An Introduction to Quantum Algorithms

Computer scientists usually find conventional expositions of quantum computation difficult, since a lot of new concepts have to be learned together to make sense of it all. In this course, we take a different approach, starting from a point where every computer science student is comfortable, and introducing the new concepts one by one into the familiar framework. We hope it will be much easier this way. So let's start with deterministic finite automata!

Chapter 1

The simplest case: Real-time, constant-space, zero error

1.1 Deterministic model

You already know the stuff in Sipser's book. Note that the transitions of a DFA can be represented as 0-1 matrices, the states can be traced with vectors, and the execution on an input string boils down to multiplying this vector with the corresponding sequence of those matrices.

1.2 Probabilistic model

We define real-time[1] probabilistic finite automata (rtPFAs) by generalizing the matrix definition of rtDFAs to column stochastic transition matrices. Consider a transition graph representation for a moment. It is as if computation flows through many arrows (associated with the same input symbol) parallelly in each step. Under this regime (exact computation) rtPFAs are identical to rtDFAs: Since none of these computational paths can arrive at an incorrect response at the end, tracing any single path is sufficient for seeing what the machine will do. So we convert each rtPFA matrix (which have to contain at least one nonzero entry in each column) to a rtDFA matrix with a 1 in just the position corresponding to the topmost nonzero entry of the

[1] A real-time machine is one that consumes its input string from left to right at a rate of one symbol per move, and its runtime is therefore n+2, if we allow for endmarker symbols delimiting the input.

rtPFA in that column. The resulting machine is guaranteed to recognize the same language, with the same number of states!

1.3 Quantum model

Quantum physics turns out to allow a more general kind of transformation than the ones seen until now. Here are the new features: Computation can "fork" in two different ways in this new setup. One way is probabilistic branching, with which we are familiar from probabilistic computations. The new way is quantum branching, in which the branches are allowed to interact, merging or even canceling each other out in a novel manner, as we will see. Transition matrices for a rtQFA represent these alternatives as follows: For a k-state machine that can branch at most m ways probabilistically in any single step, we have m k×k matrices (stack these on each other to obtain an mk×k rectangle) for each input symbol. The matrices are allowed to contain real numbers in the interval [-1,1] in each entry. The wellformedness condition is that each column in these rectangles should have norm (i.e. the square root of the sum of the squares of the numbers in it) 1, and every pair of different columns should be orthogonal to each other (the sum of the products of the entries in the corresponding rows should be 0). The first step of a rtQFA on its input amounts to multiplying each of the m matrices separately with the initial state vector (containing a 1 for the start state and 0 for the rest) and writing the vectors that are the results of these multiplications as the children of the initial vector in a computation tree. Every subsequent input symbol is handled similarly, adding more generations to this tree, which is completed with the reading of the last input symbol. The probability of acceptance is calculated by squaring each entry corresponding to an accepting state in the last generation and adding these squares up.

The branches of this computation tree correspond to classical probabilistic branching. The square of the norm of a vector, say, v, in, say, the ith level of the tree is the probability $P(v)$ that that node is reached. Ignore all nodes containing all-zero vectors, computation simply does not branch there. Normalize all vectors like v in other nodes (by dividing each entry with the norm of that vector) to obtain v_N. The machine will be in the pure state (i.e. the state vector) described by v_N with probability $P(v)$ after executing the ith step. The overall "mixture" of states at a particular level will be best described by something called a density matrix, to be defined soon. First, let us formalize things a bit:

Definition 1. *A real-time quantum finite automaton (rt-QFA) is a 5-tuple* $\{Q, \Sigma, q, \{\mathcal{E}_\sigma\}_{\sigma\in\Sigma}, F\}$ *where*

- Q *is a finite set of classical states*
- Σ *is a finite input alphabet*
- $q{\in}Q$ *is the initial state*
- $F{\subset}Q$ *is the set of accepting states*
- $\{\mathcal{E}_\sigma\}_{\sigma\in\Sigma}$ *is a finite collection of* $|Q|m \times |Q|$*-dimensional matrices, each of which are composed of* m $|Q| \times |Q|$*-dimensional matrices (operation elements)* $\{E_i | 1 \le i \le m\}$ *for* m *a positive integer.*

If our knowledge of the current superposition of a quantum system is certain, then the system is said to be in a *pure state*. Sometimes, (e.g. during classical probabilistic computation) we only know that the system is in pure state $|\psi_i\rangle$ with probability p_i for $i > 1$, and with the probabilities adding up to 1. In this case, the system is said to be in a *mixed state*.

When m=1, the conditions on the transition matrices, to be discussed soon, turn out to require that they be unitary, and the machine gets weaker, unable to recognize some regular languages.

All observations on quantum systems in our discussion will be done by **projective measurements**. In the case of the rtQFA, this process is particularly simple, since all we wish is to see whether we have ended up in an accept state or not. We just specify a matrix

$$P_a = \sum_{q\in F} |q\rangle\langle q|,$$

where F is the set of accept states, and for any column vector $|v\rangle$, $\langle v|$ denotes the conjugate transpose of $|v\rangle$.

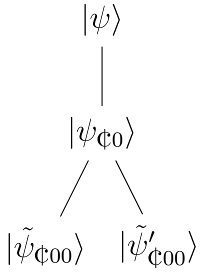
Consider the following quantum finite automaton with states $\{q_1, q_2, q_3\}$ where q_1 is the initial state, input alphabet $\{0\}$, m=2. For the only element of the input alphabet, the quantum operator $\mathcal{E}_0$ consists of matrices

$$E_{0,1} = \begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{ccc} q_1 & q_2 & q_3 \end{array} \\ \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 \end{pmatrix} \end{array} \qquad E_{0,2} = \begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{ccc} q_1 & q_2 & q_3 \end{array} \\ \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}.$$

Assume that the machine just performs the identity transformation on the left input end-marker ¢. Because q_1 is the initial state, ket vector $|\psi_{\text{¢}}\rangle = |q_1\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ represents the initial superposition.

After reading the first 0, the machine reaches new superposition $|\psi_{\text{¢}0}\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$ without a probabilistic branching.

After reading the second 0, the machine reaches superposition $|\tilde{\psi}_{\text{¢}00}\rangle = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$ in one probabilistic branch, and also reaches superposition $|\tilde{\psi}'_{\text{¢}00}\rangle = \begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \end{pmatrix}$ in another probabilistic branch. The machine is in a mixed state which is the ensemble of pure states $|\tilde{\psi}_{\text{¢}00}\rangle$ and $|\tilde{\psi}'_{\text{¢}00}\rangle$.


$|\psi\rangle$
$|\psi_{\text{¢}0}\rangle$
$|\tilde{\psi}_{\text{¢}00}\rangle$
$|\tilde{\psi}'_{\text{¢}00}\rangle$

From this point on, $|\tilde{\psi}_{\text{¢}00}\rangle$ and $|\tilde{\psi}'_{\text{¢}00}\rangle$ should be considered as separate probabilistic branches and evolution of the machine should be traced by applying the quantum operator to each probabilistic branches separately.

For a long input string, the number of probabilistic branches makes it impossible to trace a simple finite automaton by hand. For this reason, we introduce *density matrix* notation in which an n by n density matrix is enough to keep the state information of machine for any number of probabilistic and quantum branches.

Definition 2. *The **density matrix** representation of* $\{(p_i, |\psi_i\rangle) \mid 1 < i \leq$

$M < \infty\}$ *is*

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \tag{1.1}$$

Given a real-time quantum machine having a set of quantum operators $\{\mathcal{E}_{\sigma\in\Sigma}\}$ where each $\mathcal{E}_\sigma$ is an operator having elements $E_{\sigma,i}$ for each i up to m, the computation can be traced by density matrices for any given input string w. In the equation below, ρ_j denotes the system state after reading the j'th symbol of input w, that is, w_j.

$$\rho_j = \mathcal{E}_{w_j}(\rho_{j-1}) = \sum_i E_{w_j,i}\rho_{j-1}E^{\dagger}_{w_j,i}, \tag{1.2}$$

where $\rho_0 = |q_1\rangle\langle q_1|$ is the initial density matrix. The transition operators can be extended easily for any string as

$$\mathcal{E}_{w\sigma} = \mathcal{E}_w \circ \mathcal{E}_\sigma, \tag{1.3}$$

where $\sigma \in \Sigma$, $w \in \Sigma^*$, and $\mathcal{E}_\varepsilon = I$. Note that, if $\mathcal{E} = \{E_i | 1 \leq i \leq k\}$ and $\mathcal{E}' = \{E'_j | 1 \leq j \leq k'\}$, then

$$\mathcal{E}' \circ \mathcal{E} = \{E'_j E_i | 1 \leq i \leq k, 1 \leq j \leq k'\}. \tag{1.4}$$

For the example above, the density matrix representing the machine's state before starting the input is $\rho_{\text{¢}} = \sum_{i=1}^{1} |\psi\rangle\langle\psi| = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$.

Reading the first 0 can be simulated as follows:

$$\rho_{\text{¢}0} = \mathcal{E}(\rho_{\text{¢}}) = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 \end{pmatrix} +$$

$$\begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

After reading two 0's, machine is in the state $|\psi_{\text{¢}00}\rangle = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{pmatrix}$ with probability $p_1 = \frac{3}{4}$ and it is in the state $|\psi'_{\text{¢}00}\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ with probability $p_2 = \frac{1}{4}$.

Notice that vectors $|\tilde{\psi}_{\text{¢}00}\rangle$ and $|\tilde{\psi}'_{\text{¢}00}\rangle$ are normalized to obtain $|\psi_{\text{¢}00}\rangle$ and $|\psi'_{\text{¢}00}\rangle$ so that $\sum_i p_i = 1$. Therefore, density matrix after reading two 0's is

$$\rho_{\text{¢}00} = \frac{3}{4}\begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{2}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}.$$

As a direct result of their definition, being the sum of normalized square matrices reveals three important properties of density matrices:

- Numbers on the diagonal of a density matrix are always real numbers.
- i^{th} diagonal entry $d_{i,i}$ of a density matrix D is equal to the probability of being in state q_i.
- Sum of the diagonal elements (i.e. the trace) of a density matrix is 1.

Using these facts, note that the probability that a rt-QFA will accept a string w equals $\text{Tr}(P_a \rho_w)$, where ρ_w is the density matrix representing the mixed state of the machine after processing the string ¢w\$, and for any square matrix M, Tr(M) denotes the trace of M.

We are now ready to state the well-formedness condition for modern rt-QFAs. For the transition matrices of a quantum machine to be considered legal according to quantum physics, they have to induce a configuration transition matrix whose columns are orthonormal for any given input string. As an example, consider a rt-QFA M with $Q = \{q1, q2, q3, q4, q5\}$, $m = 3$ and $\Sigma = \{a, b\}$. Considering the left and right end-marker symbols as well, we have:

$$\mathcal{E}_{\text{¢}} = \begin{array}{c|c|} & q_1 \quad q_2 \quad \cdots \quad q_5 \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{\text{¢},1} \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{\text{¢},2} \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{\text{¢},3} \\ \hline \end{array} \qquad \mathcal{E}_{a} = \begin{array}{c|c|} & q_1 \quad q_2 \quad \cdots \quad q_5 \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{a,1} \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{a,2} \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{a,3} \\ \hline \end{array}$$

$$\mathcal{E}_{b} = \begin{array}{c|c|} & q_1 \quad q_2 \quad \cdots \quad q_5 \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{b,1} \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{b,2} \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{b,3} \\ \hline \end{array} \qquad \mathcal{E}_{\$} = \begin{array}{c|c|} & q_1 \quad q_2 \quad \cdots \quad q_5 \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{\$,1} \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{\$,2} \\ \hline \begin{matrix} q_1 \\ q_2 \\ \vdots \\ q_5 \end{matrix} & E_{\$,3} \\ \hline \end{array}$$

For any given input string x, the transition amplitudes in these matrices induce a corresponding set of configuration transition matrices, $\{E_k\}$, where the $(i,j)^{th}$ entry of E_k, the amplitude of the transition from configuration c_j to c_i by branching to the k'th probabilistic path, is defined by the relevant transition matrix whenever c_j is reachable from c_i in one step, and is zero otherwise. The $\{E_k\}$ form an operator $\mathcal{E}$.

In our example, we run M on an input string “aaaba”. A configuration of a rt-QFA is a <state, head position> pair. Our transition matrix should represent the transition amplitudes between all possible configurations, with all possibilities for the written register symbol. On input “aaaba”, we have (with ¢ and \$) 7 head positions and configurations $C = \{(q, l) | q \in Q, 1 \leq l \leq 7\}$. In this case, $\mathcal{E}$ is

$$\begin{array}{c|cccc|}
 & c_1 & c_2 & \dots & c_{|C|} \\ \hline
c_1 & & & & \\
c_2 & & & & \\
\vdots & & & E_1 & \\
c_{|C|} & & & & \\ \hline
c_1 & & & & \\
c_2 & & & & \\
\vdots & & & E_2 & \\
c_{|C|} & & & & \\ \hline
c_1 & & & & \\
c_2 & & & & \\
\vdots & & & E_3 & \\
c_{|C|} & & & & \\ \hline
\end{array}$$

$\mathcal{E}$ is represented above by a $|C|m \times |C|$-dimensional matrix, where $C = \{c_1, c_2, ..., c_{7|Q|}\}$, by concatenating each E_i one under the other. The well-formedness rule dictated by physics is that $\mathcal{E}$ is a superoperator, i.e. that

$$\sum_i E_i^\dagger E_i = I.$$

It can be verified that the above condition is satisfied if and only if the columns of $\mathcal{E}$ form an orthonormal set. Let c_{j_1} and c_{j_2} be two configurations with corresponding columns v_{j_1} and v_{j_2}. For an orthonormal set to be formed, we must have

$$v_{j_1}^\dagger v_{j_2} = \begin{cases} 1 & \text{if } j_1 = j_2, \\ 0 & \text{if } j_1 \neq j_2 \end{cases}$$

for all such pairs.

We can ensure that all such configuration transition matrices are well-formed by preparing the program's matrices $\mathcal{E}_{\sigma\in\tilde{\Sigma}}$ such that their columns are orthonormal.

1.4 Quantum Machines Can Simulate Classical Machines

We prove this result for real-time machines, but the same approach is valid for all higher models.

Theorem 1. *Given a rt-PFA $P = (Q_P, \Sigma, \delta_P, q_{0P}, F_P)$, there exists a corresponding rt-QFA $Q = (Q_P, \Sigma, \mathcal{E}_{\sigma\in\Sigma}, q_{0P}, F_P)$ such that*

$$p_Q^{acc}(w) = p_P^{acc}(w)$$
$$p_Q^{rej}(w) = p_P^{rej}(w)$$

for any given w.

Proof. For machine P, we have transition matrices $P_\sigma = \{p_{j,k} | j, k \in Q_P\}$ describing the transition probabilities among each pair of states of the machine, upon seeing the symbol $\sigma \in \Sigma$. We construct our $\mathcal{E}_\sigma$ operators having elements $E_{\sigma,(n)}$ for every state q_n such that, for a particular input $\sigma \in \Sigma$;

$$E_{\sigma,(n)_{i,n}} = \sqrt{P_{\sigma i,n}}$$

for all i, and all other positions are 0. □

Do not worry about this not looking like a generalization of PFAs, it is. Given a rtPFA P with k states, build the rtQFA Q corresponding to it by setting m to equal k, using a separate square matrix (with zeros everywhere except in the j'th column) in Q for the j'th column of the corresponding matrix of P. That location will contain the square roots of the entries in P's j'th column.

1.5 Promise problems

Here is an example of a superiority of quantum over classical: For a natural number k, we are promised that the input will be of the form 1^{i2^k} for some natural number i. We are supposed to accept if and only if i is even. The two-state quantum machine $M_k = (\{q_0, q_1\}, \{1\}, \delta, q_0, \{q_0\})$, and δ is given by

$P_1 = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$, where $\theta = \frac{\pi}{2^{k+1}}$ does this with zero error, whereas any probabilistic machine for this task has to have at least 2^{k+1} states, as Ambainis and Yakaryılmaz have shown.

1.6 State (in)efficiency of exact QFAs in general language recognition without promises

The number of states of a rtQFA is related to a quantity called the one-way communication complexity. Let us see what that means. Assume that two people, Alice and Bob, who are far away from each other, have to decide a language together. A prefix x of the input string is given to Alice, and the rest, y, is given to Bob. Alice is allowed to send a single message to Bob, and Bob is supposed to announce the final decision. The one-way (quantum) communication complexity of language L is the minimum number of (quantum) bits that has to be sent by Alice. Consider the infinite two-dimensional 0-1 matrix M indexed by strings from the input alphabet such that M(x,y)=1 iff $xy \in L$. Note that, for regular L, this matrix has only a finite number of distinct rows, and this number is d, the number of states of a minimal DFA for L. Any rtQFA with q states for L can be simulated by Alice and Bob, with Alice sending the state of the machine at the end of the processing of x with a $\log q$-qubit[2] message to Bob. We will prove that no exact rtQFA for L can have fewer states than d, by showing that the logarithm of this number equals the one-way quantum communication complexity of L.

The Shannon entropy H(B) of a set of messages described with random variable B equals the average number of classical bits required to encode members of that set. f $p_1, \cdots, p_n$ are the probabilities that B can take its possible values, then H(B)=$-\sum_x p_x \log p_x$. The von Neumann entropy S(ρ_B)=$-tr(\rho_B \log \rho_B)$ similarly equals the number of quantum bits. This value depends on the probability distribution (in the quantum case, the density matrix) of the messages, for instance, if we are certain that all messages

[2] A *qubit* is just a two-state quantum system.

will contain “ATTACK!” and nothing else, the entropy is zero, since no message need be sent! To get the communication complexity, we have to consider the mixed state of uniformly random rows of the communication matrix, since this information is what is supposed to be sent to Bob.

Reduce the communication matrix to d distinct rows. Imagine the rows being chosen randomly bit by bit, i.e. column by column. Let p(0) be the probability of a 0 in column 1 (i.e. the number of 0's in column 1 divided by d). Then a 0 is chosen with probability p(0), and a 1 is chosen with probability 1-p(0). Partition the rows to sets I_0 and I_1, the sets of rows starting with 0 and 1, respectively. When b is the chosen bit, the process continues with set I_b and the next column. If a complete row x is determined, let ρ_x denote the density matrix of just the message about that row.

Let ρ_t denote the density matrix of the (possibly mixed) message corresponding to a row starting with t, chosen uniformly among all such rows. The probability that a b is chosen after t is called $p_t(b)$, the associated random variable is called B, and the number of different rows beginning with t is called row_t. Note that $H(B) = -(p_t(0)\log p_t(0) + p_t(1)\log p_t(1))$. Our aim is to prove $S(\rho_\varepsilon) \geq \log row_\varepsilon$, where $row_\varepsilon = d$. We will do this by proving that $S(\rho_t) \geq \log row_t$ for any t.

Bob can decide membership of xy in L correctly with probability 1. This means that he receives exact information about the row corresponding to x in the message sent by Alice. The Holevo theorem will help us in this regard:

The Holevo Theorem. *Suppose Alice prepares a state ρ_X where X=0,...,n with probabilities $p_0, \cdots, p_n$. Bob performs a measurement on that state, with measurement outcome Y. The Holevo bound states that for any such measurement Bob may do, $H(X:Y) \leq S(\rho) - \sum_x p_x S(\rho_x)$, where $\rho = \sum_x p_x \rho_x$.*

Intuitively, mutual information (H(X:Y)) measures the information that X and Y share: it measures how much knowing one of these variables reduces uncertainty about the other. For example, if X and Y are independent, then knowing X does not give any information about Y and vice versa, so their mutual information is zero. At the other extreme, if X and Y are identical then all information conveyed by X is shared with Y: knowing X determines the value of Y and vice versa. As a result, in the case of identity the mutual information is the same as the uncertainty contained in Y (or X) alone, namely the entropy of Y (or X: clearly if X and Y are identical they have equal entropy). The exactness condition in our case means than H(X:Y)=H(X), so we get $S(\rho_t) \geq p_t(0)S(\rho_{t0}) + p_t(1)S(\rho_{t1}) + H(B)$ for any t.

Let us show by induction that $S(\rho_t) \geq \log row_t$ for any t. The basis step:

$S(\rho_x) \geq 0$ for any completely chosen x, from the definition of von Neumann entropy.

The inductive step: For any t, $S(\rho_t) \geq p_t(0)S(\rho_{t0}) + p_t(1)S(\rho_{t1}) + H(B)$. So

$S(\rho_t) \geq p_t(0) \log row_{t0} + p_t(1) \log row_{t1} + H(B)$
$= p_t(0) \log(p_t(0) row_t) + p_t(1) \log(p_t(1) row_t) - p_t(0) \log p_t(0) - p_t(1) \log p_t(1)$
$= p_t(0) \log p_t(0) + p_t(0) \log row_t + p_t(1) \log p_t(1) + p_t(1) \log row_t - p_t(0) \log p_t(0) - p_t(1) \log p_t(1)$
$= p_t(0) \log row_t + p_t(1) \log row_t$,

(since $row_{t0} = p_t(0) row_t$, etc.), yielding $S(\rho_t) \geq \log row_t$. This means that the dimension of the message space must be at least d, so the rtQFA must have at least d states.

1.7 Caveat

Zero-error computation as described here[3] is a fictional concept, because of precision issues. See pages 194-195 of Nielsen and Chuang's book for the argument that imperfect implementations of operators can approximate the behavior of perfect ones. "Approximate" is the key word here; which means that an actual implementation will involve a slight change in observation (accept/reject) probabilities, thereby destroying "exactness", and firmly reminding us that very long computations require very precise implementation of amplitudes to even sustain a nonzero error bound.

1.8 But...

But what about good old language recognition power? We'll look at exact recognition by rtQFA's in the next chapter, since it's just a special case of a more general theorem about bounded error recognition.

[3] The reader should note that this warning also applies to computation with unbounded error or one-sided error.

Chapter 2

Real-time, constant-space, bounded-error computation

A machine M recognizes a language with bounded error if there exists a number $\varepsilon < 1/2$ such that M is guaranteed to give the correct response to any input string with probability at least $1 - \varepsilon$. If you know to build such a machine M1 for a language for some error bound ε, you can build another machine M2 for the same language with far smaller error by just letting the M2 simulate the execution of M1 on the input string several times, and report the majority response given by these simulations. If the machines under consideration are real-time, this would have to be a parallel simulation. This is a good place to recall the tensor product, which allows one to view several parallel systems as a single combined system. For any positive numbers m and n, and any two matrices A (which is $m \times n$ and B, the tensor product $A \otimes B$ is the matrix

$$\begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix} \otimes B = \begin{pmatrix} a_{1,1}B & \cdots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \cdots & a_{m,n}B \end{pmatrix}.$$

So when running two QFA's (or PFA's) with n and m states parallelly, the state vector of the combined machine would be

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \otimes \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} a_1b_1 \\ \vdots \\ a_1b_m \\ a_2b_1 \\ \vdots \\ a_nb_m \end{pmatrix},$$

and the transition matrices of the resulting machine would be obtained by taking the tensor products of the operation elements. The parallel simulation of multiple copies of a machine for reducing the error can be performed by constructing such a combined machine, whose initial state is the state that corresponds to the collection of start states of the original machines, and whose accept states are those that correspond to collections containing a majority of accept states.

2.1 A Characterization of Regular Languages

Definition 3. *Let x and y be strings, and let L be any language. We say that x and y are* ***distinguishable by L*** *if some string z exists, whereby exactly one of the strings xz and yz is a member of L. Otherwise, for every string z, we have $xz \in L$ whenever $yz \in L$, and we say that x and y are* ***indistinguishable by L****. If x and y are indistinguishable by L, we write $x \equiv_L y$.*

Proposition 1. *$\equiv_L$ is an equivalence relation on the set of all strings.*

Definition 4. *Let X be a set of strings. X is* ***pairwise distinguishable by L*** *if every two distinct strings in X are distinguishable by L.*

Definition 5. *The* ***index of L*** *is the maximum number of elements in any set that is pairwise distinguishable by L.*

Proposition 2. *If a language L is recognizable by a DFA with k states, it has index at most k.*

Proof. Let M be a k-state DFA that recognizes L. Suppose for a contradiction that L has index greater than k. That means some set X with more than k elements is pairwise distingusihable by L. Because M has k states, the pigeonhole principle implies that X contains two distinct strings x and y, where $\delta(q_o, x) = \delta(q_o, y)$. Here $\delta(q_o, x)$ is the state that M is in after starting in the start state q_0 and reading input string x. Then, for any string $z \in \Sigma^*$, $\delta(q_o, xz) = \delta(q_o, yz)$. Therefore either both xz and yz are in L or neither are in L. But then x and y aren't distinguishable by L, contradicting our assumption that X is pairwise distinguishable by L. □

Proposition 3. *If the index of a language L is a finite number k, it is recognized by a DFA with k states.*

Proof. Let $X = \{s_1, .., s_k\}$ be pairwise distinguishable by L. We construct DFA $M = (Q, \Sigma, \delta, q_o, F)$ with k states recognizing L. Let $Q = \{q_l, ..., q_k\}$ and define $\delta(q_i, a)$ to be q_j, where $s_j \equiv_L s_i a$. Note that $s_j \equiv_L s_i a$ for some $s_j \in X$; otherwise, $X \cup s_i a$ would have $k + 1$ elements and would be pairwise distinguishable by L, which would contradict the assumption that L has index k. Let $F = \{q_i | s_i \in L\}$. Let the start state q_o be the q_i such that $s_i \equiv_L \varepsilon$, where ε is the empty string. M is constructed so that, for any state q_i, $\{s | \delta(q_o, s) = q_i\} = \{s | s \equiv_L s_i\}$. Hence M recognizes L.

□

Theorem 2. ***Myhill-Nerode Theorem*** *A language L is regular iff it has finite index. Moreover, its index is the size of the smallest DFA recognizing it.*

Proof. ($\Rightarrow$) Suppose that L is regular and let k be the number of states in a DFA recognizing L. Then by Proposition 2 L has index at most k.
($\Leftarrow$) Conversely, if L has index k, then by Proposition 3 it is recognized by a DFA with k states and thus is regular.
To show that the index of L is the size of the smallest DFA accepting it, suppose that L's index is exactly k. Then, by Proposition 3, there is a k-state DFA accepting L. That is the smallest such DFA because if it were any smaller, then it would follow from Proposition 2 that the index of L is less than k. □

2.2 Bounded-error QFAs can only recognize regular languages

Let us now show that no nonregular language has a bounded-error rtQFA recognizing it. We will need to talk about "distances" between quantum states, classical distributions, and the relations among these distances.

Define the distance between two quantum states A and B as

$$D(A, B) = \|A - B\|_{tr},$$

where $\|M\|_{tr} = Tr(\sqrt{MM^\dagger})$. The distance D(p,q) between two classical probability distributions p and q is just the sum of the absolute values of the differences of the probabilities of all the corresponding events in p and q. $D(A, B)$ is known to be an upper bound for the distance among the observation probability distributions from quantum states A and B.

Theorem 3. *The languages recognized by rt-QFA with bounded error are exactly the regular languages.*

Proof. It's clear that rt-QFA can recognize regular languages by simulating DFA. Now we will show that they can't recognize any other language in bounded error setting.

Suppose a nonregular language L is recognized by rt-QFA $M = \{Q, \Sigma, q, \{\mathcal{E}_\sigma\}_{\sigma\in\Sigma}, F\}$ with bounded error ϵ. Define an equivalence relation "$\equiv_L$" on $x, y \in \Sigma$ such that $x \equiv_L y$ if for any $z \in \Sigma$, $xz \in L$ if and only if $yz \in L$. Then, by Myhill-Nerode theorem, it is sufficient to prove that number of equivalence classes are finite.

Now let $S = \{A| \; \|A\|_{tr} \leq 1,$ *and* A *is a linear* operator on the vector space spanned by vectors corresponding to states in Q$\}$. Then S is a bounded subset from a finite-dimensional space. Let $\rho_x = \mathcal{E}_{x_n}\mathcal{E}_{x_{n-1}} \ldots \mathcal{E}_{x_1}(\rho_0)$, then for every input string x, it can be seen that $\rho_x \in S$, because we have $\|\rho_x\|_{tr} = Tr(\rho_x) = 1$.

Suppose $x \not\equiv_L y$, that is there exists a $z \in \Sigma$ such that $xz \in L$ and $yz \notin L$. Then we have

$$Tr(P_{acc}\mathcal{E}_z(\rho_x)) \geq 1 - \epsilon \text{ and } Tr(P_{acc}\mathcal{E}_z(\rho_y)) \leq \epsilon.$$

Therefore, we have

$$\begin{aligned}\|\mathcal{E}_z(\rho_x) - \mathcal{E}_z(\rho_y)\|_{tr} \geq\ & |Tr(P_{acc}\mathcal{E}_z(\rho_x)) - Tr(P_{acc}\mathcal{E}_z(\rho_y))| + \\ & |Tr(P_{rej}\mathcal{E}_z(\rho_x)) - Tr(P_{rej}\mathcal{E}_z(\rho_y))| \\ & \geq 1 - 2\epsilon.\end{aligned}$$

we also have $\|\rho_x - \rho_y\|_{tr} \geq \|\mathcal{E}_z(\rho_x) - \mathcal{E}_z(\rho_y)\|_{tr}$ from the fact that applying the same operator on two different operators doesn't increase the distance between them. By combining these two formulas, we get

$$\|\rho_x - \rho_y\|_{tr} \geq 1 - 2\epsilon$$

This formula says that there should be at least $1 - 2\epsilon$ difference between each ρ_x and ρ_y satisfying $x \not\equiv_L y$ and from the Myhill-Nerode theorem, we know there should be infinite number of such x and y's so that they construct infinite number of equivalence classes for L to be nonregular. Call the representatives of these classes x1, x2,.... Since S is bounded in a finite-dimensional space, one can extract a Cauchy sequence (a convergent subsequence) from this sequence to find an x and a y such that $x \not\equiv_L y$ and yet $\|\rho_x - \rho_y\|_{tr} \leq 1 - 2\epsilon$. This leads to a contradiction. Therefore, there is no such nonregular language L recognized by *rt-QFA* in bounded error setting. □

2.3 Bounded-error rtPFA's can be more succinct than rtDFA's

Let L_n be the language containing just the string a^n. The minimal DFA for this language contains $n + 2$ states. Freivalds showed that any L_n is recognized with bounded error by a rtPFA with $O(\log^2 n)$ states, as follows:

Compile a list of $O(\frac{\log n}{\log\log n})$ primes, each of magnitude $O(\log n)$. The algorithm uses these primes, and a carefully chosen probability x:

———

-Randomly pick one of these primes, p.

-While walking on the input, count modulo p, rejecting with probability x each time this counter comes back to zero. When the input w finishes, accept if and only if $|w| = n$ mod p.

———

Facts about primes guarantee that $|w| \neq n \bmod p$ for most of the p's when $|w| < n$. Setting $x = O(1)p/n$ can ensure that inputs longer than n for which a sizable fraction of the primes would equal $n \bmod p$ would be rejected with high probability before reaching the end.

2.4 Bounded-error rtQFA's can be more succinct than rtPFA's

Consider the language L_p of strings on a unary alphabet whose lengths are multiples of the prime number p. A rtPFA with a unary alphabet can be seen as a Markov chain. The theory of Markov chains says that the states of such a machine are either transient or recurrent: There exists a length l, such that the probability that the machine will be in one of the transient states after more than l steps is zero. The recurrent states are partitioned to one or more groups, which never communicate with each other. Such a group is either periodic or not. If a group is not periodic, the probabilities of the states in it converge to a stationary distribution. If a group is periodic, there is a number $d > 1$, such that the group is divided to d subgroups, and one travels these groups one by one before returning to the original subgroup d steps later. Note that a rtPFA recognizing L_p must have one or more periodic groups, since otherwise we would classify all strings above a certain length as members (or nonmembers). Assume that L_p is recognized by a rtPFA with fewer than p states and with k periodic groups. So the periods (d) of all these groups must be numbers less than p. Let D be the lcm of these periods. Multiply D with a big prime A (other than p) to obtain a "sufficiently big" length that is of course relatively prime to p. Note that $1^{AD} \notin L_p$, so the total probability of accept states at that point must be at most $\frac{1}{2} - \varepsilon$, but $1^{ADp} \in L_p$, so the total probability of accept states at that point must be at least $\frac{1}{2} + \varepsilon$, which is a contradiction with the fact that the overall probability distribution must be almost the same at these two points.

We conclude that QFAs beat PFAs in this regard, since L_p is recognized by a small rtQFA: Start by considering 2-state rtQFA's M_k for natural numbers k less than p that rotate the vector by the angle $\frac{2\pi k}{p}$. All these machines accept members of L_p with no error, but may also accept nonmembers with undesirably large probability. The trick is to see that there exists a way of choosing $O(\log n)$ machines from this set so that the combined machine that runs them parallelly on the input has an upper bound of 7/8 on its error. Combining this combined machine with further copies of itself yields the required rtQFA for any desired nonzero level of error.

Chapter 3

Two-way, constant-space computation

Let us give one more capability to our finite automata, namely, that of moving the input tape head to the left, or leaving it one the same square, during a computational step. You may ask, is this really a constant-space machine? Obviously we seem to need arbitrarily big amounts of read-only "memory" to just store the input, and the information about the position of the head on the tape requires $\log n$ bits to represent. Let us not worry about these issues in this section, and see what we can do.

3.1 Two Way Deterministic Finite Automata

We will distinguish between the **real-time**, **one-way**, and **two-way** modes of tape head movement. A tape with real-time access is scanned by its head in a single sweep from the left end to the right end with a new symbol scanned at every computational step. One-way access is more general, in the sense that pausing the head on the same square for some steps in the single left-to-right scan is also allowed. Finally, the head can move to the left, as well as to the right, in the case of two-way access.

Definition 6. *A two-way deterministic finite automaton* ***2DFA*** *is a quintuple $M = (Q, \Sigma, \delta, q_0, F_a, F_r)$ where Q, Σ and q_0 are defined as in the case of the usual real-time DFA, F_a and F_r are the sets of accept and reject states, with the remaining states designated to be non-halting as in the case of TMs, and δ is a map from $Q \times \Sigma$ to $Q \times \{L, R\}$. If $\delta(q, a) = (p, L)$ then in state q,* *scanning input symbol a, the 2DFA enters state p and moves its input tape head one square to the left. If $\delta(q, a) = (p, R)$, the 2DFA enters state p and moves its head one square to the right.*

We assume throughout the course that the input string is written on a read-only tape, sandwiched between a left end-marker symbol and a right end-marker symbol, which are not members of the input alphabet. Earlier definitions of 2DFAs involved no end-markers, and the machine was said to accept when the input head fell off the right end of the input. Our convention, with acceptance based on states, is equivalent to this from a language recognition power point of view, though not from the point of view of state economy. (It is important in our convention to assume that the head never visits the squares beyond the end-markers.)

- Let $L_m = \{a^k | k \equiv 0\, mod(m)\}$ where $m > 1$. The smallest DFA recognizing L_m has m states. A 2DFA with much fewer than $2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19$ states exists for the language $L_{(2\times3\times5\times7\times11\times13\times17\times19)}$: In the first pass, while going from the left to the right, the 2DFA checks whether the string is in L_2. In the second pass, while going from the right to the left, 2DFA checks whether the string is in L_3, and continuing this way, it is able to determine whether the string is in L_m in 8 passes.

3.2 The Reduction of 2DFA to 1DFA

Theorem 4. *2DFA can recognize only regular languages.*

Proof. Let M be a 2DFA with set of states S recognizing language L. For every string $w \in \Sigma^*$, define a function $\tau_w : \{\bar{s_0}\} \cup S \rightarrow \{0\} \cup S$. For $s \in S$, $\tau_w(s)$ describes the ultimate result of the motion of M when started in state s on the rightmost symbol of w, i.e. if with these initial conditions M ultimately leaves w from the right in the state s' then $\tau_w(s) = s'$, on the other hand if M never leaves w, or leaves w from the left, then $\tau_w(s) = 0$. $\tau_w(\bar{s_0})$ describes the result of the motion when M is started in the initial state s_0 on the leftmost symbol of w. Now it is easy to see that if $\tau_{w_1} = \tau_{w_2}$ then $w_1 \equiv_L w_2$. But if n is the number of states of M there are at most $(n+1)^{(n+1)}$ distinct τ_w. Then by the Myhill-Nerode Theorem, we conclude that L is regular, since L has finite index. □

Construction of 1DFA from 2DFA: Let M be a 2DFA recognizing language L. We create a 1DFA M' recognizing the same language L as follows:

- As the state set of M' we take different functions τ_w which arise for $w \in \Sigma^*$.
- We define the initial state of M' to be τ_ε so that $\tau_\varepsilon(\bar{s_0}) = s_0$ and $\tau_\varepsilon(s) = 0$ for all $s \in S$.
- In order to learn the function τ_w for some $w \in \Sigma^*$, we start the machine from each state on the rightmost symbol of w. For each new function τ_w, we add a new state to the state set of M'.We can detect infinite loops since there are finite number of configurations (i.e. state/head-position pairs). If the machine does not move off w, either accepting or rejecting it in less than $|w| \times (n+2)$ units of time, then some configuration is repeated, and the machine has entered an infinite loop, which means that w will not be accepted.
- Use the method of Proposition 3 to construct the transition function.
- We define the accept states of M' by simulating the 2DFA with a member string from each equivalence class of the relation associated with L. If the string is accepted, the state representing the class becomes an accept state of M'.

3.3 2pfa's

2-way PFA's can recognize some nonregular languages with bounded error. See http://www.dmtcs.org/dmtcs-ojs/index.php/dmtcs/article/view/1457/3371, (Section 6) for an example.

It is, however, known that 2PFA's cannot recognize nonregular languages in polynomial expected time:

THEOREM: For any polynomial p, 2pfa's with expected runtime $O(p(n))$ recognize only the regular languages with bounded error.

Proof. One starts by defining a quantitative measure of the nonregularity of a language $L \subseteq \Sigma^*$. For a positive integer n, two strings $w, w' \in \Sigma^*$ are *n-dissimilar*, written $w \nsim_{L,n} w'$, if $|w| \leq n$, $|w'| \leq n$, and there exists a *distinguishing string* $v \in \Sigma^*$ with $|wv| \leq n$, $|w'v| \leq n$, and $wv \in L$ iff $w'v \notin L$. Let $N_L(n)$ be the maximum k such that there exist k distinct strings that are pairwise $\nsim_{L,n}$. It can be shown that

FACT (*): If L is not regular, then $N_L(n) \geq \frac{n}{2} + 1$ for infinitely many n.

In the rest of the proof, one develops a technique for constructing a Markov chain $P_{A,xy}$ with $2c$ states that models the computation of a given 2pfa A with c states on the concatenated string xy, where x and y are given strings. State 1 of the Markov chain corresponds to A being at the beginning of its computation on the last symbol of $¢x$. (Note that every 2pfa can be modified to start here, without changing the recognized language. The non-halting states of the modified 2pfa are $\{q_1, q_2, \ldots, q_{c-1}\}$.) For $1 \leq j \leq c-1$, state j of the Markov chain corresponds to A being in the configuration with the machine in state q_j and the head on the last symbol of $¢x$, and state $c+j-1$ corresponds to A being in the configuration with the machine in state q_j and the head on the first symbol of $y¢$. State $2c-1$ corresponds to a disjunction of rejection, infinite loop with the head never leaving the region $¢x$, and infinite loop within the region $y¢$. State $2c$ corresponds to acceptance. The probability that $P_{A,xy}$ is absorbed in state $2c$ when started in state 1 equals the probability that A accepts xy.

The proof then considers any 2pfa M that recognizes language L in expected time $T(n)$, and proceeds to establish a lower bound, in terms of $N_L(n)$, on $T(n)$. This is accomplished by showing that, for sufficiently large values of n, if the desired lower bound does not exist, then there must be two pairwise $\nsim_{L,n}$ strings w and w', with distinguishing string v, such that the Markov chains $P_{M,wv}$ and $P_{M,w'v}$ are "too close" according to a notion of closeness defined in It is proven that, if $P_{M,wv}$ and $P_{M,w'v}$ are so close, and if it is guaranteed that both Markov chains are absorbed to state $2c-1$ or $2c$ with total probability 1 within expected time $T(n)$, then the acceptance probabilities of wv and $w'v$ must be so close that they must both be members (or non-members) of L, contradicting their n-dissimilarity, thereby establishing the desired bound on $T(n)$. The Theorem statement is then obtained by combining this result with Fact (*).

□

3.4 2qcfa's (2pfa's plus a few qubits)

The model that we will use is the two-way finite automaton with quantum and classical states (2qcfa), introduced by Ambainis and Watrous,in which the quantum and classical memories are nicely separated, allowing a precise quantification of the amount of "quantumness" required for the task at hand. These machines can be viewed as two-way deterministic automata augmented with a quantum register of constant size (typically, just one or two qubits). The input string is assumed to be written between two end-marker symbols on a tape. A 2qcfa starts with both its quantum and classical parts set to be in their respective initial states, and operates separately on these parts at each step:

- First, a superoperator (Figure 3.1), determined by the current classical state and the symbol being scanned on the input tape, is applied to the quantum register, yielding an outcome.

- Then, the next classical state and tape head movement direction is determined by the current classical state, the symbol being scanned on the input tape, and the observed outcome.

Execution halts when the machine enters an accepting or rejecting classical state.

For a 2qcfa with j quantum states, each superoperator $\mathcal{E}$ is composed of a finite number of $j \times j$ matrices called *operation elements*, $\mathcal{E} = \{E_1, \ldots, E_k\}$, satisfying

$$\sum_{i=1}^{k} E_i^{\dagger} E_i = I, \tag{3.1}$$

where $k \in \mathbb{Z}^+$, and the indices label the possible outcomes. When a superoperator $\mathcal{E}$ is applied to a quantum register in superposition $|\psi\rangle$, one observes the outcome i with probability $p_i = \langle\widetilde{\psi_i}|\widetilde{\psi_i}\rangle$, where $|\widetilde{\psi_i}\rangle$ is calculated as $|\widetilde{\psi_i}\rangle = E_i|\psi\rangle$, and $1 \leq i \leq k$. If the outcome i is observed ($p_i > 0$), the new superposition is obtained by normalizing $|\widetilde{\psi_i}\rangle$, yielding $|\psi_i\rangle = \frac{|\widetilde{\psi_i}\rangle}{\sqrt{p_i}}$. Unitary operations such as rotations, as well as measurements and probabilistic branchings can be realized within this framework.

Figure 3.1: Superoperators

Here is how such a machine can recognize a nonregular language with bounded error in polynomial expected time:

```
Check if the input is of the form $a^+b^+$, otherwise REJECT.
LOOP:
 Move the head right to the next $a$.
 Set the qubit to $|q_0\rangle$.
 While the currently scanned symbol is $a$:
  Rotate the qubit with angle $\sqrt{2}\pi$.
  Move the head to the right.
 While the currently scanned symbol is $b$:
  Rotate the qubit with angle $-\sqrt{2}\pi$.
  Move the head to the right.
 Measure the qubit. If the result is $q_1$, REJECT. (I)
 (The currently scanned symbol is the right end-marker.)
 Repeat twice:
  Move the tape head to the first input symbol.
  While the currently scanned symbol is not an end-marker, do the following:
   Simulate a classical coin flip. If the result is heads, move right. Otherwise, move left.
 If the process ended at the right end-marker both times, and two more coin flips both turn out heads, goto EXIT.
 Move the head to the left end-marker, and goto LOOP.
EXIT:
 ACCEPT the input.
```

Figure 3.2: A 2qcfa

After an easy deterministic check, this 2qcfa enters an infinite loop where each iteration compares the number of a's (i) with the number of b's (j). This is achieved by first rotating the qubit counterclockwise for exactly as many times as there are a's, then rotating it clockwise for the b's, and finally checking whether it has returned to its original orientation $|q_0\rangle$. Since the rotation angle is an irrational multiple of π, the probability r_i that the machine will reject at the line marked (I) in Figure 3.2 is zero if and only if there are exactly as many a's as there are b's. Otherwise, r_i will be at least (see the Ambainis-Watrous paper)

$$\frac{1}{2(i-j)^2} > \frac{1}{2(i+j)^2}.$$

We therefore conclude that any input string w with unequal numbers of a's and b's which has survived the deterministic check in the beginning will be rejected with a probability greater than $\frac{1}{2|w|^2}$ in each iteration of the infinite loop.

If the input w has not been rejected after the rotation procedure described above, the 2qcfa makes two consecutive random walks starting on the first input symbol, and ending at an end-marker. The probability that both these walks will end at the right end-marker, and the subsequent coin tosses both yield heads, leading to acceptance in this iteration of the infinite loop, is $\frac{1}{4(|w|+1)^2}$, and the expected runtime for this stage is $O(|w|^2)$. This means that the machine will halt within $O(|w|^2)$ expected iterations of the loop, leading to an overall expected runtime of $O(|w|^4)$.

To conclude, the 2qcfa of Figure 3.2 will accept any string in $\{a^n b^n : n \geq 1\}$ with probability 1. On the other hand, any w not in this language that makes it into the loop has a rejection probability that is more than twice as large as its acceptance probability in each iteration, and therefore will be rejected with probability greater than $\frac{2}{3}$.

3.5 One-way quantum finite (?) automata

An older model of quantum finite automaton works by defining the whole system (including the tape head) to be made of "quantum material", so that the machine can be in the quantum superposition of several configurations with different head positions. Since it is not very fashionable nowadays, we will just give a quick example of this model that illustrates the quantum Fourier transform.

Say that the head can move to the right, or stay put on the same square as commanded by the program at each step. We already know that 2pfa's recognize only regular languages in polynomial time, so it is trivial that the same can be said for 1pfa's. To see that 1QFAs can do more than 1PFAs, we will be demonstrating the ability of the former to recognize a language which can not be recognized by the latter. Let's consider the following language.

$$L_{a=b}\{w|number\ of\ a's\ in\ w = number\ of\ b's\ in\ w\}$$

$L_{a=b}$ is a non-regular language and hence it can not be recognized by a 1PFA with bounded error. In the following we will claim that it is possible to construct a 1QFA for recognizing $L_{a=b}$ with bounded error and show how to do so.

Let $\mathcal{M}$ be a 1QFA which on input w performs the following steps

– At the beginning of the execution (while the input head is pointing to the left end-marker ¢) branch into N distinct paths (all with amplitude $1/\sqrt{N}$) and move the head to the first character of w.

- On each input symbol, the j'th path will
  - wait j steps if the tape head is pointing to an a or
  - wait $N - j + 1$ steps if the tape is pointing to a b.
- When the right end-marker ($ character) is reached, each of the paths will perform an N-way Quantum Fourier Transform, shown below for path j:

$$\tfrac{1}{\sqrt{N}} \textstyle\sum_{k=1}^{N} e^{\frac{2\pi i}{N}jk} |P_k\rangle$$

- Accept w if the state is P_N, reject otherwise.

Note that the delays for each path are arranged so that when $w \notin L_{a=b}$ each of the N branches of $\mathcal{M}$ will encounter $ at separate steps so that no two of them will make QFT at the same time. On the other hand if $w \in L_{a=b}$ all of the N branches of $\mathcal{M}$ will encounter $ at the same time and hence will apply QFT together.

What QFT does is to transform this superposition so that w will be accepted with probability 1 if all branches apply QFT at the same time (this is the case where $w \in L_{a=b}$) and it will be accepted with probability $1/N$ if the branches apply QFT separately(this is the case where $w \notin L_{a=b}$)

Here is an example on how QFT works for $N = 2$

$$\frac{1}{\sqrt{N}} \sum_{k=1}^{N} e^{\frac{2\pi i}{N}jk} |P_k\rangle = \frac{1}{\sqrt{2}} \sum_{k=1}^{2} e^{\frac{2\pi i}{2}jk} |P_k\rangle$$

$$= \frac{1}{\sqrt{2}} e^{\frac{2\pi i}{2}j1} |P_1\rangle + \frac{1}{\sqrt{2}} e^{\frac{2\pi i}{2}j2} |P_2\rangle$$

$$j = 1 \rightarrow \frac{-1}{\sqrt{2}} |P_1\rangle + \frac{1}{\sqrt{2}} |P_2\rangle,$$

$$j = 2 \rightarrow \frac{1}{\sqrt{2}} |P_1\rangle + \frac{1}{\sqrt{2}} |P_2\rangle,$$

If the two branches apply the QFTs at the same step the non-accepting state ($|P_1\rangle$) will disappear from the superposition, and only the accepting state ($|P_2\rangle$) will remain alive. Therefore the strings which bring different paths to the tape end simultaneously will be accepted with probability 1. On the other hand if the two arrive the tape end at different steps each individual branch will accept the string with probability $= 1/2$

If $\mathcal{M}$ can be constructed as defined here it accepts $w \in L_{a=b}$ with probability $= 1$ and it will reject $w \notin L_{a=b}$ with probability $= (N-1)/N$. Therefore it recognizes $L_{a=b}$ with one-sided bounded error. (Note that the error term can be made arbitrarily small with increasing N with also increasing time and space cost, although runtime is still linear and memory used (if you forget about the head) is still constant.)

But what if we want only real amplitudes, rather than ones with imaginary components? No worries. Complex numbers can be represented by 2X2 matrices of reals. It is easy to simulate the complex numbers with use of real numbers by replacing every complex number $a + bi$ with the real valued 2×2 matrix

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix}$$

in all our transition matrices. Our machines could have been defined with complex amplitudes all along, with the necessary generalizations of the well-formedness conditions.

Chapter 4

$\omega(1)$ space

We get the definitions of probabilistic Turing machines in the very same way that we modified the definitions of real-time and two-way finite automata to obtain the definitions of their probabilistic counterparts.

My current favorite model of quantum Turing machines is obtained by adding a quantum tape to an otherwise deterministic TM with an input tape and a classical work tape, in the same way we added a quantum register to a classical machine to obtain a 2qcfa (Figure 13 in https://cs.uwaterloo.ca/ watrous/Papers/QuantumComputationalComplexity.pdf).

This machine executes superoperators[1] on the quantum systems (which are really like the constant-memory machines we are used to until now) that are currently pointed to by its heads, as determined by the current classical state, and information scanned on the input and classical work tapes. The outcome of this superoperator then plays a role in the next classical move, which can also move the quantum heads.

Why three heads? Until now, whenever we constructed some well-formed superoperator for our finite automata, let's say on 10 states, we just assumed that some kind person in the quantum physics lab would implement it physically, working out all the angles, polarizations and whatnot. This is not fair, as it assumes that this job is equally easy for all matrices. A better approach is to allow ourselves to use only a fixed number of "elementary" operators, or "gates", and let the burden of specifying how all the quantum transformations will be realized in terms of these gates fall on the program-

[1] In fact, all superoperators in the algorithms to be discussed in this chapter are actually unitary operators. These two models are equivalent for time-bounded computations up to a polynomial overhead.

mer (i.e. us). (It is similar to what's going on with Boolean circuits and the "universal" sets of gates like $\{AND, NOT\}$.) A set of quantum gates is said to be universal if any quantum transformation on any number of qubits can be approximated by the application of a particular sequence of these gates on selected qubits.[2] Among many others, the set consisting of the Toffoli and Hadamard gates is known to be universal:

The Hadamard operation works on a single quantum bit, with the 2X2 transition

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix},$$

and the Toffoli operation works on three qubits, with transition

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix},$$

(so when you perform it on three bits with values x, y and z, it transforms them to x, y, and (z XOR (x AND y))). (I'm assuming that you are familiar with tensor products. If you have a system of four qubits, and you apply Hadamard to the first one, and Toffoli to the second, third and fourth ones in that order, say, just take the tensor product of these two matrices to see what the overall transformation on the whole system looks like.)

(Toffoli alone is known to be universal for classical computation, so if you need to implement a classical subroutine on the quantum tape, you can do it.) We have three heads on the quantum tape since Toffoli is a three-qubit gate. We assume that the qubits in each square of the quantum tape are originally set to $|0\rangle$.

The two most famous quantum algorithms are Grover's and Shor's algorithms. They can be described in terms of QTMs, or in other equivalent formalisms.

[2] The classical algorithm described in http://arxiv.org/pdf/quant-ph/0505030v2.pdf can be used to compile a quantum program using arbitrary single-qubit operations to one approximating it using only operators from a fixed set efficiently.

4.1 Grover's Search Algorithm

Determining whether a given boolean formula is satisfiable or not is an NP-Complete problem. The corresponding language recognition problem refers to the language below:

$$SAT = \{f | f \text{ is a satisfiable boolean formula } \}$$

The best known classical algorithm for solving SAT runs in $poly(n)2^n$ time where n denotes the number of boolean variables occurring in the input formula. Grover's algorithm to be introduced below runs in $poly(n)2^{n/2}$ time and hence it provides a quadratic speedup over the best known classical algorithm for a related task.

Theorem 5. *There is a quantum algorithm that given any polynomial time computable function $f : \{0,1\}^n \rightarrow \{0,1\}$ (i.e., represented as a circuit computing f) as input, where $f(a) = 1$ for just one input string a, returns a in $poly(n)2^{n/2}$ time.*

Proof. Before giving the details of the algorithm we will be defining two vectors.

- $u = \frac{1}{2^{n/2}} \sum_{w \in \{0,1\}^n} |w\rangle$ represents the equal superposition of the vectors for all strings of length n

- $e = \sum_{w \in \{0,1\}^n, w \neq a} |w\rangle$ represents the sum of vectors for all strings of length n which are mapped to 0 by the function f.

Note that $|a\rangle$ and e are orthogonal. The two-dimensional space spanned by $|a\rangle$ and e will be useful in visualization of the working of the algorithm.

The algorithm runs on a quantum register of $n + 1 + m$ qubits where the last m bits are the auxiliary bits used in the quantum implementation of the circuit part for computing f, and since f is polynomially computable, m is not large. Therefore the algorithm will be described on the first $n + 1$ bits initially prepared in the state $|0^{n+1}\rangle$. In the following x will be used to denote the state of the first n qubits and σ will be used to denote the state of the $(n + 1)$st qubit. See Algorithm 1 for the details of Grover's search.

Steps 1.2 and 2.2.2 can be achieved by a one-qubit operation which leaves the n+1'st bit unchanged if it is initially 0, and multiplies it by -1 if it is 1.

Algorithm 1 Grover's Search Algorithm

```
Runs on $n+1$ qubit register initially prepared to $|0^n 0\rangle$.

• Initialization: Apply Hadamard transformation on the first n qubits.
• Loop: Repeat $Round(\frac{arccos(\frac{1}{2^{n/2}})}{2arcsin(\frac{1}{2^{n/2}})})$ times:
    - Step 1: Reflect the vector in the first n qubits around $e = \sum_{w \in \{0,1\}^n, w \neq a} |w\rangle$:
        - 1.1: On the first n+1 qubits, compute $|x\sigma\rangle \rightarrow |x(\sigma \oplus f(x))\rangle$
        - 1.2: If $\sigma = 1$ then multiply $|x\rangle$ by $-1$
        - 1.3: On the first n+1 qubits, compute $|x\sigma\rangle \rightarrow |x(\sigma \oplus f(x))\rangle$
    - Step 2: Reflect the vector in the first n qubits around $u = \frac{1}{2^{n/2}} \sum_{w \in \{0,1\}^n} |w\rangle$:
        - 2.1: Apply Hadamard transformation on the first n qubits.
        - 2.2: Reflect around $|0^n\rangle$:
            - 2.2.1: If the first n qubits are not all zero, flip the n+1st qubit
            - 2.2.2: If $\sigma = 1$ then multiply $|x\rangle$ by $-1$
            - 2.2.3: If the first n qubits are not all zero, flip the n+1st qubit
        - 2.3: Apply Hadamard transformation on the first n qubits.
• Measurement: Measure $x$ and compute $f(x)$. If the result is 1, return x. Otherwise repeat the algorithm.
```

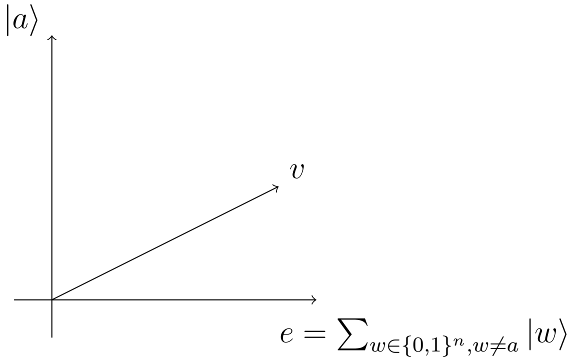


Figure 4.1: Two dimensional space spanned by $|a\rangle$ and e .

In the following parts we will be showing how Grover's algorithm operates on the state vectors and how it comes up with a satisfying assignment. We will visualize the change in the state vectors represented in the two dimensional space spanned by $|a\rangle$ and e. In doing so we will use

- $\mathbf{x_{i,0}}$ to denote the state of the first n bits at the beginning of the i'th iteration of the algorithm,
- $\mathbf{x_{i,1}}$ to denote the state of the first n bits at i'th iteration after step 1 and
- $\mathbf{x_{i,2}}$ to denote the state of the first n bits at i'th iteration after step 2.

Let's see how x evolves during the execution of the algorithm. At the beginning the first n bits are initialized to the equal superposition of all strings of length n by Hadamard transformation. Therefore we have $x_{1,0} = u$.

Then comes the first iteration of the loop where the states $x_{1,1}$ and $x_{1,2}$ are computed as shown in Figure 4.3.

Figure 4.3 suggests that first iteration of the Grover loop which moves the state x of the first n bits from $x_{1,0}$ to $x_{1,2}$ cuts the angular distance between a and x by 2θ. Figure 4.4 will show that further iterations of the loop will continue to do the same.

Figure 4.3 and Figure 4.4 provide sufficient means to conclude that each iteration of the Grover loop will decrease the angle (which is originally

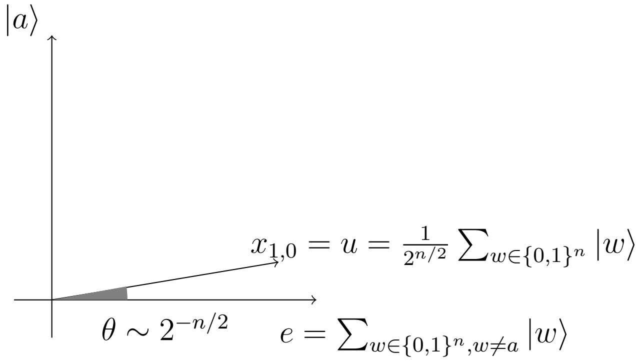
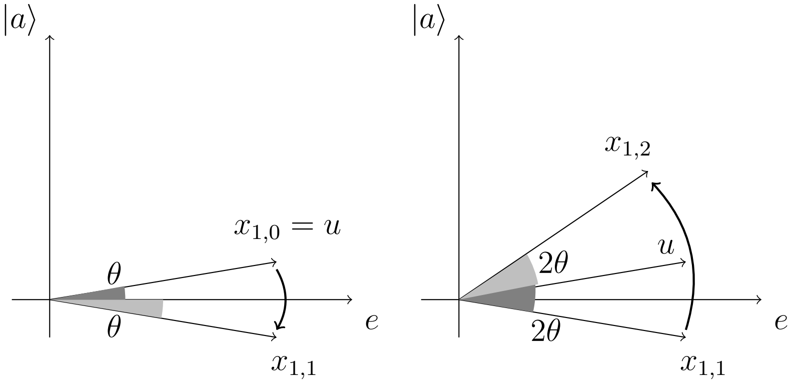

$|a\rangle$
$x_{1,0} = u = \frac{1}{2^{n/2}} \sum_{w \in \{0,1\}^n} |w\rangle$
$\theta \sim 2^{-n/2}$
$e = \sum_{w \in \{0,1\}^n, w \neq a} |w\rangle$

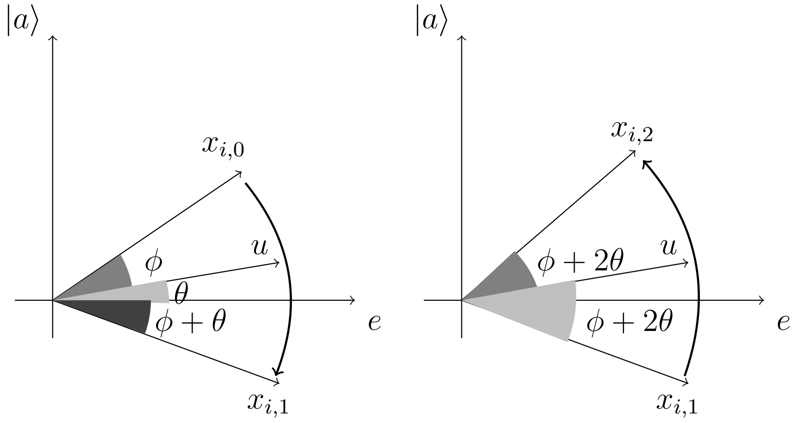
Figure 4.2: x is initialized to equal superposition of all strings of length n.



(a) Step 1: Reflect around e

(b) Step 2: Reflect around u

Figure 4.3: The first iteration of the Grover loop



(a) Step 1: Reflect around e (b) Step 2: Reflect around u

Figure 4.4: The i'th iteration of the Grover loop

arccos($\frac{1}{2^{n/2}}$))between a and x by 2θ radians. In order to maximize the probability of observing a in the measurement step one should terminate the loop at the point where distance between x and a is as small as possible.

For large values of n, the number of iterations is $O(2^{n/2})$ (the numerator tends to $\pi/2$, whereas the denominator tends to $2\frac{1}{2^{n/2}}$, since sin(x)=x for small x). □

4.2 Shor's Algorithm

In 1994, Peter Shor formulated a quantum algorithm for integer factorization: Given an integer N, which is promised to be the product of at most two prime numbers, find those primes.

The algorithm has two parts:

1. A reduction, which can be efficiently done on a classical computer, of the factoring problem to the problem of order finding.

2. A quantum algorithm to solve the order finding problem.

Algorithm 2 Reduction of factoring to order finding

Given N, find a prime factor of N:

- **Step 1:** Check if N is a prime itself, if so print N.
- **Step 2:** Check if N is a perfect power, i.e. $N = C^D$ for some $C, D \in \mathbb{Z}$. If so, call this algorithm on C.
- **Step 3:** Randomly choose A in $[2, \dots, N-1]$
- **Step 4:** g=gcd(A,N). If $g > 1$, call this algorithm on g.
- **Step 5:** Run Shor's algorithm on A and N to obtain a number r such that $A^r \equiv 1 (modN)$.
- **Step 6:** If r is odd, then go to step 3 to pick another A.
- **Step 7:** Let $y = A^{\frac{r}{2}} (modN)$, if $y = -1$, then go to step 3.
- **Step 8:** Let $z = gcd(y-1, N)$, if $z = 1$ then r was not the order, go to step 5 to try to obtain a correct r.
- **Step 9:** Run this algorithm on z.

4.2.1 Factoring reduces to order finding

For step 1, we know that $PRIMES \in P$, so we can efficiently check the primality of N on a classical computer. In step 2, we know that the binary representation of our input N has length $n = log(N)$ and if $N = C^D$, then we know that $n = Dlog(C)$, therefore the value of D can be at most the length of our input, thus it can be computed efficiently that if N is a perfect power by exhaustively going through all candidate values of D and using binary search to see if the D'th root for this D is an integer. For step 5, we want to find the smallest r (order), such that $r \leq N$.

Lemma 1. *For every N and $y \in \{1, \ldots, N-1\}$, if $y^2 = 1(modN)$ but $y(modN) \notin \{-1, +1\}$, then $gcd(y-1, N) \notin \{1, N\}$.*

Proof. N divides $y^2 - 1 = (y+1)(y-1)$ but does not divide neither (y-1) nor (y+1). However, this means $gcd(y-1, N) > 1$, since if y-1 and N were co-primes, then since N divides (y-1)(y+1), it would have to divide (y+1). □

Lemma 2. *For every nonprime N that is not a prime power, the probability that a random element A of $\mathcal{Z}_N^* = \{A \in \{0, \ldots, N-1\} | gcd(A, N) = 1\}$ has an even order r and furthermore, $A^{\frac{r}{2}} \neq -1(modN)$, is at least $\frac{1}{4}$.*

Proof. If N=PQ where P and Q are primes, following the Chinese Remainder Theorem, we know that choosing a random $A \in \mathcal{Z}_N^*$ is equivalent to choosing two random numbers Y,Z in $\mathcal{Z}_P^*$ and $\mathcal{Z}_Q^*$ respectively, and setting A to be the unique number corresponding to the pair $< Y, Z >$. For every k, $A^k(modN)$ is isomorphic to $< Y^k \mod P, Z^k \mod Q >$ and so the order of A is the least common multiple of the orders of Y and Z modulo P and Q respectively.

Let us show that;

- **A** - with probability at least $\frac{1}{2}$, the order of Y is even : $2^k.c$ for $k \geq 1$ and c is odd.
- **B** - with probability at least $\frac{1}{2}$, the order of Z has the form $2^l.d$ for d is odd and $l \neq k$.

Proof. **A** - The set of numbers in $\mathcal{Z}_P^*$ with odd order is a subgroup of $\mathcal{Z}_P^*$, since if Y,Y' have odd orders r,r', then $< Y, Y' >^{rr'} = 1(modP)$ which means that the order of YY' divides the odd number rr'. Consider -1 which has even order, so not everyone has odd order. $\mathcal{Z}_P^*$ with odd order is a proper subgroup, meaning at most $\frac{1}{2}$ of the population will be odd. □

Proof. **B** - For every l, define G_l to be the subset of $\mathcal{Z}_Q^*$ whose order modulo Q is of the form $2^j.d$ where $j \leq l$ and d is odd. For every l, G_l is a subgroup of G_{l+1}. Since Q is prime, the mapping $x \rightarrow x^2$ (mod Q) is two-to-one and maps G_{l+1} to G_l, so;

$$|G_l| \geq \frac{|G_{l+1}|}{2}$$

Let l_0 name the largest number such that G_{l_0} is a proper subgroup of $\mathcal{Z}_Q^*$. Then

$$|G_{l_0}| = \frac{|\mathcal{Z}_Q^*|}{2}.$$

So with probability exactly $\frac{1}{2}$, the order of a random $\mathcal{Z} \in \mathcal{Z}_Q^*$ is a number of the form $2^l.d$ for $l \leq l_0$. We conclude that for a specific k, the order being of the form $2^k.d$ has probability at most $\frac{1}{2}$. □

This implies that, with probability at least $\frac{1}{4}$, the order of A is $r = 2^{max(k,l)}.lcm(c,d)$, for $k \neq l$. $A^{\frac{r}{2}}$ will therefore be isomorphic to a pair which has at least one 1 in it. We know that -1 is isomorphic to the pair $< -1, -1 >$, which leads to a contradiction for the case that $A^{\frac{r}{2}} = -1(modN)$.

This argument can be generalized to any composite N which is not a prime power. □

4.2.2 Order finding

We can now introduce Shor's actual contribution, the polynomial time order finding algorithm. (Consult the pseudocode.)

Modular exponentiation has an efficient classical algorithm.

We know the Fourier Transform:

$$\tilde{f}(x) = \frac{1}{\sqrt{M}} \sum_{y \in \mathcal{Z}_M} f(y)\omega^{xy}$$

where $\omega = e^{\frac{2\pi i}{M}}$, an M^{th} root of unity.

The quantum version does this transformation on the amplitudes, mapping $\sum_{x\in\mathcal{Z}_M} f(x)|x\rangle$ to $\sum_{x\in\mathcal{Z}_M} \tilde{f}(x)|x\rangle$.

Algorithm 3 Shor's algorithm for order finding

Given numbers A and N, such that $A < N$, N is written in n bits, and $gcd(A, N) = 1$, find a number r, such that $A^r = 1(modN)$. We will call the first $m = \lceil 5log(N) \rceil$ bits of the tape 'the first register'. The n bits after that are called 'the second register'. Let $M = 2^m$.

- **Step 0:** Initialize m+n qubits to $|0^m 0^n\rangle$.
- **Step 1:** Apply Hadamards to each qubit in the first register.
- **Step 2:** Compute modular exponentiation, i.e. $|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus (A^x \bmod N\rangle$.
- **Step 3:** Measure the second register to obtain a y_0.
- **Step 4:** Apply Quantum Fourier Transform to the first register.
- **Step 5:** Observe the first register to obtain an x. Let z=0.
- **Step 6:** Find a rational approximation $\frac{a}{b}$ with $gcd(a, b) = 1$ and $b < N$, that approximates the number $\frac{x}{2^{m-z}}$ within $\frac{1}{10 \times 2^{m-z}}$ accuracy. If this approximation satisfies $A^{2^z b} = 1(modN)$, then output $2^z b$ and terminate. If $z > 3n$, terminate. Otherwise, increment z and go to the beginning of Step 6.

The QFT on m bits ($2^m = M$ states) can be implemented efficiently by noting that its action on a basis state

$$|x\rangle \to \frac{1}{\sqrt{M}} \sum_{y=0}^{M-1} e^{\frac{2\pi ixy}{N}} |y\rangle$$

can be rewritten in terms of the action on individual bits as

$$|x_1, \cdots, x_n\rangle \to \frac{(|0\rangle + e^{2\pi i 0.x_n}|1\rangle)(|0\rangle + e^{2\pi i 0.x_{n-1}x_n}|1\rangle) \cdots (|0\rangle + e^{2\pi i 0.x_1x_2\cdots x_n}|1\rangle)}{2^{\frac{m}{2}}}$$

This formulation leads to an implementation using gates of the form

$$R_k = \left(\begin{array}{cc} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{array} \right),$$

which was shown in class. The Solovay-Kitaev theorem guarantees the existence of an algorithm which we can use to approximate these gates efficiently in terms of our fixed set of quantum operations.

Step 6 is classical. The efficient continued fractions algorithm generates a sequence of better and better approximations of the form a/b with bigger and bigger values of b to the number x/2^{m-z} as long as $b < N$.

Step 1 computes

$$\frac{1}{\sqrt{M}} \sum_{x \in \mathcal{Z}_M} |x\rangle|0^n\rangle$$

after which every amplitude is set to $\frac{1}{\sqrt{M}}$.

In step 2, we're doing the modular exponentiation, which transforms the second register by computing

$$\frac{1}{\sqrt{M}} \sum_{x \in \mathcal{Z}_M} |x\rangle|A^x(modN)\rangle$$

After the step 3, i.e. the measurement of the second register, the first register partially collapses

$$\frac{1}{\sqrt{K}} \sum_{l=0}^{K-1} |x_0 + lr\rangle|y_0\rangle$$

such that x_0 is the smallest number such that $A^{x_0} = y_0(modN)$ and $K = \lfloor \frac{M-1-x_0}{r} \rfloor + 1,$.

After the Quantum Fourier Transform in step 4, we have

$$\frac{1}{\sqrt{MK}}(\sum_{x\in\mathcal{Z}_M}\sum_{l=0}^{K-1}\omega^{(x_0+lr)x}|x\rangle)|y_0\rangle$$

Observing the first register, we obtain a value x. There are two cases here:

1 - r divides M.

If M=rc for some integer c, then the absolute value of the amplitude of $|x\rangle$ before the measurement is $\frac{1}{\sqrt{Mc}}|\sum_{l=0}^{c-1}\omega^{lrx}|$. If c does not divide x, then $\sum_{l=0}^{c-1}\omega^{lrx} = 0$ (since ω^r is a c'th root of unity by our assumptions, and the formula for the sum of a finite geometric series can be applied since the denominator is not zero). If x=ca for some a, then $\sum_{l=0}^{c-1}\omega^{lrx} = \sum_{l=0}^{c-1}\omega^{lrca} = \sum_{l=0}^{c-1}\omega^{lMa} = c$. For all the r possible values in $\{0, 1, \cdots, r-1\}$ of a, the probability that such an x will be observed is thus c/M=1/r.

So the observed x will satisfy x/M=a/r for some randomly selected a in that set. Since there are $\Omega(\frac{r}{logr})$ prime numbers less than r, and at most $\log r$ of these are factors of r, we can put a lower bound on the probability that a/r will not be "simplified", and the continued fractions algorithm will print r.

2 - The measured value x doesn't satisfy that M divides xr.

Lemma 3. *There exists $\Omega(\frac{r}{\log r})$ values $x \in \mathcal{Z}_M$ such that*

1. $0 < xr'(mod\ M') < \frac{r'}{10}$
2. $\lfloor\frac{xr'}{M'}\rfloor$ *and r' are coprime,*

where $r' = r/gcd(r, M)$ and $M' = M/gcd(r, M)$.

Proof. Let $gcd(r, M) = d$. Then $r' = r/d$ and $M' = M/d$, and $x \to r'x(mod$ M') is a permutation of $\mathcal{Z}'_M$. There are at least $\Omega(\frac{r}{d\log r})$ numbers in $[1, \ldots, \frac{r'}{10}]$ that are coprime to r'. So at least $\Omega(\frac{r}{d\log r})$ numbers x exist such that $r'x(modM') = xr' - \lfloor\frac{xr'}{M'}\rfloor M'$ is in $[1, \ldots, \frac{r'}{10}]$ and is coprime to r', and for these values of x, $\lfloor\frac{xr'}{M'}\rfloor$ and r' have to be coprime, since if these two have a shared factor, that factor would be shared with $r'x(mod$ M') as well.

Since r'x = r'(x+cM') modulo M' for any c, we have a total of $\Omega(\frac{r}{\log r})$ such numbers $x \in \mathcal{Z}_M$ with $xr'(mod\ \text{M'}) < \frac{r'}{10}$. □

Lemma 4. *If x satisfies* $0 < xr'(mod\ M') < \frac{r'}{10}$*, where r' is a factor of r greater than 1, then the coefficient of* $|x\rangle$ *is at least* $\Omega(\frac{1}{\sqrt{r}})$*.*

Proof. The absolute value of $|x\rangle$'s amplitude is

$$\frac{1}{\sqrt{MK}}|\sum_{l=0}^{K-1}\omega^{lrx}| \tag{4.1}$$

We have $\frac{M}{2r} < K \leq \frac{M}{r}$ since $x_0 < N \ll M$.

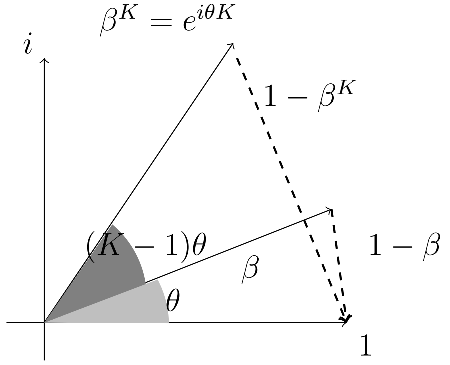
Let $\beta = \omega^{rx}$. Note that $\beta \neq 1$, since x doesn't satisfy that M divides xr. Using the formula for the sum of a geometric series, (4.1) is at least

$$\frac{\sqrt{r}}{M}|\frac{1-\beta^K}{1-\beta}|,$$

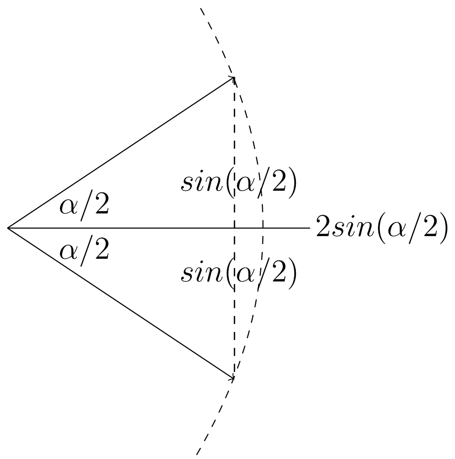
and at most

$$\frac{\sqrt{2r}}{M}|\frac{1-\beta^K}{1-\beta}|.$$

Letting $\theta = 2\pi\frac{rx(modM)}{M}$ be the angle such that $\beta = e^{i\theta}$, the following figure shows the relationship between these variables in the complex plane:



Furthermore, it can be seen from the figure below that if the angle between the lines decreases, the length of the chord length approximates $2sin\frac{\alpha}{2}$.



But then the amplitude under consideration is

$$\Theta(\frac{\sqrt{r}}{M}\frac{sin(\frac{K\theta}{2})}{sin(\frac{\theta}{2})}).$$

Note that, if $xr'(mod \text{ M'}) < \frac{r'}{10}$, then $xr(mod \text{ M}) < \frac{r}{10}$. Since $K \leq M/r$, $\theta = 2\pi\frac{rx(modM)}{M}$, and $xr(mod \text{ M}) < \frac{r}{10}$, we have $K\theta/2 < \pi/10$. Since small angles can be approximated with their sines, the amplitude is $\Theta(\frac{\sqrt{r}}{M}\lceil\frac{M}{r}\rceil) = \Theta(\frac{1}{\sqrt{r}})$. □

Combining these two lemmas, we conclude that, with probability $\Omega(\frac{1}{\log r})$, the measured value of x will satisfy $xr' - cM' < r'/10$ for $c = \lfloor\frac{xr'}{M'}\rfloor$, meaning $|\frac{x}{M'} - \frac{c}{r'}| < \frac{1}{10M}$.

So if we hit on the correct value of M' (which is $M/2^z$ for some z, recalling that M is a power of 2) by dividing M repeatedly by 2, c/r' won't "simplify further" and will approximate x/M' closely enough that it will be the fraction found by the continued fractions algorithm when it runs. Since r equals $2^z r'$, it will be found as a result of the subsequent checks. Since $z = O(\log N)$, the repetitions add up to polynomial runtime.

[1]

Bibliography

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.