on branching programs is Wegener's book [23]; for an introduction to quantum computation see Nielsen and Chuang [15].

Definition 1. A *branching program* is a finite directed acyclic graph which will be used to recognize some subset of $\{0,1\}^n$. Each node (except for a sink node) is labeled with an integer $1 \leqslant i \leqslant n$ and has two outgoing arrows labeled 0 and 1. This pair of edges corresponds to querying the ith bit x_i of the input, and making a transition along one outgoing edge or the other depending on the value of x_i. There is a single source node, called the start node, and a subset Accept of the sink nodes corresponding to accepting nodes. An input x is *accepted* if and only if it induces a chain of transitions leading to a node in Accept, and the set of such inputs is the language accepted by the program. A branching program is *oblivious* if the nodes can be partitioned into levels $V_1, \dots, V_\ell$ and a level $V_{\ell+1}$ such that the nodes in $V_{\ell+1}$ are the sink nodes, nodes in each level V_j with $j \leqslant \ell$ have outgoing edges only to nodes in the next level V_{j+1}, and all nodes in a given level V_j query the same bit x_{i_j} of the input. Such a program is said to have *length* ℓ, and *width* k if each level has at most k nodes.

Note that $V_{\ell+1}$ may include inconsistent final nodes, i.e., those reached by paths where a variable takes different values in multiple queries.

Oblivious branching programs have an elegant algebraic definition. Recall that a *monoid* is a set with an associative binary operation $\cdot$ and an identity 1 such that $1 \cdot a = a \cdot 1 = a$ for all a.

Definition 2. Let M be a monoid and $S \subset M$ an accepting set. Let x_i for $1 \leqslant i \leqslant n$ be a set of Boolean variables. A *branching program over M of length ℓ* is a string of ℓ *instructions*; the jth instruction is a triple $(i_j, a_j, b_j) \in \{1, \ldots, n\} \times M \times M$, which we interpret as a_j, if $x_{i_j} = 0$; and as b_j, if $x_{i_j} = 1$. Given an input x, the *yield* $Y(x)$ of the program is the product in M of all its instructions. We say that the input x is *accepted* if $Y(x) \in S$, and the set of such inputs is the language recognized by the program.

Such programs are often called *non-uniform deterministic finite automata* (NUDFAs). A computation in a deterministic finite automaton can be thought of as taking a product in its syntactic monoid; in a NUDFA we generalize this by allowing the same variable to be queried many times, and allowing "true" and "false" to be mapped into a different pair of monoid elements in each query.

A common monoid is T_k, the set of functions from a set of k objects into itself. Then the program makes transitions among k nodes, and we can equivalently define oblivious, width-k branching programs by choosing an initial node and a set of accepting final nodes, with k nodes in each level V_j.

Definition 3. An *oblivious width-k branching program* is a branching program over T_k, where the accepting set $S \subset T_k$ consists of those elements of T_k that map an initial node $s \in \{1, \ldots, k\}$ to a final node $t \in$ Accept for some subset Accept $\subset \{1, \ldots, k\}$.

3. Bounded width branching programs

We define language classes recognized by (non-uniform) families of bounded-width branching programs whose length increases polynomially with n:

Definition 4. k-BWBP is the class of languages recognized by polynomial-length branching programs of width k, and BWBP $= \cup_k$ k-BWBP.

Recall that a *group* is a monoid where every element has an inverse, and a group is *Abelian* if $ab = ba$ for all a, b. A subgroup $H \subseteq G$ is *normal* if the left and right cosets coincide, $aH = Ha$ for all $a \in G$. A group is *simple* if it has no normal subgroups other than itself and $\{1\}$.

Barrington [6] studied branching programs over the permutation group on k objects $S_k \subset T_k$; such programs are called *permutation programs*. He showed that polynomial-length programs over S_5, and therefore width-5 branching programs, can recognize any language in NC^1, the class of languages recognizable by Boolean circuits of polynomial width and logarithmic depth [16]. Indeed, this follows from an earlier algebraic result by Maurer and Rhodes [11]. The version of Barrington's result that we will use is:

Theorem 1 ([6,13]). *Let G be a non-Abelian simple group, and let $a \neq 1$ be any non-identity element. Then any language L in NC^1 can be recognized by a family of polynomial-length branching programs over G such that their yield is $Y(x) = a$ if $x \in L$ and 1 otherwise.*

Since the smallest non-Abelian simple group is $A_5 \subset S_5$, the group of even permutations of 5 objects, and since we can choose a permutation a that maps some initial node s to some other final node t, width 5 suffices. Conversely, we can model a width-k branching program as a Boolean product of ℓ transition matrices of dimension k, and a simple divide-and-conquer algorithm allows us to calculate this product in $O(\log \ell)$ depth. Thus, BWBP $\subset$ NC^1, so we have

$$\text{5-BWBP} = \text{BWBP} = \text{NC}^1 \; .$$

The definition of a *linear branching program* is based on the oblivious model. This is a generalization of the definition of quantum branching program presented in [2]. Deterministic, stochastic, and quantum oblivious branching programs are particular cases of linear branching programs. Let $\mathbf{V}^k$ be a k-dimensional vector space; here k is the width of the program, and the vectors μ we will consider will be probability distributions or quantum superpositions over the k states. We use $|\mu\rangle$ and $\langle\mu|$ to denote column vectors and row vectors, respectively, from $\mathbf{V}^k$, and $\langle\mu_1 \mid \mu_2\rangle$ denotes the complex inner product. This "bra-ket" notation was invented by Dirac, and is now widely used in quantum mechanics. We write μ when it is not important whether it is in column or row form.

Definition 5 (*Linear branching program*). A Linear Branching Program P of width k and length ℓ (a $(k,\ell) - LBP$) over $\mathbf{V}^k$ is defined as

$$\mathcal{P} = \langle T, |\mu_0\rangle, \text{Accept}\rangle,$$

where T is a sequence (of length ℓ) of k-dimensional linear transformations of the vector space $\mathbf{V}^k$:

$$T = \left(i_j, M_j(0), M_j(1)\right)_{j=1}^{\ell} \; .$$

Vectors $|\mu\rangle \in \mathbf{V}^k$ are called states (state vectors) of P, $|\mu_0\rangle \in \mathbf{V}^k$ is the initial state of P, and Accept $\subseteq \{1,\ldots,k\}$ is the accepting set.

We define a computation on P with an input $x = x_1,\ldots,x_n \in \{0,1\}^n$ as follows:

1. A computation of P starts from the initial state $|\mu_0\rangle$;
2. The jth step of P queries a variable x_{i_j}, and applies the transition matrix $M_j = M_j(x_{i_j})$ to the current state μ to obtain the state $\mu' = M_j(x_{i_j})\mu$;
3. The final state (i.e., the state after step $\ell + 1$) is

$$|\mu(x)\rangle = \prod_{j=\ell}^{1} M_j(x_{i_j})|\mu_0\rangle.$$

Recall that in a read-once program, each variable is queried at most once during the computation. For such a program, at each step j, an arbitrary transformation $M_j(x_{i_j}) \in \{M_j(0), M_j(1)\}$ might be applied since the i_j are all distinct. However, if a variable x_i is queried more than once during the computation, we must see it take the same value in every query, and this constrains the transformations M_j to be consistent across the set of j such that $i_j = i$. This observation motivates the following definition:

Definition 6. We call a state μ of a branching program *consistent* if there exists an input x that induces a chain of transitions leading to the μ from the initial state μ_0. Otherwise, we call μ an inconsistent state.

For each $j \in \{1 \ldots, \ell + 1\}$ we let $\mathcal{V}_j$ denote the set of states μ (both consistent and inconsistent) of the branching program. For stochastic and quantum programs defined below, states in $\mathcal{V}_j$ will be vectors (probability distributions or vectors of ℓ_2-norm 1, respectively) over the basis set V_j.

Now oblivious deterministic, stochastic, and quantum branching programs can be presented as follows:

Deterministic branching programs. A *deterministic* branching program is a linear branching program over a vector space $\mathbb{R}^k$. A state μ of such a program is a Boolean vector with exactly one 1. The matrices M_j correspond to elements of T_k, and so have exactly one 1 in each column. For branching programs over groups this is true of the rows as well; in which case, the M_j are permutation matrices.

Stochastic branching programs. The concept of deterministic branching programs naturally generalizes to stochastic branching programs, by letting μ be a probability distribution, and by letting the M_j be *stochastic* matrices, i.e., matrices with non-negative entries where each column sums to 1. If the M_j also have rows that sum to 1, then one has *doubly stochastic* matrices, and the corresponding doubly stochastic branching programs. Recall that, by Birkhoff's Theorem [21], doubly stochastic matrices are convex combinations of permutation matrices.

In the deterministic and stochastic cases, for a final state vector (consistent or inconsistent) $\mu \in \mathcal{V}_{\ell+1}$, we define

$$\Pr(\mu) = \sum_{i \in \text{Accept}} \langle i \mid \mu \rangle = \left\| \Pi_{\text{Accept}} \mu \right\|_1, \tag{1}$$

and we define the probability of acceptance as $\Pr(x) = \Pr(\mu(x))$. Here, $|i\rangle$ is the basis vector with support on the node i, and Π_{Accept} is a projection operator on the *accepting subspace* $\text{span}\{|i\rangle : i \in \text{Accept}\}$.

Quantum branching programs. We define a *quantum* branching program as a linear branching program over a Hilbert space $\mathbb{C}^k$. The μ for such a program are complex state vectors with $\|\mu\|_2 = 1$, and the M_j are complex-valued unitary matrices. For a final state vector (consistent or inconsistent) $\mu \in \mathcal{V}_{\ell+1}$, we define $\Pr(\mu)$ as

$$\Pr(\mu) = \sum_{i \in \text{Accept}} |\langle i \mid \mu \rangle|^2 = \left\| \Pi_{\text{Accept}} \mu \right\|_2^2, \tag{2}$$

and the probability of acceptance as $\Pr(x) = \Pr(\mu(x))$; that is, the probability that if we measure $\mu(x)$, we will observe it in the accepting subspace. Note that this is a "measure-once" model analogous to the model of quantum finite automata in [12], in which the system envolves unitarily except for a single measurement at the end. We could also allow multiple measurements during the computation, by representing the state as a density matrix ρ, and by making the M_j superoperators, but we do not consider this here.

We can define recognition in several ways for the quantum case. We say that a language L is accepted *with unbounded error* if $\Pr(x) > 1/2$ if $x \in L$ and $\Pr(x) \leqslant 1/2$ if $x \notin L$. We say that a language L is accepted *with bounded error* if there is some $\epsilon > 0$ such that $\Pr(x) \geqslant 1/2 + \epsilon$ if $x \in L$ and

$\Pr(x) \leqslant 1/2 - \epsilon$ if $x \notin L$. For the case $\epsilon = 1/2$, we have that $\Pr(x) = 1$, if $x \in L$; and $\Pr(x) = 0$, if $x \notin L$. So as we did in the deterministic case, we say that L is accepted *exactly*.

4. Syntactic stochastic and quantum programs

For unbounded and bounded error stochastic and quantum branching programs we define two subsets $\mathcal{A}$ and $\mathcal{R}$ of the set $\mathcal{V}_{\ell+1}$ of final state vectors (consistent and inconsistent) as follows: For unbounded error programs, we define

$$\mathcal{A} = \{\mu \in \mathcal{V}_{\ell+1} : \Pr(\mu) > 1/2\} \quad \text{and} \quad \mathcal{R} = \{\mu \in \mathcal{V}_{\ell+1} : \Pr(\mu) \leqslant 1/2\};$$

and for bounded error programs, we define

$$\mathcal{A} = \{\mu \in \mathcal{V}_{\ell+1} : \Pr(\mu) \geqslant 1/2 + \epsilon\} \quad \text{and} \quad \mathcal{R} = \{\mu \in \mathcal{V}_{\ell+1} : \Pr(\mu) \leqslant 1/2 - \epsilon\}.$$

We call $\mathcal{A}$ and $\mathcal{R}$ the *accepting* and *rejecting* sets, respectively.

Recall that $\mathcal{V}_{\ell+1}$ includes the final states reachable by all possible paths, both consistent and inconsistent. Then:

Definition 7. We call a stochastic or a quantum branching program *syntactic* if its accepting and rejecting set of state vectors form a partition of the set of final states, i.e., if $\mathcal{V}_{\ell+1} = \mathcal{A} \cup \mathcal{R}$.

Note that without the syntactic restriction, it might happen that $\mathcal{V}_{\ell+1} \neq \mathcal{A} \cup \mathcal{R}$, and that some inconsistent final state vector $\mu \in \mathcal{V}_{\ell+1}$ has the property that $1/2 - \epsilon < \Pr(\mu) < 1/2 + \epsilon$.

We denote by $B\cdot$ the language classes recognized by standard (non-syntactic) programs with bounded error and denote by $E\cdot$ those recognized exactly. The notations SBP and QBP stand for stochastic and quantum branching programs, respectively. We denote the classes of languages recognized by width-k stochastic and quantum programs of polynomial length as k-BSBP, k-BQBP, and k-EQBP. Note that we remove "BW" to avoid acronym overload. We write BSBP for $\cup_k k$-BSBP and define BQBP and EQBP similarly. Clearly we have

$$\text{BWBP} \subseteq \text{EQBP} \subseteq \text{BQBP}$$

and

$$\text{BWBP} \subseteq \text{BSBP}$$

but in principle k-BSBP could be incomparable with k-EQBP or k-BQBP.

5. Width-2 stochastic and quantum programs

In this section, we show that width-2 syntactic quantum programs with exact acceptance contain NC^1, and also that this class of programs is stronger than width-2 syntactic doubly stochastic programs.

Lemma 1. *Any width-*2 *doubly stochastic program on n variables is equivalent to one which queries each variable once and in the order* $x_1, x_2, \ldots, x_n$.

Proof. Any 2×2 stochastic matrix can be written as $\begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}$ for some $p \in [0,1]$. It is easy to verify that matrices of this form commute. Hence, if we have a product of such matrices $\prod_{j=n}^{1} M_j(x_{i_j})$ we can rewrite it so that we first take the product of all the matrices that depend on x_1, then those that depend on x_2, and so on. To finish the proof, we note that products of doubly stochastic matrices are again doubly stochastic, so we can use a single doubly stochastic matrix for the product of all the matrices that depend on a given x_i. □

The above lemma shows we can convert any width-2 doubly stochastic program into one which is read-once and with a fixed variable ordering. i.e., a randomized ordered binary decision diagram (OBDD). Hence, for width-2 programs, the syntactic and non-syntactic models are equivalent.

Next we note that stochastic programs are stronger than permutation programs for width-2. It is easy to see that any program over $\mathbb{Z}_2$ simply yields the parity of some subset of the x_i. The AND_n function, which accepts only the input with $x_i = 1$ for all i, is not of this form, and so this function cannot be recognized by a width-2 permutation program. However, it can easily be recognized by a stochastic program P with bounded error which queries each variable once as follows: for $i < n$ it maps $x_i = 1$ and 0 to the identity $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and the matrix $\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}$, respectively, and for x_n it maps 1 and 0 to $\begin{pmatrix} 3/4 & 0 \\ 1/4 & 1 \end{pmatrix}$ and $\begin{pmatrix} 3/8 & 3/8 \\ 5/8 & 5/8 \end{pmatrix}$, respectively. Taking the first node to be both the initial and final node, P accepts with probability 3/4 if $x_i = 1$ for all i and 3/8 otherwise. Note that except for one matrix this is in fact a doubly stochastic program; if we had treated the variable x_n in the same fashion as the other variables we would have gotten a syntactic doubly stochastic program accepting AND_n with one-sided error.

Despite being stronger than their permutation counterparts, the next result shows width-2 doubly stochastic branching programs are not that strong. Let $MULT_k^n$ be the Boolean function which computes the k'th bit of the product of two n-bit integers. Define $MULT^n$ to be $MULT_{n-1}^n$, i.e., the middle bit of the product. We will argue that any width-2 stochastic program that calculates this function (i.e., that recognizes the set of inputs for which $MULT^n = 1$) requires exponential width.

Ablayev and Karpinski [4] investigated randomized OBDDs, i.e., those which accept with bounded error.

Theorem 2 ([4]). *Any randomized* OBDD *that bounded error computes* $MULT^n$ *has width at least* $2^{\Omega(n/\log n)}$.

So by Lemma 1 we have immediately:

Corollary 1. $MULT^n$ *cannot be computed by a width-*2 *doubly stochastic program.*

While width-2 doubly stochastic programs are quite weak, the next result shows that width-2 *quantum* programs are surprisingly strong. Note that a width-2 quantum program has a state space equivalent to a single qubit, such as a single spin-1/2 particle.

Theorem 3. NC^1 *is contained in syntactic* 2-EQBP.

Proof. We show that Barrington's simulation of NC[1] can be carried out in $U(2)$, the group of 2×2 unitary matrices. Recall that $SU(2)$, the group of 2×2 unitary matrices with determinant 1, is an algebraic double cover of $SO(3)$, the group of three-dimensional rotations, and $SO(3)$ has a subgroup isomorphic to A_5, namely the group of rotations of the icosahedron.

To make this explicit, we recall a well-known 2-to-1 mapping from $SU(2)$ to $SO(3)$. Consider a qubit $a|0\rangle + b|1\rangle$ with $|a|^2 + |b|^2 = 1$; we can make a real by multiplying by an overall phase. The *Bloch sphere* representation (see e.g. [15]) views this state as the point on the unit sphere with latitude θ and longitude ϕ , i.e., $(\cos\phi\cos\theta, \sin\phi\cos\theta, \sin\theta)$, where $a = \cos\theta/2$ and $b = e^{i\phi}\sin\theta/2$.

Given this representation, an element of $SU(2)$ is equivalent to some rotation of the unit sphere. Recall the *Pauli matrices*

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & \mathrm{i} \\ -\mathrm{i} & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Then we can rotate an angle α around the x, y or z axes with the following operators:

$$\begin{aligned} R_x(\alpha) &= \mathrm{e}^{\mathrm{i}(\alpha/2)\sigma_x} = \begin{pmatrix} \cos\alpha/2 & -\mathrm{i}\sin\alpha/2 \\ -\mathrm{i}\sin\alpha/2 & \cos\alpha/2 \end{pmatrix}, \\ R_y(\alpha) &= \mathrm{e}^{\mathrm{i}(\alpha/2)\sigma_y} = \begin{pmatrix} \cos\alpha/2 & -\sin\alpha/2 \\ \sin\alpha/2 & \cos\alpha/2 \end{pmatrix}, \text{ and} \\ R_z(\alpha) &= \mathrm{e}^{\mathrm{i}(\alpha/2)\sigma_z} = \begin{pmatrix} \mathrm{e}^{-\mathrm{i}\alpha/2} & 0 \\ 0 & \mathrm{e}^{\mathrm{i}\alpha/2} \end{pmatrix}. \end{aligned}$$

This makes $SU(2)$ a *double cover* of $SO(3)$, where each element of $SO(3)$ corresponds to two elements $\pm U$ in $SU(2)$. (Note that angles get halved by this mapping.) Therefore, $SU(2)$ has a subgroup which is a double cover of A_5. One way to generate this subgroup is with $2\pi/5$ rotations a and b around two adjacent vertices of an icosahedron. Since two such vertices are an angle $\tan^{-1} 2$ apart, if one lies on the z axis and the other lies in the x–z plane, we have

$$\begin{aligned} a &= R_z(2\pi/5) = \begin{pmatrix} \mathrm{e}^{\mathrm{i}\pi/5} & 0 \\ 0 & \mathrm{e}^{-\mathrm{i}\pi/5} \end{pmatrix}, \\ b &= R_y(\tan^{-1} 2) \cdot a \cdot R_y(-\tan^{-1} 2) \\ &= \frac{1}{\sqrt{5}} \begin{pmatrix} \mathrm{e}^{\mathrm{i}\pi/5}\tau + \mathrm{e}^{-\mathrm{i}\pi/5}\tau^{-1} & -2\mathrm{i}\sin\pi/5 \\ -2\mathrm{i}\sin\pi/5 & \mathrm{e}^{-\mathrm{i}\pi/5}\tau + \mathrm{e}^{\mathrm{i}\pi/5}\tau^{-1} \end{pmatrix}, \end{aligned}$$

where $\tau = (1 + \sqrt{5})/2$ is the golden ratio. Now consider the group element $c = a \cdot b \cdot a$; this rotates the icosahedron by π around the midpoint of the edge connecting these two vertices. In $SU(2)$, this maps each of the eigenvectors of σ_y to the other times an overall phase. These eigenvectors are

$$e_+ = \frac{|0\rangle + \mathrm{i}|1\rangle}{\sqrt{2}}, \quad e_- = \frac{|0\rangle - \mathrm{i}|1\rangle}{\sqrt{2}}$$

so we have

$$|\langle e_+| \pm c|e_-\rangle|^2 = 1$$

while, since they are orthogonal,

$$|\langle e_+|1|e_-\rangle|^2 = 0.$$

Now, Theorem 1 tells us that for any language L in NC^1 we can construct a polynomial-length program over A_5 that yields the element equivalent to c if the input is in the language and 1 otherwise. Using the embedding of A_5 in $SO(3)$, and then lifting to $SU(2)$, gives a program which yields $\pm c$ or 1. If we take the initial state to be $\mu_0 = e_-$ and the accepting subspace to be that spanned by e_+, this program accepts L exactly. □

6. Deterministic simulations of syntactic stochastic and quantum branching programs

In this section, we give general results on simulating syntactic stochastic and quantum programs with deterministic ones. Specifically, Theorem 4 shows that syntactic stochastic and quantum programs that accept with bounded error can be simulated by deterministic programs of the same length and larger (but still bounded) width. Below we use this to place upper bounds on the computational power of stochastic programs with various widths and error thresholds.

Theorem 4. *Let P be a syntactic stochastic or quantum branching program of width k and length ℓ that recognizes a language L with probability $1/2 + \epsilon$. Then, there exists a deterministic branching program P' of width k' and length ℓ that recognizes L, where*

$$k' \leqslant \left(\frac{1}{\epsilon}\right)^{k-1} \tag{3}$$

if P is stochastic, and

$$k' \leqslant \left(\frac{2}{\epsilon}\right)^{2k} \tag{4}$$

if P is quantum.

Proof. Our proof is based on arguments of Rabin [17] and Kondacs and Watrous [10]. For each step of the program, we define an equivalence relation on state vectors, where two state vectors are equivalent if they lead to the same outcome (acceptance or rejection). Since P recognizes L with bounded error, inequivalent states must be bounded away from each other, and since the state space is compact the number of equivalence classes is finite. These equivalence classes then become the states of our deterministic program P'.

First, we construct an ℓ-length, 2^ℓ-width oblivious deterministic branching program P'' in a form of complete binary tree. Nodes of levels $V_1'', \ldots, V_{\ell+1}''$ of P'' are labeled by state vectors μ (consistent and non-consistent) of P as follows: V_1'' contains unique node labeled by the initial state μ_0. For all $1 \leqslant j \leqslant \ell$ each node in V_j'' labeled by $\mu \in \mathcal{V}_j''$ has two outgoing edges to nodes in V_{j+1}'' labeled by $M_j(0)\mu, M_j(1)\mu$. The syntactic property allows us to partition nodes in $V_{\ell+1}''$ into accepting and rejecting subsets Acc and Rej according to equations 1 and 2. Clearly that such deterministic program P'' recognizes L deterministically.

Now we inductively define an equivalence relation $\equiv_j$ on each level $\mathcal{V}_j$ of program P. First, we let $\mathcal{A}$ and $\mathcal{R}$ be the equivalence classes of $\equiv_{\ell+1}$. Then, for each $1 \leqslant j \leqslant \ell$, define

$$\mu \equiv_j \mu' \;\Leftrightarrow\; M_j(0)\mu \equiv_{j+1} M_j(0)\mu' \text{ and } M_j(1)\mu \equiv_{j+1} M_j(1)\mu'.$$

The equivalence relation $\equiv_j$ has the following properties:

1. $\mathcal{V}_{\ell+1} = \mathcal{A} \cup \mathcal{R}$ since P is syntactical program;
2. equivalence classes have *transitive closure* property under transformation;
3. since stochastic and unitary transformations do not increase the distance between states, the distance between two arbitrary different $\equiv_j$ classes are bounded away from each other by constant.

The equivalence relation $\equiv_j$ on $\mathcal{V}_j$ of program P determines the equivalence relation on V''_j of program P''. We keep the same notation $\equiv_j$ for such this equivalence relation on V''_j.

We now define P' as the deterministic branching program whose nodes V'_j, for each level j, are the equivalence classes of $\equiv_j$ of program P'', and whose accepting subset is the singleton $\{\mathcal{A}\}$. By properties (1–3) above program P' is well defined and recognizes L deterministically; it just remains to show that the number of equivalence classes for each j is bounded.

First, we show that two inequivalent state vectors in $\mathcal{V}_j$ must be far apart, using the following standard argument [17,10]. □

Lemma 2. *Suppose $\mu, \mu' \in \mathcal{V}_j$ and $\mu \not\equiv_j \mu'$. Then*

$$\|\mu - \mu'\|_1 \geqslant 4\epsilon$$

if P is stochastic, and

$$\|\mu - \mu'\|_2 \geqslant 2\epsilon$$

if P is quantum.

Proof. Since stochastic and unitary matrices both preserve or decrease the appropriate norm, it suffices to show this for the last step. Therefore, suppose that $j = \ell + 1$, $\mu \in \mathcal{A}$ and $\mu' \in \mathcal{R}$. We can decompose both vectors, μ, μ', into their components inside the accepting subspace and into their components inside the subspace transverse to the accepting subspace. That is, we can write $\mu = \mu_A + \mu_R$ where $\mu_A = \Pi_{\text{Accept}}\mu$ and $\mu_R = (1 - \Pi_{\text{Accept}})\mu$, and similarly write $\mu' = \mu'_A + \mu'_R$. In the stochastic case, $\|\mu_A\|_1 \geqslant 1/2 + \epsilon$ and $\|\mu'_A\|_1 \leqslant 1/2 - \epsilon$, and so

$$\begin{aligned}\|\mu - \mu'\|_1 &= \|\mu_A - \mu'_A\|_1 + \|\mu_R - \mu'_R\|_1 \\ &\geqslant 2\,[(1/2 + \epsilon) - (1/2 - \epsilon)] \\ &= 4\epsilon\ .\end{aligned}$$

In the quantum case, $\|\mu_A\|_2 \geqslant \sqrt{1/2+\epsilon}$ and $\|\mu'_A\|_2 \leqslant \sqrt{1/2-\epsilon}$, so

$$\begin{aligned}\|\mu-\mu'\|_2^2 &= \|\mu_A-\mu'_A\|_2^2 + \|\mu_R-\mu'_R\|_2^2 \\ &\geqslant 2\left[\sqrt{1/2+\epsilon}-\sqrt{1/2-\epsilon}\right]^2 \\ &= 2\left(1-\sqrt{1-4\epsilon^2}\right) \\ &\geqslant 4\epsilon^2\end{aligned}$$

so $\|\mu-\mu'\|_2 \geqslant 2\epsilon$. □

It follows that the width k' of P' is at most the largest number of balls of radius 2ϵ or ϵ (in the stochastic and quantum case, respectively) one can fit inside the state space. In the stochastic case, the state space is a $(k-1)$-dimensional simplex. Its L_1-diameter is 2, so each ball of radius 2ϵ covers a fraction at least $(1/\epsilon)^{k-1}$ of its volume, yielding (3). This bound is crude in that it assumes that the center of each ball is at a corner of the simplex; balls whose center are in the interior of the simplex cover up to 2^{k-1} times as much volume. In particular, if $k=2$ then $k' \leqslant 1+1/(2\epsilon)$.

In the quantum case, the state space is isomorphic to the surface of the $2k$-dimensional sphere of radius 1. The crude bound of (4) comes from noticing that this sphere, and the balls of radius ϵ whose centers lie on its surface, are all contained in a $2k$-dimensional ball of radius 2.

Theorem 4 shows that bounded-error syntactic stochastic and quantum programs of constant width can be simulated by deterministic programs of constant (though, exponentially larger) width, and are therefore contained in NC^1. Conversely, we showed in Theorem 3 that NC^1 is contained in width-2 syntactic quantum programs. Therefore, the following classes all coincide with NC^1:

Corollary 2. *For syntactic programs,*

$$\text{2-EQBP} = \text{2-BQBP} = \text{EQBP} = \text{BQBP} = \text{BSBP} = \text{BWBP} = \mathrm{NC}^1 \ .$$

Of all the program classes discussed in this paper, the only ones *not* included in this collapse are stochastic programs of width less than 5. Theorem 4 allows us to place upper bounds on their computational abilities if their error margins are sufficiently large. For instance, since Yao [24] showed that width-2 deterministic programs require superpolynomial length to compute the majority function, we have

Corollary 3. *For the syntactic case, width-*2 *stochastic branching programs of polynomial length cannot recognize the majority function with probability* $1/2+\epsilon$ *if* $\epsilon > 1/4$.

Similarly, recall that $\mathrm{ACC} = \cup_p \mathrm{ACC}[p]$ where $\mathrm{ACC}[p]$ is the class of languages recognizable by constant-depth circuits with AND, OR, and mod-p counting gates of arbitrary fan-in. It is known that $\mathrm{ACC}[p] \subsetneq \mathrm{NC}^1$ for prime p [18,20], and strongly believed, but not known, that $\mathrm{ACC} \subsetneq \mathrm{NC}^1$. Since its is known [7] that deterministic programs of width less than 5 recognize languages in ACC[6], we have

Corollary 4. *Suppose L is recognized with probability* $1/2+\epsilon$ *by a width-k stochastic syntactic branching program of polynomial length. If* $k=2$ *and* $\epsilon > 1/8$, *or* $k=3$ *and* $\epsilon > 1/3$, *or* $k=4$ *and* $\epsilon > 3/8$, *then* $L \in \mathrm{ACC}$.

Proof. For each k, we consider the problem of how small ϵ has to be to fit 5 points into the $(k-1)$-dimensional simplex with an L_1 distance 4ϵ between them. While these values of ϵ are smaller that those given by (3), they follow easily from assuming without loss of generality that k of the points lie on the simplex's corners. □

However, we conjecture that stochasticity does not greatly increase the power of bounded-width branching programs, in the following sense:

Conjecture 1. *If L is recognized with bounded error by a stochastic branching program of width less than* 5, *then* $L \in \mathrm{ACC}$.

7. Read-once branching programs

In this section, we investigate the computational power of *read-once* branching programs, i.e., those in which each variable x_i is queried only once during a computation. In particular, a read-once branching program on n variables has length n. Read-once programs have been well-studied and are more commonly called Ordered Binary Decision Diagrams (OBDDs) [23]. We define quantum and stochastic OBDDs using the definitions of acceptance given above.

First, we note that quantum OBDDs of exponential width can compute arbitrary Boolean functions. This is simply because quantum branching programs include permutation programs, and it is easy to see that a permutation program with width 2^n can simply read the input and devote a different final state to every possible input.

To show that quantum OBDDs are more powerful than classical ones, we consider the symmetric Boolean function MOD_p defined as follows: For an input $x = x_1 \dots x_n \in \{0,1\}^n$, the function $MOD_p(x) = 1$ if and only if the number of ones in x is divisible by p.

Theorem 5. *Let $p < n/2$. Then MOD_p can be computed by a read-once quantum branching program of width $O(\log p)$ with one-sided error $\epsilon \geqslant 0$.*

The proof of this theorem will be presented in the subsection below. In contrast to this result, we have that any deterministic OBDD for MOD_p has $\Omega(p)$ width. This follows from the fact that any deterministic OBDD for MOD_p must keep for each input sequence the number of ones (by mod p) in each level of computation.

7.1. Proof of Theorem 5

We will start by giving a quantum branching program P of width $O(\log p)$ that accepts inputs $x \in MOD_p^{-1}(1)$ with probability 1 and rejects inputs $x \in MOD_p^{-1}(0)$ with probability at least 1/8. We will then apply standard techniques to reduce the error to an arbitrarily small ϵ.

The program P is defined using width-2 programs P^k, for $k \in \{1, \dots, p-1\}$. We will construct P by selecting a good set (of cardinality $t = \lceil 16 \ln p \rceil$) of these P^k's and running them in superposition.

For a given k, P^k has as its start vector $|\psi_0^k\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and has transition matrices $T^k = (\langle i, U^k(0), U^k(1)\rangle)_{i=1}^n$ where

$$U^k(0) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, U^k(1) = \begin{pmatrix} \cos(2\pi k/p) & -\sin(2\pi k/p) \\ \sin(2\pi k/p) & \cos(2\pi k/p) \end{pmatrix}.$$

The accepting set of P^k is $\{1\}$.

Let $l(x)$ the number of 1's in the sequence x, i.e., $l(x) = \sum_{i=1}^{n} x_i$. Then:

Lemma 3. *After reading an input $x = x_1 \ldots x_n$ the state vector of P^k is*

$$|\psi\rangle = (\cos\theta_k)|1\rangle + (\sin\theta_k)|2\rangle$$

where $\theta_k = 2\pi k(l(x) \bmod p)/p$.

Proof. This follows from the definition of P^k. □

If the $l(x)$ is a multiple of p, then θ_k is a multiple of 2π for all k and $\cos\theta_k = 1$, Therefore, all P^k accept inputs $x \in MOD_p^{-1}(1)$ with probability 1.

Call P^k "*good*" for an input $x \in MOD_p^{-1}(0)$ if P^k rejects x with probability at least 1/2.

Lemma 4. *For any $x \in MOD_p^{-1}(0)$, at least $(p-1)/2$ of all P^k are "good."*

Proof. According to Lemma 3 after reading an input $x = x_1 \ldots x_n$ the state vector of P^k is $|\psi\rangle = (\cos\theta_k)|1\rangle + (\sin\theta_k)|2\rangle$.

Therefore, the probability of accepting an input $x \in MOD_p^{-1}(0)$ is $\cos^2\theta_k$, which is less than or equal to 1/2 if and only if $\theta_k \in [\pi/4, 3\pi/4]$ or $\theta_k \in [5\pi/4j, 7\pi/4]$. As p is prime and $l(x)$ is relatively prime with p, it must be that $l(x) \bmod p$, $2l(x) \bmod p, \ldots,$ $(p-1)l(x) \bmod p$ are some permutation of $1, 2, \ldots, p-1$. Consequently, it is enough to find the size of the set $I = \{i_1, \ldots, i_l\} \subset \{1, \ldots, p-1\}$ such that $2\pi i_j/p \in [\pi/4, 3\pi/4]$ or $2\pi i_j/p \in [5\pi/4, 7\pi/4]$. Since the p points $2\pi/p, \ldots, 2\pi(p-1)/p, 2\pi$ are regularly distributed on the circumference of the circle and the sectors $[\pi/4, 3\pi/4]$ and $[5\pi/4, 7\pi/4]$ are exactly half of the circumference, we have $|I| \geqslant \lfloor p/2 \rfloor \geqslant (p-1)/2$. □

We call a set of quantum programs $S = \{P^{i_1}, \ldots, P^{i_t}\}$ "good" for $x \in MOD_p^{-1}(0)$ if at least 1/4 of all its elements are "good" for this x.

Lemma 5. *There is a set S of width-2 quantum branching programs with $|S| = t = \lceil 16 \ln p \rceil$ which is "good" for all inputs $x \in MOD_p^{-1}(0)$.*

Proof. The following procedure $\mathcal{A}$ is used to construct the set S:

For a fixed input x with $l(x) \leqslant p-1$, $\mathcal{A}$ selects a quantum branching program uniformly at random from $\{P^1, \ldots, P^{p-1}\}$.

The probability of selecting a "good" QBP at each step is at least 1/2. Using Chernoff bounds, we get the probability that less than 1/4 of all QBPs from the set S are "good" for any fixed x with $l(x) \leqslant p-1$ is at most

$$\exp(-16 \ln p)/2(1/2)^2/2 = 1/p.$$

Hence, the probability of constructing a set that is not "good" for at least one input x with $l(x) \leqslant p-1$ is at most $(p-1)/p > 0$. Therefore, there exists a set which is "good" for all inputs x with $l(x) \leqslant p-1$. This set is "good" for inputs x with $l(x) > p$ as well, since any QBP, P^k, returns the current state vector to the start state vector after reading every p ones, and hence, works the same way on any inputs x, x' with $l(x) = l(x') \bmod p$. $\square$

A program P accepting inputs $x \in MOD_p^{-1}(1)$ with probability 1 and rejecting inputs $x \in MOD_p^{-1}(0)$ with probability at least 1/8 can now be described: P's start and accepting states are the same as for the P^k's. Its transitions consist of a superposition of the transitions of QBP's from a "good" set $S = \{P^{i_1}, \dots, P^{i_t}\}$, weighted with equal amplitudes.

Notice the inputs $x \in MOD_p^{-1}(1)$ are always accepted by P with probability 1 because all P^k's accept them. On the other hand, for any input $x \in MOD_p^{-1}(0)$ at least 1/4 of all $P^k \in S$ reject it with probability at least 1/2 and the total probability of rejecting any $x \in MOD_p^{-1}(0)$ is at least 1/8.

The error can now be made as small as needed using standard techniques for reducing error in one-sided error computations. That is, $d = d(\epsilon)$ copies of P are taken and run uniformly at random. In this case the width of the resulting program will be $O(\log p)$.

Definition 8. A branching program P is called *stable* if its transformations do not depend on the level of P, i.e., $M_j(0)$ and $M_j(1)$ do not depend on j.

Observe that the proof of the above theorem constructs a quantum branching program for MOD_p that is stable.

Corollary 5. *The function MOD_p can be presented by a stable, read-once, width-$O(\log p)$ quantum branching program with one-sided error $\epsilon > 0$.*

Now we show that the MOD_p function is hard for randomized OBDD's.

7.2. Lower Bounds for randomized OBDDs for MOD

We start by listing some basic facts from Markov chain theory we will need in order to prove a lower bound for implementing the *MOD* function on a randomized OBDD (Theorem 6). For more background information the reader is advised to consult Section 2 of the book [9].

1. According to the classification theorem for Markov chains, the states of a Markov chain can be divided into ergodic and transient states. An *ergodic set of states* is a set which a process cannot leave once it has entered. A *transient set of states* is a set which a process can leave, but cannot return once it has left.
2. An arbitrary Markov chain C has at least one ergodic set. It is possible to have a Markov chain C without any transient set. If a Markov chain C has more than one ergodic set, then there is absolutely no interaction between these sets. Hence, we have two or more unrelated Markov chains lumped together. These chains can be studied separately. If a Markov chain consists of a single ergodic set, then the chain is called an *ergodic chain*. According to the classification theorem for Markov chains, every ergodic chain is either regular or cyclic.
3. An ergodic chain is *regular*, if for sufficiently high powers of the state transition matrix, A has only positive elements. Thus, no matter where such a process starts, after a sufficiently large number of steps it can be in any state. Moreover, there is a limiting vector of probabilities of being in the given states of the chain, and this vector does not depend on the initial state.

4. An ergodic chain is *cyclic*, if the chain has a period t, and all of its states are subdivided into t cyclic subsets ($t > 1$). For a given starting state a process moves through the cyclic subsets in a definite order, returning to the subset with the starting state every t steps. It is known that after a sufficient time has elapsed, the process can be in any state of the cyclic subset appropriate for the moment. Hence, for each of the t cyclic subsets, the tth power of the state transition matrix A^t describes a regular Markov chain.

Theorem 6. *Any stable probabilistic OBDD computing MOD_p has width at least p.*

Proof. Assume that there is a stable probabilistic OBDD P of width $q < p$ computing MOD_p with probability $1/2 + \epsilon$ for some fixed $\epsilon \in (0, 1/2]$. Without loss of generality, assume P reads the inputs in the natural order $x_1, \ldots, x_n$. We can also suppose without loss of generality that each level of P has exactly q nodes. During the computation, the macrostate of the program P on a level of nodes of P can be described by a probability distribution vector $\mu = (\mu_1, \ldots, \mu_q)$, where μ_i is the probability of being in the ith node of the level. So we can describe a computational process of P on an input $x = x_1 \ldots x_n$ as follows:

- The computation of P starts from an initial probability distribution vector $\mu(e)$ (here e denotes empty word).
- On the jth step, $1 \leqslant j \leqslant n$, P reads x_j and transforms the current vector μ to $\mu' = \mu A$, where A is the $q \times q$ stochastic matrix, $A = A(0)$ if $x_j = 0$ and $A = A(1)$ if $x_j = 1$.
- Suppose after the last (nth) step of the computation, the probability distribution vector is $\mu(x) = (\mu_1, \ldots, \mu_q)$. The program P then accepts the input x with probability $P_{\text{acc}}(x) = \sum_{i \in F} \mu_i$. So if $f(x) = 1$, then $P_{\text{acc}}(x) \geqslant 1/2 + \epsilon$; otherwise, $P_{\text{acc}}(x) \leqslant 1/2 - \epsilon$.

Let $\Sigma = \{x^{(n)}, \ldots, x^{(1)}\}$ be a set of input sequences where $x^{(i)} = 0^{n-i}1^i$. Here $0^k = \underbrace{0 \ldots 0}_{k}$, $1^k = \underbrace{1 \ldots 1}_{k}$. From now on, we consider only input sequences to P from Σ.

For each $x^{(i)} \in \Sigma$, according to our notations, we have that $\mu(0^{n-i}) = \mu(e)A^{n-i}(0)$. On the remaining part 1^i of $x^{(i)}$ a computation of P can be described by a Markov chain C. In this case, $\mu(0^{n-i})$ is the initial probability distribution for the Markov process and $A(1)$ is the transition probability matrix.

Now we estimate a number of states in the ergodic set of the Markov chain C. It is known that if an ergodic chain is a cyclic chain with the period t, then it has at least t states. Let $t_1, \ldots, t_h$ be the periods of the cyclic chains of C (if an ergodic chain is regular then $t = 1$).

>From the assumption that $q < p$, we get that $t_i < p$ for each cyclic chain. Denote by D the least common multiple of all such t. Because p is prime, t is relatively prime to p, D is relatively prime to p, and so is any positive degree D^m of D.

For the input sequence $x^{(k)}$ consider the final vector $\mu(x^{(k)})0$. Without loss of generality we can assume that there is a single accepting state. Let $\mu_{\text{acc}}(x^{(k)})$ be the probability of being in the accepting state after reading $x^{(k)}$. As after every D steps a process can be in any set of states containing the accepting state, the Dth power of A describes a regular Markov chain for this set. According to property 3 of Markov chains listed above we have that there exists an α such that $\lim_{r\to\infty} \mu_{\rm acc}(x^{(rD)}) = \alpha$. Hence, for any $\epsilon > 0$, it holds that

$$|\mu_{\rm acc}(x^{(D^m)}) - \mu_{\rm acc}(x^{(D^m p)})| < 2\epsilon$$

for m large enough. As P is supposed to $1/2 + \epsilon$ compute MOD_p, we have that $\mu_{\rm acc}(x^{(D^m p)}) \geqslant 1/2 + \epsilon$ and that $\mu_{\rm acc}(x^{(D^m)}) \leqslant 1/2 - \epsilon$. This contradicts the inequality above. □

8. General lower bound for quantum OBDDs

A general lower bound on the width of read-once quantum branching programs is now presented.

Theorem 7. *Let $\epsilon \in (0, 1/2)$. Let $f(x_1, \ldots, x_n)$ be a Boolean function $(1/2 + \epsilon)$-computed (computed with margin ϵ) by a quantum read-once branching program Q. Then*

$$\mathrm{width}(Q) = \Omega(\log \mathrm{width}(P)),$$

where P is a deterministic OBDD of minimal width computing $f(x_1, \ldots, x_n)$.

The proof of the Theorem 7 uses the same idea used in proving Theorem 4 and directly follows that proof. A deterministic OBDD P that represents the same function f is constructed such that

$$\mathrm{width}(P) \leqslant \left(\frac{2}{\epsilon}\right)^{2\cdot \mathrm{width}(Q)}.$$

The lower bound on width(Q) given by Theorem 7 proves that the quantum OBDD constructed for the MOD_p function has the optimal width possible.

Acknowledgments

We are grateful to Sasha Razborov for numerous suggestions on the results and presentations. We thank Denis Thérien for helpful discussions on ACC and Sasha Shen for productive discussions on the syntactic model. We also thank Tracy Conrad for helpful conversations.

References

[1] F. Ablayev, C. Moore, C. Pollett, Quantum and stochastic branching programs of bounded width, in: Proc. 29th Intl. Colloquium on Automata, Languages and Programming Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 343–354.

[2] F. Ablayev, A. Gainutdinova, M. Karpinski, On the computational power of quantum branching, in: Proc. FCT 2001, Lecture Notes in Computer Science, vol. 2138, 2001, pp. 59–70.

[3] F. Ablayev, M. Karpinski, On the power of randomized branching programs, in: Proceedings of the ICALP'96, Lecture Notes in Computer Science, vol. 1099, Springer-Verlag, 1996, pp. 348–356.

[4] F. Ablayev, M. Karpinski, A lower bound for integer multiplication on randomized ordered read-once branching programs, Informat. Comput. 186 (1) (2003) 78–89.
[5] A. Ambainis, L. Schulman, U. Vazirani, Computing with highly mixed states, in: Proc. 32nd ACM Symp. on Theory of Computing, 2000, pp. 697–704.
[6] D.A. Barrington, Bounded-width polynomial branching programs recognize exactly those languages in NC^1, J. Comput. Syst. Sci. 38 (1) (1989) 150–164.
[7] D.A. Barrington, D. Therien, Finite monoids and the fine structure of NC^1, J. ACM 35 (4) (1988) 941–952.
[8] A. Borodin, A. Razborov, R. Smolensky, On lower bounds for read-k-times branching programs, Comput. Complexity 3 (1993) 1–18.
[9] J.G. Kemeny, J.L. Snell, Finite Markov Chains, Van Nostrand, 1960.
[10] A. Kondacs, J. Watrous, On the power of quantum finite automata, Proc. 38th IEEE Conf. on Foundations of Computer Science (1997) 66–75.
[11] W. Maurer, J. Rhodes, A property of finite simple non-abelian groups, Proc. AMS 16 (1965) 552–554.
[12] C. Moore, J.P. Crutchfield, Quantum automata and quantum grammars, Theor. Comput. Sci. 237 (2000) 275–306.
[13] C. Moore, D. Thérien, F. Lemieux, J. Berman, A. Drisko, Circuits and expressions with non-associative gates, J. Comput. Syst. Sci. 60 (2000) 368–394.
[14] M. Nakanishi, K. Hamaguchi, T. Kashiwabara, Ordered quantum branching programs are more powerful than ordered probabilistic branching programs under a bounded-width restriction, in: Proc. 6th Intl. Conf. on Computing and Combinatorics (COCOON), Lecture Notes in Computer Science, vol. 1858, 2000, pp. 467–476.
[15] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press Cambridge, MA, 2000.
[16] C. Papadimitriou, Computational Complexity, Addison-Wesley Reading, MA, 1994.
[17] M. Rabin, Probabilistic automata, Informat. Control 6 (1963) 230–245.
[18] A.A. Razborov, Lower bounds for the size of circuits of bounded depth with basis $\{\&, \oplus\}$, Math. Notes Acad. Sci. USSR 41 (4) (1987) 333–338.
[22] M. Sauerhoff, D. Sieling, Quantum branching programs and space-bounded nonuniform quantum complexity, Theoret. Comput. Sci. 334 (1-3) (2005) 177–225.
[19] P. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26 (5) (1997) 1484–1509.
[20] R. Smolensky, Algebraic methods in the theory of lower bounds for Boolean circuit complexity, in: Proc. 19th ACM Symposium on the Theory of Computing, 1987, pp. 77–82.
[21] J.H. van Lint, R.M. Wilson, A Course in Combinatorics, second ed., Cambridge, 2001.
[23] I. Wegener, Branching Programs and Binary Decision Diagrams, SIAM Monographs Discrete Math. Appl. (2000).
[24] A.C. Yao, Lower bounds by probabilistic arguments, in: Proc. 24th IEEE Conf. on Foundations of Computer Science, 1983, pp. 420–428.