Lecture Notes for Ph219/CS219: Quantum Information and Computation Chapter 5

John Preskill
California Institute of Technology

Updated July 2015

Contents

5
Classical and quantum circuits

5.1 Classical Circuits

The concept of a quantum computer was introduced in Chapter 1. Here we will specify our model of quantum computation more precisely, and we will point out some basic properties of the model; later we will investigate the power of the model. But before we explain what a quantum computer does, we should say what a classical computer does.

5.1.1 Universal gates

A (deterministic) classical computer evaluates a function: given n-bits of input it produces m-bits of output that are uniquely determined by the input; that is, it finds the value of the function

$$f : \{0,1\}^n \to \{0,1\}^m, \tag{5.1}$$

for a particular specified n-bit argument x. A function with an m-bit output is equivalent to m functions, each with a one-bit output, so we may just as well say that the basic task performed by a computer is the evaluation of

$$f : \{0,1\}^n \to \{0,1\}. \tag{5.2}$$

A function talking an n-bit input to a one-bit output is called a Boolean function. We may think of such a function as a binary string of length 2^n, where each bit of the string is the output $f(x)$ for one of the 2^n possible values of the input x. Evidently, there are 2^{2^n} such strings; that's a lot of functions! Already for $n = 5$ there are $2^{32} \approx 4.3 \times 10^9$ Boolean functions — you've encountered only a tiny fraction of these in your lifetime.

It is sometimes useful to regard a Boolean function as a subset Σ of the n-bit strings containing those values of the input x such that $f(x) = 1$; we

say these strings are "accepted" by f. The complementary set $\bar{\Sigma}$ contains values of x such that $f(x) = 0$, which we say are "rejected" by f.

The evaluation of a Boolean function f can be reduced to a sequence of simple logical operations. To see how, denote the n-bit strings accepted by f as $\Sigma = \{x^{(1)}, x^{(2)}, x^{(3)}), \dots\}$ and note that for each $x^{(a)}$ we can define a function $f^{(a)} : \{0,1\}^n \to \{0,1\}$ such that

$$f^{(a)}(x) = \begin{cases} 1 & x = x^{(a)} \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Then f can be expressed as

$$f(x) = f^{(1)}(x) \vee f^{(2)}(x) \vee f^{(3)}(x) \vee \dots, \tag{5.4}$$

the logical OR ($\vee$) of all the $f^{(a)}$'s. In binary arithmetic the $\vee$ operation of two bits may be represented

$$x \vee y = x + y - x \cdot y; \tag{5.5}$$

it has the value 0 if x and y are both zero, and the value 1 otherwise.

Now consider the evaluation of $f^{(a)}$. We express the n-bit string x as

$$x = x_{n-1}x_{n-2}\dots x_2x_1x_0. \tag{5.6}$$

In the case where $x^{(a)} = 11\dots111$, we may write

$$f^{(a)}(x) = x_{n-1} \wedge x_{n-2} \wedge \dots \wedge x_2 \wedge x_1 \wedge x_0; \tag{5.7}$$

it is the logical AND ($\wedge$) of all n bits. In binary arithmetic, the AND is the product

$$x \wedge y = x \cdot y. \tag{5.8}$$

For any other $x^{(a)}$, $f^{(a)}$ is again obtained as the AND of n bits, but where the NOT ($\neg$) operation is first applied to each x_i such that $x_i^{(a)} = 0$; for example

$$f^{(a)}(x) = \dots(\neg x_3) \wedge x_2 \wedge x_1 \wedge (\neg x_0) \tag{5.9}$$

if

$$x^{(a)} = \dots0110. \tag{5.10}$$

The NOT operation is represented in binary arithmetic as

$$\neg x = 1 - x. \tag{5.11}$$

We have now constructed the function $f(x)$ from three elementary logical connectives: NOT, AND, OR. The expression we obtained is called the "disjunctive normal form" (DNF) of $f(x)$. We have also implicitly used another operation INPUT(x_i), which inputs the ith bit of x.

These considerations motivate the circuit model of computation. A computer has a few basic components that can perform elementary operations on bits or pairs of bits, such as NOT, AND, OR. It can also input a variable bit or prepare a constant bit. A computation is a finite sequence of such operations, a *circuit*, applied to a specified string of input bits. Each operation is called a *gate*. The result of the computation is the final value of all remaining bits, after all the elementary operations have been executed. For a Boolean function (with a one-bit output), if there are multiple bits still remaining at the end of the computation, one is designated as the output bit. A circuit can be regarded as a directed acyclic graph, where each vertex in the graph is a gate, and the directed edges indicate the flow of bits through the circuit, with the direction specifying the order in which gates are applied. By acyclic we mean that no directed closed loops are permitted.

We say that the gate set {NOT, AND, OR, INPUT} is "universal," meaning that any function can be evaluated by building a circuit from these components. It is a remarkable fact about the world that an arbitrary computation can be performed using such simple building blocks.

5.1.2 Most functions require large circuits

Our DNF construction shows that any Boolean function with an n-bit input can be evaluated using no more than 2^n OR gates, $n2^n$ AND gates, $n2^n$ NOT gates, and $n2^n$ INPUT gates, a total of $(3n+1)2^n$ gates. Of course, some functions can be computed using much smaller circuits, but for *most* Boolean functions the smallest circuit that evaluates the function really does have an exponentially large (in n) number of gates. The point is that if the circuit size (*i.e.*, number of gates) is subexponential in n, then there are many, many more functions than circuits.

How many circuits are there with G gates acting on an n-bit input? Consider the gate set from which we constructed the DNF, where we will also allow inputting of a constant bit (either 0 or 1) in case we want to use some scratch space when we compute. Then there are $n+5$ different gates: NOT, AND, OR, INPUT(0), INPUT(1), and INPUT(x_i) for $i = 0, 1, 2, \ldots n-1$. Each two-qubit gate acts on a pair of bits which are outputs from preceding gates; this pair can be chosen in fewer than G^2 ways. Therefore the total number of size-G circuits can be bounded as

$$N_{\text{circuit}}(G) \leq \left((n+5)G^2\right)^G . \tag{5.12}$$

If $G = c\frac{2^n}{2n}$, where c is a constant independent of n, then

$$\begin{aligned}\log_2 N_{\text{circuit}}(G) &\leq G\left(\log_2(n+5) + 2\log_2 G\right) \\ &= c2^n\left(1 + \frac{1}{2n}\log_2\left(\frac{c^2(n+5)}{4n^2}\right)\right) \leq c2^n,\end{aligned} \tag{5.13}$$

where the second inequality holds for n sufficiently large. Comparing with the number of Boolean functions, $N_{\text{function}}(n) = 2^{2^n}$, we find

$$\log_2\left(\frac{N_{\text{circuit}}(G)}{N_{\text{function}}(n)}\right) \leq (c-1)2^n \tag{5.14}$$

for n sufficiently large. Therefore, for any $c < 1$, the number of circuits is smaller than the number of functions by a huge factor. We did this analysis for one particular universal gate set, but the counting would not have been substantially different if we had used a different gate set instead.

We conclude that for any positive ε, then, most Boolean functions require circuits with at least $(1-\varepsilon)\frac{2^n}{2n}$ gates. The circuit size is so large because most functions have no structure that can be exploited to construct a more compact circuit. We can't do much better than consulting a "look-up table" that stores a list of all accepted strings, which is essentially what the DNF does.

5.1.3 Circuit complexity

So far, we have only considered a computation that acts on an input with a fixed (n-bit) size, but we may also consider *families* of circuits that act on inputs of variable size. Circuit families provide a useful scheme for analyzing and classifying the *complexity* of computations, a scheme that will have a natural generalization when we turn to quantum computation.

Boolean functions arise naturally in the study of complexity. A Boolean function f may be said to encode a solution to a "decision problem" — the function examines the input and issues a YES or NO answer. Often, what might not be stated colloquially as a question with a YES/NO answer can be "repackaged" as a decision problem. For example, the function that defines the FACTORING problem is:

$$f(x,y) = \begin{cases} 1 & \text{if integer } x \text{ has a divisor } z \text{ such that } 1 < z < y, \\ 0 & \text{otherwise;} \end{cases} \tag{5.15}$$

knowing $f(x,y)$ for all $y < x$ is equivalent to knowing the *least* nontrivial factor of x (if there is one).

Another example of a decision problem is the HAMILTONIAN PATH problem: let the input be an ℓ-vertex graph, represented by an $\ell \times \ell$ adjacency matrix (a 1 in the ij entry means there is an edge linking vertices i and j); the function is

$$f(x) = \begin{cases} 1 & \text{if graph } x \text{ has a Hamiltonian path,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

A path on the graph is Hamiltonian if it visits each vertex exactly once.

For the FACTORING problem the size of the input is the number of bits needed to specify x and y, while for the HAMILTONIAN PATH problem the size of the input is the number of bits needed to specify the graph. Thus each problem really defines a family of Boolean functions with variable input size. We denote such a function family as

$$f : \{0,1\}^* \to \{0,1\}, \tag{5.17}$$

where the $*$ indicates that the input size is variable. When x is an n-bit string, by writing $f(x)$ we mean the Boolean function in the family which acts on an n-bit input is evaluated for input x. The set L of strings accepted by a function family

$$L = \{x \in \{0,1\}^* : f(x) = 1\} \tag{5.18}$$

is called a *language*.

We can quantify the hardness of a problem by stating how the computational resources needed to solve the problem scale with the input size n. In the circuit model of computation, it is natural to use the circuit size (number of gates) to characterize the required resources. Alternatively, we might be interested in how much *time* it takes to do the computation if many gates can be executed in parallel; the *depth* of a circuit is the number of time steps required, assuming that gates acting on distinct bits can operate simultaneously (that is, the depth is the maximum length of a directed path from the input to the output of the circuit). The *width* of a circuit, the maximum number of gates (including identity gates acting on "resting" bits) that act in any one time step, quantifies the storage space used to execute the computation.

We would like to divide the decision problems into two classes: easy and hard. But where should we draw the line? For this purpose, we consider decision problems with variable input size, where the number of bits of input is n, and examine how the size of the circuit that solves the problem scales with n.

If we use the scaling behavior of a circuit family to characterize the difficulty of a problem, there is a subtlety. It would be cheating to hide the difficulty of the problem in the *design* of the circuit. Therefore, we should

restrict attention to circuit families that have acceptable "uniformity" properties — it must be "easy" to build the circuit with $n+1$ bits of input once we have constructed the circuit with an n-bit input.

Associated with a family of functions $\{f_n\}$ (where f_n has n-bit input) are circuits $\{C_n\}$ that compute the functions. We say that a circuit family $\{C_n\}$ is "polynomial size" if the size $|C_n|$ of C_n grows with n no faster than a power of n,

$$\text{size } (C_n) \leq \text{poly}(n), \tag{5.19}$$

where poly denotes a polynomial. Then we define:

$$\text{P} = \{\text{decision problems solved by polynomial-size uniform circuit families}\}$$

(P for "polynomial time"). Decision problems in P are "easy." The rest are "hard." Notice that C_n computes $f_n(x)$ for every possible n-bit input, and therefore, if a decision problem is in P we can find the answer even for the "worst-case" input using a circuit of size no greater than poly(n). As already noted, we implicitly assume that the circuit family is "uniform" so that the *design* of the circuit can itself be solved by a polynomial-time algorithm. Under this assumption, solvability in polynomial time by a circuit family is equivalent to solvability in polynomial time by a universal Turing machine.

Of course, to determine the size of a circuit that computes f_n, we must know what the elementary components of the circuit are. Fortunately, though, whether a problem lies in P does not depend on what gate set we choose, as long as the gates are universal, the gate set is finite, and each gate acts on a constant number of bits. One universal gate set can *simulate* another efficiently.

The way of distinguishing easy and hard problems may seem rather arbitrary. If $|C_n| \sim n^{1000}$ we might consider the problem to be intractable in practice, even though the scaling is "polynomial," and if $|C_n| \sim n^{\log\log\log n}$ we might consider the problem to be easy in practice, even though the scaling is "superpolynomial." Furthermore, even if $|C_n|$ scales like a modest power of n, the constants in the polynomial could be very large. Such pathological cases seem to be uncommon, however. Usually polynomial scaling is a reliable indicator that solving the problem is feasible.

Of particular interest are decision problems that can be answered by exhibiting an example that is easy to verify. For example, given x and $y < x$, it is hard (in the worst case) to determine if x has a factor less than y. But if someone kindly provides a $z < y$ that divides x, it is easy for us to check that z is indeed a factor of x. Similarly, it is hard to determine if a graph has a Hamiltonian path, but if someone kindly provides a path, it is easy to verify that the path really is Hamiltonian.

This concept that a problem may be hard to solve, but that a solution can be easily verified once found, can be formalized. The complexity class of decision problems for which the answer can be checked efficiently, called NP, is defined as

Definition. NP. *A language L is in NP iff there is a polynomial-size* verifier $V(x, y)$ *such that*

If $x \in L$, then there exists y such that $V(x, y) = 1$ (completeness),

If $x \notin L$, then, for all y, $V(x, y) = 0$ (soundness).

The verifier V is the uniform circuit family that checks the answer. Completeness means that for each input in the language (for which the answer is YES), there is an appropriate "witness" such that the verifier accepts the input if that witness is provided. Soundness means that for each input not in the language (for which the answer is NO) the verifier rejects the input no matter what witness is provided. It is implicit that the witness is of polynomial length, $|y| = \text{poly}(|x|)$; since the verifier has a polynomial number of gates, including input gates, it cannot make use of more than a polynomial number of bits of the witness. NP stands for "nondeterministic polynomial time;" this name is used for historical reasons, but it is a bit confusing since the verifier is actually a deterministic circuit (evaluates a function).

If is obvious that P $\subseteq$ NP; if the problem is in P then the polynomial-time verifier can decide whether to accept x on its own, without any help from the witness. But some problems in NP seem to be hard, and are believed not to be in P. Much of complexity theory is built on a fundamental conjecture:

$$\text{Conjecture : P} \neq \text{NP}. \tag{5.20}$$

Proving or refuting this conjecture is the most important open problem in computer science, and one of the most important problems in mathematics.

Why should we believe P $\neq$ NP? If P = NP that would mean we could easily find the solution to any problem whose solution is easy to check. In effect, then, we could automate creativity; in particular, computers would be able to discover all the mathematical theorems which have short proofs. The conjecture P $\neq$ NP asserts that our machines will never achieve such awesome power — that the mere existence of a succinct proof of a statement does not ensure that we can find the proof by any systematic procedure in any reasonable amount of time.

An important example of a problem in NP is CIRCUIT-SAT. In this case the input is a Boolean circuit C, and the problem is to determine

whether any input x is accepted by C. The function to be evaluated is

$$f(C) = \begin{cases} 1 & \text{if there exists } x \text{ with } C(x) = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{5.21}$$

This problem is in NP because if the circuit C has polynomial size, then if we are provided with an input x accepted by C it is easy to check that $C(x) = 1$.

The problem CIRCUIT-SAT is particularly interesting because it has a remarkable property — if we have a machine that solves CIRCUIT-SAT we can use it to solve any other problem in NP. We say that every problem in NP is (efficiently) *reducible* to CIRCUIT-SAT. More generally, we say that problem B reduces to problem A if a machine that solves A can be used to solve B as well.

That is, if A and B are Boolean function families, then "B reduces to A" means there is a function family R, computed by poly-size circuits, such that $B(x) = A(R(x))$. Thus B accepts x iff A accepts $R(x)$. In particular, then, if we have a poly-size circuit family that solves A, we can hook A up with R to obtain a poly-size circuit family that solves B.

A problem B in NP reduces to CIRCUIT-SAT because problem B has a poly-size verifier $V(x, y)$, such that B accepts x iff there exists some witness y such that V accepts (x, y). For each fixed x, asking whether such a witness y exists is an instance of CIRCUIT-SAT. So a poly-size circuit family that solves CIRCUIT-SAT can also be used to solve problem B.

We say a problem A in NP is *NP-complete* if every problem in NP is reducible to A. Hence, CIRCUIT-SAT is NP-complete. The NP-complete problems are the "hardest" problems in NP, in the sense that if we can solve any NP-complete problem then we can solve every NP problem. Furthermore, to show that problem A is NP-complete, it is enough to show that B reduces to A where B is NP-complete. If C is any problem in NP and B is NP-complete then there is a poly-size reduction R such that $C(x) = B(R(x))$, and if B is reducible to A then there is another poly-size reduction R' such that $B(y) = A(R'(y))$. Hence $C(x) = A(R'(R(x)))$, and since the composition $R' \circ R$ of two poly-size reductions is also poly-size, we see that an arbitrary problem C in NP reduces to A, and therefore A is NP-complete. NP-completeness is a useful concept because hundreds of "natural" computational problems turn out to be NP-complete. For example, one can exhibit a polynomial reduction of CIRCUIT-SAT to HAMILTONIAN PATH, and it follows that HAMILTONIAN PATH is also NP-complete.

Another noteworthy complexity class is called co-NP. While NP problems can be decided by exhibiting an *example* if the answer is YES, co-NP

problems can be answered by exhibiting a *counter-example* if the answer is NO. More formally:

Definition. co-NP. *A language L is in co-NP iff there is a polynomial-size verifier $\bar{V}(x, y)$ such that*

If $x \notin L$, then there exists y such that $\bar{V}(x, y) = 1$,

If $x \in L$, then, for all y, $\bar{V}(x, y) = 0$.

For NP the witness y testifies that x is in the language while for co-NP the witness testifies that x is *not* in the language. Thus if language L is in NP, then its complement $\bar{L}$ is in co-NP and vice-versa. We see that whether we consider a problem to be in NP or in co-NP depends on how we choose to frame the question — while "Is there a Hamiltonian path?" is in NP, the complementary question "Is there no Hamiltonian path?" is in co-NP.

Though the distinction between NP and co-NP may seem arbitrary, it is nevertheless interesting to ask whether a problem is in *both* NP and co-NP. If so, then we can easily verify the answer (once a suitable witness is in hand) regardless of whether the answer is YES or NO. It is believed (though not proved) that NP $\neq$ co-NP. For example, we can show that a graph has a Hamiltonian path by exhibiting an example, but we don't know how to show that it has *no* Hamiltonian path that way!

If we assume that P $\neq$ NP, it is known that there exist problems in NP of intermediate difficulty (the class NPI), which are neither in P nor NP-complete. Furthermore, assuming that that NP $\neq$ co-NP, it is known that no co-NP problems are NP-complete. Therefore, problems in the intersection of NP and co-NP, if not in P, are good candidates for inclusion in NPI.

In fact, a problem in NP $\cap$ co-NP believed not to be in P is the FACTORING problem. As already noted, FACTORING is in NP because, if we are offered a factor of x, we can easily check its validity. But it is also in co-NP, because it is known that if we are given a prime number we can efficiently verify its primality. Thus, if someone tells us the prime factors of x, we can efficiently check that the prime factorization is right, and can *exclude* that any integer less than y is a divisor of x. Therefore, it seems likely that FACTORING is in NPI.

We are led to a crude (conjectured) picture of the structure of NP $\cup$ co-NP. NP and co-NP do not coincide, but they have a nontrivial intersection. P lies in NP $\cap$ co-NP but the intersection also contains problems not in P (like FACTORING). No NP-complete or co-NP-complete problems lie in NP $\cap$ co-NP.

5.1.4 Randomized computation

It is sometimes useful to consider *probabilistic* circuits that have access to a random number generator. For example, a gate in a probabilistic circuit might act in either one of two ways, and flip a fair coin to decide which action to execute. Such a circuit, for a single fixed input, can sample many possible computational paths. An algorithm performed by a probabilistic circuit is said to be "randomized."

If we run a randomized computation many times on the same input, we won't get the same answer every time; rather there is a probability distribution of outputs. But the computation is useful if the probability of getting the right answer is high enough. For a decision problem, we would like a randomized computation to accept an input x which is in the language L with probability at least $\frac{1}{2} + \delta$, and to accept an input x which is not in the language with probability no greater than $\frac{1}{2} - \delta$, where $\delta > 0$ is a constant independent of the input size. In that case we can *amplify* the probability of success by performing the computation many times and taking a majority vote on the outcomes. For $x \in L$, if we run the computation N times, the probability of rejecting in more than half the runs is no more than $e^{-2N\delta^2}$ (the *Chernoff bound*). Likewise, for $x \notin L$, the probability of accepting in the majority of N runs is no more than $e^{-2N\delta^2}$.

Why? There are all together 2^N possible sequences of outcomes in the N trials, and the probability of any particular sequence with N_W wrong answers is

$$\left(\frac{1}{2} - \delta\right)^{N_W} \left(\frac{1}{2} + \delta\right)^{N - N_W} . \tag{5.22}$$

The majority is wrong only if $N_W \geq N/2$, so the probability of any sequence with an incorrect majority is no larger than

$$\left(\frac{1}{2} - \delta\right)^{N/2} \left(\frac{1}{2} + \delta\right)^{N/2} = \frac{1}{2^N} \left(1 - 4\delta^2\right)^{N/2} . \tag{5.23}$$

Using $1 - x \leq e^{-x}$ and multiplying by the total number of sequences 2^N, we obtain the Chernoff bound:

$$\text{Prob(wrong majority)} \leq \left(1 - 4\delta^2\right)^{N/2} \leq e^{-2N\delta^2} . \tag{5.24}$$

If we are willing to accept a probability of error no larger than ε, then, it suffices to run the computation a number of times

$$N \geq \frac{1}{2\delta^2} \ln\left(1/\varepsilon\right) . \tag{5.25}$$

Because we can make the error probability very small by repeating a randomized computation a modest number of times, the value of the constant δ does not really matter for the purpose of classifying complexity, as long as it is positive and independent of the input size. The standard convention is to specify $\delta = 1/6$, so that $x \in L$ is accepted with probability at least $2/3$ and $x \notin L$ is accepted with probability no more than $1/3$. This criterion defines the class BPP ("bounded-error probabilistic polynomial time") containing decision problems solved by polynomial-size randomized uniform circuit families.

It is clear that BPP contains P, since a deterministic computation is a special case of a randomized computation, in which we never consult the source of randomness. It is widely believed that BPP=P, that randomness does not enhance our computational power, but this has not been proven. It is not even known whether BPP is contained in NP.

We may define a randomized class analogous to NP, called MA ("Merlin-Arthur"), containing languages that can be checked when a randomized verifier is provided with a suitable witness:

Definition. MA. *A language L is in MA iff there is a polynomial-size randomized verifier $V(x, y)$ such that*

If $x \in L$, then there exists y such that $\text{Prob}(V(x, y) = 1) \geq 2/3$,

If $x \notin L$, then, for all y, $\text{Prob}(V(x, y) = 1) \leq 1/3$.

The colorful name evokes a scenario in which the all-powerful Merlin uses his magical powers to conjure the witness, allowing the mortal Arthur, limited to polynomial time computation, to check the answer. Obviously BPP is contained in MA, but we expect BPP $\neq$ MA just as we expect P $\neq$ NP.

5.2 Reversible computation

In devising a model of a quantum computer, we will generalize the circuit model of classical computation. But our quantum logic gates will be unitary transformations, and hence will be invertible, while classical logic gates like the AND gate are not invertible. Before we discuss quantum circuits, it is useful to consider some features of *reversible* classical computation.

5.2.1 Landauer's principle

Aside from providing a bridge to quantum computation, classical reversible computing is interesting in its own right, because of *Landauer's principle*. Landauer observed that erasure of information is necessarily a

dissipative process. His insight is that erasure always involves the compression of phase space, and so is thermodynamically, as well as logically, irreversible.

For example, I can store one bit of information by placing a single molecule in a box, either on the left side or the right side of a partition that divides the box. Erasure means that we move the molecule to the right side (say) irrespective of whether it started out on the left or right. I can suddenly remove the partition, and then slowly compress the one-molecule "gas" with a piston until the molecule is definitely on the right side. This procedure changes the entropy of the gas by $\Delta S = -k \ln 2$ (where k is Boltzmann's constant) and there is an associated flow of heat from the box to its environment. If the process is quasi-static and isothermal at temperature T, then work $W = -kT\Delta S = kT \ln 2$ is performed on the box, work that I have to provide. If I erase information, someone has to pay the power bill.

Landauer also observed that, because irreversible logic elements erase information, they too are necessarily dissipative, and therefore require an unavoidable expenditure of energy. For example, an AND gate maps two input bits to one output bit, with 00, 01, and 10 all mapped to 0, while 11 is mapped to one. If the input is destroyed and we can read only the output, then if the output is 0 we don't know for sure what the input was — there are three possibilities. If the input bits are chosen uniformly at random, than on average the AND gate destroys $\frac{3}{4} \log_2 3 \approx 1.189$ bits of information. Indeed, if the input bits are uniformly random any 2-to-1 gate must "erase" at least one bit on average. According to Landauer's principle, then, we need to do an amount of work at least $W = kT \ln 2$ to operate a 2-to-1 logic gate at temperature T.

But if a computer operates reversibly, then in principle there need be no dissipation and no power requirement. We can compute for free! At present this idea is not of great practical importance, because the power consumed in today's integrated circuits exceeds kT per logic gate by at least three orders of magnitude. As the switches on chips continue to get smaller, though, reversible computing might eventually be invoked to reduce dissipation in classical computing hardware.

5.2.2 Reversible gates

A reversible computer evaluates an invertible function taking n bits to n bits

$$f : \{0,1\}^n \to \{0,1\}^n \ . \tag{5.26}$$

An invertible function has a unique input for each output, and we can run the computation backwards to recover the input from the output. We

may regard an invertible function as a permutation of the 2^n strings of n bits — there are $(2^n)!$ such functions. If we did not insist on invertibility, there would be $\left(2^{2^n}\right)^n = (2^n)^{2^n}$ functions taking n bits to n bits (the number of ways to choose n Boolean functions); using the Stirling approximation, $(2^n)! \approx (2^n/e)^{2^n}$, we see that the fraction of all functions which are invertible is quite small, about e^{-2^n}.

Any irreversible computation can be "packaged" as an evaluation of an invertible function. For example, for any $f : \{0,1\}^n \to \{0,1\}$, we can construct $\tilde{f} : \{0,1\}^{n+1} \to \{0,1\}^{n+1}$ such that

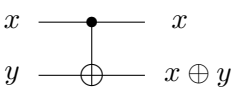
$$\tilde{f}(x, y) = (x, y \oplus f(x)). \tag{5.27}$$

Here y is a bit and $\oplus$ denotes the XOR gate (addition mod 2) — the n-bit input x is preserved and the last bit flips iff $f(x) = 1$. Applying $\tilde{f}$ a second time undoes this bit flip; hence $\tilde{f}$ is invertible, and equal to its own inverse. If we set $y = 0$ initially and apply $\tilde{f}$, we can read out the value of $f(x)$ in the last output bit.

Just as for Boolean functions, we can ask whether a complicated reversible computation can be executed by a circuit built from simple components — are there universal reversible gates? It is easy to see that one-bit and two-bit reversible gates do not suffice; we will need three-bit gates for universal reversible computation.

Of the four 1-bit $\to$ 1-bit gates, two are reversible; the trivial gate and the NOT gate. Of the $(2^4)^2 = 256$ possible 2-bit $\to$ 2-bit gates, $4! = 24$ are reversible. One of special interest is the controlled-NOT (CNOT) or reversible XOR gate that we already encountered in Chapter 4:

$$\text{XOR}: \ (x, y) \mapsto (x, x \oplus y), \tag{5.28}$$

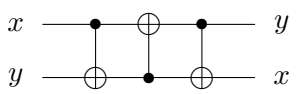


This gate flips the second bit if the first is 1, and does nothing if the first bit is 0 (hence the name controlled-NOT). Its square is trivial; hence it inverts itself. Anticipating the notation that will be convenient for our discussion of quantum gates, we will sometimes use $\Lambda(\boldsymbol{X})$ to denote the CNOT gate. More generally, by $\Lambda(\boldsymbol{G})$ we mean a gate that applies the operation $\boldsymbol{G}$ to a "target" conditioned on the value of a "control bit;" $\boldsymbol{G}$ is applied if the control bit is 1 and the identity is applied if the control bit is 0. In the case of the CNOT gate, $\boldsymbol{G}$ is the Pauli operator $\boldsymbol{X}$, a bit flip.

The CNOT gate performs a NOT on the second bit if the first bit x is set to 1, and it performs the copy operation if y is initially set to zero:

$$\text{CNOT}: \ (x,0) \mapsto (x,x). \tag{5.29}$$

With the circuit



constructed from three XOR's, we can swap two bits:

$$(x,y) \to (x, x\oplus y) \to (y, x\oplus y) \to (y,x). \tag{5.30}$$

With these swaps we can shuffle bits around in a circuit, bringing them together if we want to act on them with a "local gate" at a fixed location.

To see that the one-bit and two-bit gates are nonuniversal, we observe that all these gates are *linear*. Each reversible two-bit gate has an action of the form

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x' \\ y' \end{pmatrix} = M \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix}; \tag{5.31}$$

the pair of bits $\binom{a}{b}$ can take any one of four possible values, and the matrix M is one of the six invertible matrices with binary entries

$$\begin{aligned} M = &\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \\ &\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}. \end{aligned} \tag{5.32}$$

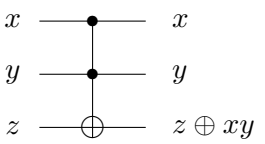
(All addition is performed modulo 2.) Combining the six choices for M with the four possible constants, we obtain 24 distinct gates, exhausting all the reversible $2 \to 2$ gates.

Since the linear transformations are closed under composition, any circuit composed from reversible $2 \to 2$ (and $1 \to 1$) gates will compute a linear function

$$x \mapsto Mx + a. \tag{5.33}$$

But for $n \geq 3$, there are invertible functions on n-bits that are nonlinear. An important example is the 3-bit *Toffoli gate* (or controlled-controlled-NOT) $\Lambda^2(\boldsymbol{X})$

$$\Lambda^2(\boldsymbol{X}) : (x,y,z) \to (x,y,z\oplus xy); \tag{5.34}$$



it flips the third bit if the first two are 1 and does nothing otherwise, thus invoking the (nonlinear) multiplication of the two bits x and y. The $\Lambda^2(\cdot)$ notation indicates that the operation acting on the target bit is triggered only if both control bits are set to 1. Like the CNOT gate $\Lambda(\boldsymbol{X})$, $\Lambda^2(\boldsymbol{X})$ is its own inverse.

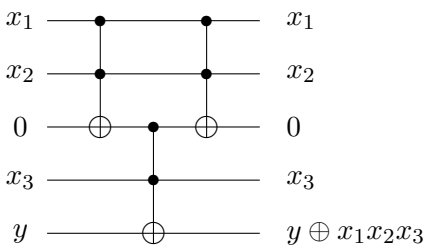
Unlike the reversible 2-bit gates, the Toffoli gate serves as a universal gate for Boolean logic, if we can provide constant input bits and ignore output bits. If we fix $x = y = 1$, then the Toffoli gate performs NOT acting on the third bit, and if z is set to zero initially, then the Toffoli gate outputs $z = x \wedge y$ in the third bit. Since NOT/AND/OR are a universal gate set, and we can construct OR from NOT and AND ($x \vee y = \neg(\neg x \wedge \neg y)$), this is already enough to establish that the Toffoli gate is universal. Note also that if we fix $x = 1$ the Toffoli gate functions like a CNOT gate acting on y and z; we can use it to copy.

The Toffoli gate $\Lambda^2(\boldsymbol{X})$ is also universal in the sense that we can build a circuit to compute any reversible function using Toffoli gates alone (if we can fix input bits and ignore output bits). It will be instructive to show this directly, without relying on our earlier argument that NOT/AND/OR is universal for Boolean functions. Specifically, we can show the following: From the NOT gate and the Toffoli gate $\Lambda^2(\boldsymbol{X})$, we can construct any invertible function on n bits, provided we have one extra bit of scratchpad space available.

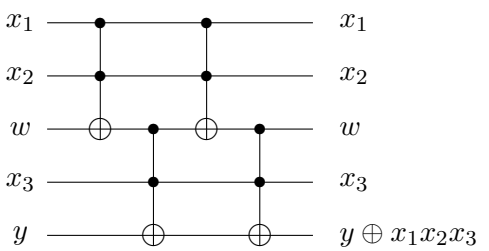
The first step is to show that from the three-bit Toffoli-gate $\Lambda^2(\boldsymbol{X})$ we can construct an n-bit Toffoli gate $\Lambda^{n-1}(\boldsymbol{X})$ that acts as

$$(x_1, x_2, \ldots x_{n-1}, y) \to (x_1, x_2, \ldots, x_{n-1}, y \oplus x_1 x_2 \ldots x_{n-1}) \tag{5.35}$$

using one extra bit of scratch space. For example, we construct $\Lambda^3(\boldsymbol{X})$ from $\Lambda^2(\boldsymbol{X})$'s with the circuit



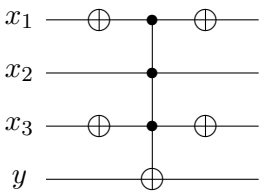
The purpose of the last $\Lambda^3(\boldsymbol{X})$ gate is to reset the scratch bit back to its original value 0. Actually, with one more gate we can obtain an implementation of $\Lambda^3(\boldsymbol{X})$ that works irrespective of the initial value of the scratch bit:



We can see that the scratch bit really is necessary, because $\Lambda^3(\boldsymbol{X})$ is an odd permutation (in fact a transposition) of the 4-bit strings — it transposes 1111 and 1110. But $\Lambda^2(\boldsymbol{X})$ acting on any three of the four bits is an even permutation; *e.g.*, acting on the last three bits it transposes both 0111 with 0110 and 1111 with 1110. Since a product of even permutations is also even, we cannot obtain $\Lambda^3(\boldsymbol{X})$ as a product of $\Lambda^2(\boldsymbol{X})$'s that act only on the four bits.

This construction of $\Lambda^3(\boldsymbol{X})$ from four $\Lambda^2(\boldsymbol{X})$'s generalizes immediately to the construction of $\Lambda^{n-1}(\boldsymbol{X})$ from two $\Lambda^{n-2}(\boldsymbol{X})$'s and two $\Lambda^2(\boldsymbol{X})$'s (just expand x_1 to several control bits in the above diagram). Iterating the construction, we obtain $\Lambda^{n-1}(\boldsymbol{X})$ from a circuit with $2^{n-2} + 2^{n-3} - 2$ $\Lambda^2(\boldsymbol{X})$'s. Furthermore, just one bit of scratch suffices. (With more scratch space, we can build $\Lambda^{n-1}(\boldsymbol{X})$ from $\Lambda^2(\boldsymbol{X})$'s much more efficiently — see Exercise 5.1.)

The next step is to note that, by conjugating $\Lambda^{n-1}(\boldsymbol{X})$ with NOT gates, we can in effect modify the value of the control string that "triggers" the gate. For example, the circuit



flips the value of y if $x_1x_2x_3 = 010$, and it acts trivially otherwise. Thus this circuit transposes the two strings 0100 and 0101. In like fashion, with $\Lambda^{n-1}(\boldsymbol{X})$ and NOT gates, we can devise a circuit that transposes any two n-bit strings that differ in only one bit. (The location of the bit where they differ is chosen to be the *target* of the $\Lambda^{n-1}(\boldsymbol{X})$ gate.)

But in fact a transposition that exchanges any two n-bit strings can be expressed as a product of transpositions that interchange strings that differ in only one bit. If a_0 and a_s are two strings that are Hamming distance s apart (differ in s places), then there is a sequence of strings

$$a_0, a_1, a_2, a_3, \ldots, a_s, \tag{5.36}$$

such that each string in the sequence is Hamming distance one from its neighbors. Therefore, each of the transpositions

$$(a_0a_1), (a_1a_2), (a_2a_3), \ldots, (a_{s-1}a_s), \tag{5.37}$$

can be implemented as a $\Lambda^{n-1}(\boldsymbol{X})$ gate conjugated by NOT gates. By composing transpositions we find

$$\begin{aligned}(a_0a_s) = (a_{s-1}a_s)(a_{s-2}a_{s-1})\ldots(a_2a_3)(a_1a_2)(a_0a_1)(a_1a_2)(a_2a_3)\\ \ldots(a_{s-2}a_{s-1})(a_{s-1}a_s);\end{aligned} \tag{5.38}$$

we can construct the Hamming-distance-s transposition from $2s-1$ Hamming-distance-one transpositions. It follows that we can construct (a_0a_s) from $\Lambda^{n-1}(\boldsymbol{X})$'s and NOT gates.

Finally, since every permutation is a product of transpositions, we have shown that every invertible function on n-bits (every permutation of the n-bit strings) is a product of $\Lambda^{n-1}(\boldsymbol{X})$'s and NOT's, using just one bit of scratch space.

Of course, a NOT can be performed with a $\Lambda^2(\boldsymbol{X})$ gate if we fix two input bits at 1. Thus the Toffoli gate $\Lambda^2(\boldsymbol{X})$ is universal for reversible computation, if we can fix input bits and discard output bits.

5.2.3 Saving space: the pebble game

We have seen that with Toffoli and NOT gates we can compute any invertible function using very little scratch space, and also that by fixing constant input bits and ignoring output bits, we can simulate any (irreversible) Boolean circuit using only reversible Toffoli gates. In the latter case, though, we generate two bits of junk every time we simulate an AND gate. Our memory gradually fills with junk, until we reach the stage where we cannot continue with the computation without erasing some bits to clear some space. At that stage, we will finally have to pay the power bill for the computing we have performed, just as Landauer had warned.

Fortunately, there is a general procedure for simulating an irreversible circuit using reversible gates, in which we can erase the junk without using any power. We accumulate and save all the junk output bits as the simulation proceeds, and when we reach the end of the computation we make a copy of the output. The COPY operation, which is logically reversible, can be done with a CNOT or Toffoi gate. Then we run the full computation in reverse, executing the circuit in the opposite order and replacing each gate by its inverse. This procedure cleans up all the junk bits, and returns all registers to their original settings, without any irreversible erasure steps. Yet the result of the computation has been retained, because we copied it before reversing the circuit.

Because we need to run the computation both forward and backward, the reversible simulation uses roughly twice as many gates as the irreversible circuit it simulates. Far worse than that, this simulation method requires a substantial amount of memory, since we need to be able to store about as many bits as the number of gates in the circuit before we finally start to clear the memory by reversing the computation.

It is possible, though, at the cost of modestly increasing the simulation time, to substantially reduce the space required. The trick is to clear space during the course of the simulation by running a part of the computation backward. The resulting tradeoff between time and space is worth discussing, as it illustrates both the value of "uncomputing" and the concept of a recursive simulation.

We imagine dividing the computation into steps of roughly equal size. When we run step k forward, the first thing we do is make a copy of the output from the previous step, then we execute the gates of step k, retaining all the junk accumulated by those gates. We cannot run step k forward unless we have previously completed step $k-1$. Furthermore, we will not be able to run step k backward if we have already run step $k-1$ backward. The trouble is that we will not be able to reverse the COPY step at the very beginning of step k unless we have retained the output

from step $k-1$.

To save space in our simulation we want to minimize at all times the number of steps that have already been computed but have not yet been uncomputed. The challenge we face can be likened to a game — the *reversible pebble game.* The steps to be executed form a one-dimension directed graph with sites labeled $1, 2, 3, \ldots, T$. Execution of step k is modeled by placing a pebble on the kth site of the graph, and executing step k in reverse is modeled as removal of a pebble from site k. At the beginning of the game, no sites are covered by pebbles, and in each turn we add or remove a pebble. But we cannot place a pebble at site k (except for $k = 1$) unless site $k-1$ is already covered by a pebble, and we cannot remove a pebble from site k (except for $k = 1$) unless site $k-1$ is covered. The object is to cover site T (complete the computation) without using more pebbles than necessary (generating a minimal amount of garbage).

We can construct a recursive procedure that enables us to reach site $t = 2^n$ using $n+1$ pebbles and leaving only one pebble in play. Let $F_1(k)$ denote placing a pebble at site k, and $F_1(k)^{-1}$ denote removing a pebble from site k. Then

$$F_2(1,2) = F_1(1)F_1(2)F_1(1)^{-1}, \tag{5.39}$$

leaves a pebble at site $k = 2$, using a maximum of two pebbles at intermediate stages. Similarly

$$F_3(1,4) = F_2(1,2)F_2(3,4)F_2(1,2)^{-1}, \tag{5.40}$$

reaches site $k = 4$ using three pebbles, and

$$F_4(1,8) = F_3(1,4)F_3(5,8)F_3(1,4)^{-1}, \tag{5.41}$$

reaches $k = 8$ using four pebbles. Proceeding this way we construct $F_n(1, 2^n)$ which uses a maximum of $n+1$ pebbles and leaves a single pebble in play.

Interpreted as a routine for simulating $T_{\rm irr} = 2^n$ steps of an irreversible computation, this strategy for playing the pebble game represents a reversible simulation requiring space $S_{\rm rev}$ scaling like

$$S_{\rm rev} \approx S_{\rm step} \log_2 \left(T_{\rm irr}/T_{\rm step} \right), \tag{5.42}$$

where $T_{\rm step}$ is the number of gates is a single step, and $S_{\rm step}$ is the amount of memory used in a single step. How long does the simulation take? At each level of the recursive procedure described above, two steps forward are replaced by two steps forward and one step back. Therefore, an irreversible computation with $T_{\rm irr}/T_{\rm step} = 2^n$ steps is simulated in $T_{\rm rev}/T_{\rm step} = 3^n$ steps, or

$$T_{\rm rev} = T_{\rm step} \left(T_{\rm irr}/T_{\rm step} \right)^{\log 3/\log 2} = T_{\rm step} (T_{\rm irr}/T_{\rm step})^{1.58}, \tag{5.43}$$

a modest power law slowdown.

We can improve this slowdown to

$$T_{\rm rev} \sim (T_{\rm irr})^{1+\varepsilon}, \tag{5.44}$$

for any $\varepsilon > 0$. Instead of replacing two steps forward with two forward and one back, we replace ℓ forward with ℓ forward and $\ell - 1$ back. A recursive procedure with n levels reaches site ℓ^n using a maximum of $n(\ell-1)+1$ pebbles. Now we have $T_{\rm irr} \propto \ell^n$ and $T_{\rm rev} \propto (2\ell-1)^n$, so that

$$T_{\rm rev} = T_{\rm step}(T_{\rm irr}/T_{\rm step})^{\log(2\ell-1)/\log\ell}; \tag{5.45}$$

the power characterizing the slowdown is

$$\frac{\log(2\ell-1)}{\log\ell} = \frac{\log 2\ell + \log\left(1-\frac{1}{2\ell}\right)}{\log\ell} \simeq 1 + \frac{\log 2}{\log\ell} \equiv 1+\varepsilon, \tag{5.46}$$

and the space requirement scales as

$$S_{\rm rev}/S_{\rm step} \approx \ell n \approx \ 2^{1/\varepsilon} \log_\ell (T_{\rm irr}/T_{\rm step}) \approx \ \varepsilon\ 2^{1/\varepsilon} \log_2 (T_{\rm irr}/T_{\rm step})\,, \tag{5.47}$$

where $1/\varepsilon = \log_2 \ell$. The required space still scales as $S_{\rm rev} \sim \log T_{\rm irr}$, yet the slowdown is no worse than $T_{\rm rev} \sim (T_{\rm irr})^{1+\varepsilon}$. By using more than the minimal number of pebbles, we can reach the last step faster.

You might have worried that, because reversible computation is "harder" than irreversible computation, the classification of complexity depends on whether we compute reversibly or irreversibly. But don't worry — we've now seen that a reversible computer can simulate an irreversible computer pretty easily.

5.3 Quantum Circuits

Now we are ready to formulate a mathematical model of a quantum computer. We will generalize the circuit model of classical computation to the quantum circuit model of quantum computation.

A classical computer processes bits. It is equipped with a finite set of gates that can be applied to sets of bits. A quantum computer processes qubits. We will assume that it too is equipped with a discrete set of fundamental components, called *quantum gates*. Each quantum gate is a unitary transformation that acts on a fixed number of qubits. In a quantum computation, a finite number n of qubits are initially set to the value $|00\ldots0\rangle$. A circuit is executed that is constructed from a finite number of quantum gates acting on these qubits. Finally, an orthogonal measurement of all the qubits (or a subset of the qubits) is performed,

projecting each measured qubit onto the basis $\{|0\rangle, |1\rangle\}$. The outcome of this measurement is the result of the computation.

Several features of this model invite comment:

(1) *Preferred decomposition into subsystems.* It is implicit but important that the Hilbert space of the device has a preferred decomposition into a tensor product of low-dimensional subsystems, in this case the qubits. Of course, we could have considered a tensor product of, say, qutrits instead. But anyway we assume there is a natural decomposition into subsystems that is respected by the quantum gates — the gates act on only a few subsystems at a time. Mathematically, this feature of the gates is crucial for establishing a clearly defined notion of quantum complexity. Physically, the fundamental reason for a natural decomposition into subsystems is *locality*; feasible quantum gates must act in a bounded spatial region, so the computer decomposes into subsystems that interact only with their neighbors.

(2) *Finite instruction set.* Since unitary transformations form a continuum, it may seem unnecessarily restrictive to postulate that the machine can execute only those quantum gates chosen from a discrete set. We nevertheless accept such a restriction, because we do not want to invent a new physical implementation each time we are faced with a new computation to perform. (When we develop the theory of fault-tolerant quantum computing we will see that only a discrete set of quantum gates can be well protected from error, and we'll be glad that we assumed a finite gate set in our formulation of the quantum circuit model.)

(3) *Unitary gates and orthogonal measurements.* We might have allowed our quantum gates to be trace-preserving completely positive maps, and our final measurement to be a POVM. But since we can easily simulate a TPCP map by performing a unitary transformation on an extended system, or a POVM by performing an orthogonal measurement on an extended system, the model as formulated is of sufficient generality.

(4) *Simple preparations.* Choosing the initial state of the n input qubits to be $|00\ldots0\rangle$ is merely a convention. We might want the input to be some nontrivial classical bit string instead, and in that case we would just include NOT gates in the first computational step of the circuit to flip some of the input bits from 0 to 1. What is important, though, is that the initial state is easy to prepare. If we allowed the input state to be a complicated entangled state of the n qubits, then we might be hiding the difficulty of executing the quantum algorithm in the difficulty of preparing the input state. We start with a product state instead, regarding it as uncontroversial that preparation of a product state is easy.

(5) *Simple measurements.* We might allow the final measurement to be a collective measurement, or a projection onto a different basis. But

any such measurement can be implemented by performing a suitable unitary transformation followed by a projection onto the standard basis $\{|0\rangle, |1\rangle\}^n$. Complicated collective measurements can be transformed into measurements in the standard basis only with some difficulty, and it is appropriate to take into account this difficulty when characterizing the complexity of an algorithm.

(6) *Measurements delayed until the end.* We might have allowed measurements at intermediate stages of the computation, with the subsequent choice of quantum gates conditioned on the outcome of those measurements. But in fact the same result can always be achieved by a quantum circuit with all measurements postponed until the end. (While we can postpone the measurements in principle, it might be very useful in practice to perform measurements at intermediate stages of a quantum algorithm.)

A quantum gate, being a unitary transformation, is reversible. In fact, a classical reversible computer is a special case of a quantum computer. A classical reversible gate

$$x \to y = f(x), \tag{5.48}$$

implementing a permutation of k-bit strings, can be regarded as a unitary transformation U acting on k qubits, which maps the "computational basis" of product states

$$\{|x_i\rangle, i = 0, 1, \ldots 2^k - 1\} \tag{5.49}$$

to another basis of product states $\{|y_i\rangle\}$ according to

$$U|x_i\rangle = |y_i\rangle. \tag{5.50}$$

Since U maps one orthonormal basis to another it is manifestly unitary. A quantum computation constructed from such reversible classical gates takes $|0 \ldots 0\rangle$ to one of the computational basis states, so that the outcome of the final measurement in the $\{|0\rangle, |1\rangle\}$ basis is deterministic.

There are four main issues concerning our model that we would like to address in this Chapter. The first issue is *universality.* The most general unitary transformation that can be performed on n qubits is an element of $U(2^n)$. Our model would seem incomplete if there were transformations in $U(2^n)$ that we were unable to reach. In fact, we will see that there are many ways to choose a discrete set of *universal quantum gates.* Using a universal gate set we can construct circuits that compute a unitary transformation coming as close as we please to any element in $U(2^n)$.

Thanks to universality, there is also a machine independent notion of *quantum complexity.* We may define a new complexity class BQP

("bounded-error quantum polynomial time") — the class of languages that can be decided with high probability by polynomial-size uniform quantum circuit families. Since one universal quantum computer can simulate another efficiently, the class does not depend on the details of our hardware (on the universal gate set that we have chosen).

Notice that a quantum computer can easily simulate a probabilistic classical computer: it can prepare $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and then project to $\{|0\rangle, |1\rangle\}$, generating a random bit. Therefore BQP certainly contains the class BPP. But as we discussed in Chapter 1, it seems quite reasonable to expect that BQP is actually larger than BPP, because a probabilistic classical computer cannot easily simulate a quantum computer. The fundamental difficulty is that the Hilbert space of n qubits is huge, of dimension 2^n, and hence the mathematical description of a typical vector in the space is exceedingly complex.

Our second issue is to better characterize the resources needed to simulate a quantum computer on a classical computer. We will see that, despite the vastness of Hilbert space, a classical computer can simulate an n-qubit quantum computer even if limited to an amount of memory space that is polynomial in n. This means the BQP is contained in the complexity class PSPACE, the decision problems that can be solved with polynomial space, but may require exponential time. We also know that NP is contained in PSPACE, because we can determine whether a verifier $V(x, y)$ accepts the input x for any witness y by running the verifier for all possible witnesses. Though there are an exponential number of candidate witnesses to interrogate, each one can be checked in polynomial time and space.

The third important issue we should address is *accuracy*. The class BQP is defined formally under the idealized assumption that quantum gates can be executed with perfect precision. Clearly, it is crucial to relax this assumption in any realistic implementation of quantum computation. A polynomial size quantum circuit family that solves a hard problem would not be of much interest if the quantum gates in the circuit were required to have exponential accuracy. In fact, we will show that this is not the case. An idealized T-gate quantum circuit can be simulated with acceptable accuracy by noisy gates, provided that the error probability per gate scales like $1/T$.

The fourth important issue is *coverage*. We saw that polynomial-size classical circuits can reach only a tiny fraction of all Boolean functions, because there are many more functions than circuits. A similar issue arises for unitary transformations — the unitary group acting on n qubits is vast, and there are not nearly enough polynomial-size quantum circuits to explore it thoroughly. Most quantum states of n qubits can never

be realized in Nature, because they cannot be prepared using reasonable resources.

Despite this limited reach of polynomial-size quantum circuits, quantum computers nevertheless pose a serious challenge to the strong Church–Turing thesis, which contends that any physically reasonable model of computation can be simulated by probabilistic *classical* circuits with at worst a polynomial slowdown. We have good reason to believe that classical computers are unable in general to simulate quantum computers efficiently, in complexity theoretic terms that

$$\mathrm{BPP} \neq \mathrm{BQP}, \tag{5.51}$$

yet this remains an unproven conjecture. Proving BPP $\neq$ BQP is a great challenge, and no proof should be expected soon. Indeed, a corollary would be

$$\mathrm{BPP} \neq \mathrm{PSPACE}, \tag{5.52}$$

which would settle a long-standing and pivotal open question in classical complexity theory.

It might be less unrealistic to hope for a proof that BPP $\neq$ BQP follows from another standard conjecture of complexity theory such as P $\neq$ NP, though no such proof has been found so far. The most persuasive evidence we have suggesting that BPP $\neq$ BQP is that there are some problems which *seem* to be hard for classical circuits yet can be solved efficiently by quantum circuits.

It seems likely, then, that the classification of complexity will be different depending on whether we use a classical computer or a quantum computer to solve problems. If such a separation really holds, it is the quantum classification that should be regarded as the more fundamental, for it is better founded on the physical laws that govern the universe.

5.3.1 Accuracy

Let's discuss the issue of accuracy. We imagine that we wish to implement a computation in which the quantum gates $\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_T$ are applied sequentially to the initial state $|\varphi_0\rangle$. The state prepared by our ideal quantum circuit is

$$|\varphi_T\rangle = \boldsymbol{U}_T \boldsymbol{U}_{T-1} \ldots \boldsymbol{U}_2 \boldsymbol{U}_1 |\varphi_0\rangle. \tag{5.53}$$

But in fact our gates do not have perfect accuracy. When we attempt to apply the unitary transformation $\boldsymbol{U}_t$, we instead apply some "nearby" unitary transformation $\tilde{\boldsymbol{U}}_t$. If we wish to include environmental decoherence in our model of how the actual unitary deviates from the ideal

one, we may regard $\tilde{\boldsymbol{U}}_t$ as a transformation acting jointly on the system and environment, where the ideal unitary is a product $\boldsymbol{U}_t \otimes \boldsymbol{V}_t$, with $\boldsymbol{U}_t$ acting on the computer and $\boldsymbol{V}_t$ acting on the environment.

The errors cause the actual state of the computer to wander away from the ideal state. How far does it wander? After one step, the ideal state would be

$$|\varphi_1\rangle = \boldsymbol{U}_1|\varphi_0\rangle. \tag{5.54}$$

But if the actual transformation $\tilde{\boldsymbol{U}}_1$ where applied instead the state would be

$$\tilde{\boldsymbol{U}}_1|\varphi_0\rangle = |\varphi_1\rangle + |E_1\rangle, \tag{5.55}$$

where

$$|E_1\rangle = (\tilde{\boldsymbol{U}}_1 - \boldsymbol{U}_1)|\varphi_0\rangle \tag{5.56}$$

is an unnormalized vector. (We could also suppose that the initial state deviates from $|\varphi_0\rangle$, which would contribute an additional error to the computation that does not depend on the size of the circuit. We'll ignore that error because we are trying to understand how the error scales with the circuit size.)

Now, if $\tilde{\boldsymbol{U}}_t$ denotes the actual gate applied at step t, $|\tilde{\varphi}_t\rangle$ denotes the actual state after t steps, and $|\varphi_t\rangle$ denotes the ideal state, then we may write

$$\begin{aligned} |\tilde{\varphi}_t\rangle = \tilde{\boldsymbol{U}}_t|\tilde{\varphi}_{t-1}\rangle &= \boldsymbol{U}_t|\varphi_{t-1}\rangle + \left(\tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t\right)|\varphi_{t-1}\rangle + \tilde{\boldsymbol{U}}_t\left(|\tilde{\varphi}_{t-1}\rangle - |\varphi_{t-1}\rangle\right) \\ &= |\varphi_t\rangle + |E_t\rangle + \tilde{\boldsymbol{U}}_t\left(|\tilde{\varphi}_{t-1}\rangle - |\varphi_{t-1}\rangle\right), \end{aligned} \tag{5.57}$$

where $|E_t\rangle = \left(\tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t\right)|\varphi_{t-1}\rangle$. Hence,

$$\begin{aligned} |\tilde{\varphi}_2\rangle &= \tilde{\boldsymbol{U}}_2|\tilde{\varphi}_1\rangle = |\varphi_2\rangle + |E_2\rangle + \tilde{\boldsymbol{U}}_2|E_1\rangle, \\ |\tilde{\varphi}_3\rangle &= \tilde{\boldsymbol{U}}_3|\tilde{\varphi}_2\rangle = |\varphi_3\rangle + |E_3\rangle + \tilde{\boldsymbol{U}}_3|E_2\rangle + \tilde{\boldsymbol{U}}_3\tilde{\boldsymbol{U}}_2|E_1\rangle, \end{aligned} \tag{5.58}$$

and so forth, and after T steps we obtain

$$\begin{aligned} |\tilde{\varphi}_T\rangle = |\varphi_T\rangle + |E_T\rangle + \tilde{\boldsymbol{U}}_T|E_{T-1}\rangle + \tilde{\boldsymbol{U}}_T\tilde{\boldsymbol{U}}_{T-1}|E_{T-2}\rangle \\ + \ldots + \tilde{\boldsymbol{U}}_T\tilde{\boldsymbol{U}}_{T-1}\ldots\tilde{\boldsymbol{U}}_2|E_1\rangle. \end{aligned} \tag{5.59}$$

Thus we have expressed the difference between $|\tilde{\varphi}_T\rangle$ and $|\varphi_T\rangle$ as a sum of T remainder terms. The worst case yielding the largest deviation of $|\tilde{\varphi}_T\rangle$ from $|\varphi_T\rangle$ occurs if all remainder terms line up in the same direction, so that the errors interfere constructively. Therefore, we conclude that

$$\begin{aligned} \| \, |\tilde{\varphi}_T\rangle - |\varphi_T\rangle \, \| \;\; \leq \;\; & \| \, |E_T\rangle \, \| + \| \, |E_{T-1}\rangle \, \| \\ & + \ldots + \| \, |E_2\rangle \, \| + \| \, |E_1\rangle \, \|, \end{aligned} \tag{5.60}$$

where we have used the property $\| \boldsymbol{U}|E_t\rangle \| = \| |E_t\rangle \|$ for any unitary $\boldsymbol{U}$.

Let $\| \boldsymbol{A} \|_{\sup}$ denote the sup norm of the operator $\boldsymbol{A}$ — that is, the largest eigenvalue of $\sqrt{\boldsymbol{A}^\dagger \mathbf{A}}$. We then have

$$\| |E_t\rangle \| = \| \left(\tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t \right) |\varphi_{t-1}\rangle \| \;\; \leq \;\; \| \tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t \|_{\sup} \tag{5.61}$$

(since $|\varphi_{t-1}\rangle$ is normalized). Now suppose that, for each value of t, the error in our quantum gate is bounded by

$$\| \tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t \|_{\sup} \;\; \leq \;\; \varepsilon; \tag{5.62}$$

then after T quantum gates are applied, we have

$$\| |\tilde{\varphi}_T\rangle - |\varphi_T\rangle \| \;\; \leq \;\; T\varepsilon; \tag{5.63}$$

in this sense, the accumulated error in the state grows linearly with the length of the computation.

The distance bounded in eq.(5.62) can equivalently be expressed as $\| \boldsymbol{W}_t - \boldsymbol{I} \|_{\sup}$, where $\boldsymbol{W}_t = \tilde{\boldsymbol{U}}_t \boldsymbol{U}_t^\dagger$. Since $\boldsymbol{W}_t$ is unitary, each of its eigenvalues is a phase $e^{i\theta}$, and the corresponding eigenvalue of $\boldsymbol{W}_t - \boldsymbol{I}$ has modulus

$$|e^{i\theta} - 1| = (2 - 2\cos\theta)^{1/2}, \tag{5.64}$$

so that eq.(5.62) is the requirement that each eigenvalue satisfies

$$\cos\theta > 1 - \varepsilon^2/2, \tag{5.65}$$

(or $|\theta| \lesssim \varepsilon$, for ε small). The origin of eq.(5.63) is clear. In each time step, $|\tilde{\varphi}\rangle$ rotates relative to $|\varphi\rangle$ by (at worst) an angle of order ε, and the distance between the vectors increases by at most of order ε.

How much accuracy is good enough? In the final step of our computation, we perform an orthogonal measurement, and the probability of outcome a, in the ideal case, is

$$p(a) = |\langle a|\varphi_T\rangle|^2. \tag{5.66}$$

Because of the errors, the actual probability is

$$\tilde{p}(a) = |\langle a|\tilde{\varphi}_T\rangle|^2. \tag{5.67}$$

It is shown in Exercise ?? that the L^1 distance between the ideal and actual probability distributions satisfies

$$\|\tilde{p} - p\|_1 = \frac{1}{2} \sum_a |\tilde{p}(a) - p(a)| \;\; \leq \;\; \| |\tilde{\varphi}_T\rangle - |\varphi_T\rangle \| \;\; \leq \;\; T\varepsilon. \tag{5.68}$$

Therefore, if we keep $T\varepsilon$ fixed (and small) as T gets large, the error in the probability distribution also remains fixed (and small).

If we use a quantum computer to solve a decision problem, we want the actual quantum circuit to get the right answer with success probability $\frac{1}{2} + \tilde{\delta}$, where $\tilde{\delta}$ is a positive constant. If the ideal quantum circuit contains T gates and has success probability $\frac{1}{2} + \delta$, where $\delta > 0$, eq.(5.68) shows that $\tilde{\delta}$ is also positive provided $\varepsilon < \delta/T$. We should be able to solve hard problems using quantum computers as long as we can improve the accuracy of the gates linearly with the circuit size. This is still a demanding requirement, since performing very accurate quantum gates is a daunting challenge for the hardware builder. Fortunately, we will be able to show, using the theory of quantum fault tolerance, that *physical* gates with constant accuracy (independent of T) suffice to achieve *logical* gates acting on encoded quantum states with accuracy improving like $1/T$, as is required for truly scalable quantum computing.

5.3.2 BQP $\subseteq$ PSPACE

A randomized classical computer can simulate any quantum circuit if we grant the classical computer enough time and storage space. But how much memory does the classical computer require? Naively, since the simulation of an n-qubit circuit involves manipulating matrices of size 2^n, it may seem that an amount of memory space exponential in n is needed. But we will now show that the classical simulation of a quantum computer can be done to acceptable accuracy (albeit very slowly!) in polynomial space. This means that the quantum complexity class BQP is contained in the class PSPACE of problems that can be solved with polynomial space on a classical computer.

The object of the randomized classical simulation is to sample from a probability distribution that closely approximates the distribution of measurement outcomes for the specified quantum circuit. We will actually exhibit a classical simulation that performs a potentially harder task — estimating the probability $p(a)$ for each possible outcome a of the final measurement, which can be expressed as

$$p(a) = |\langle a|\boldsymbol{U}|0\rangle|^2, \tag{5.69}$$

where

$$\boldsymbol{U} = \boldsymbol{U}_T \boldsymbol{U}_{T-1} \ldots \boldsymbol{U}_2 \boldsymbol{U}_1, \tag{5.70}$$

is a product of T quantum gates. Each $\boldsymbol{U}_t$, acting on the n qubits, can be represented by a $2^n \times 2^n$ unitary matrix, characterized by the complex matrix elements

$$\langle y|\boldsymbol{U}_t|x\rangle, \tag{5.71}$$

where $x, y \in \{0, 1 \ldots, 2^n - 1\}$. Writing out the matrix multiplication explicitly, we have

$$\langle a|\boldsymbol{U}|0\rangle = \sum_{\{x_t\}} \langle a|\boldsymbol{U}_T|x_{T-1}\rangle\langle x_{T-1}|\boldsymbol{U}_{T-1}|x_{T-2}\rangle \ldots \\ \ldots \langle x_2|\boldsymbol{U}_2|x_1\rangle\langle x_1|\boldsymbol{U}_1|0\rangle. \tag{5.72}$$

Eq.(5.72) is a sort of "path integral" representation of the quantum computation – the probability amplitude for the final outcome a is expressed as a coherent sum of amplitudes for each of a vast number $(2^{n(T-1)})$ of possible computational paths that begin at 0 and terminate at a after T steps.

Our classical simulator is to add up the $2^{n(T-1)}$ complex numbers in eq.(5.72) to compute $\langle a|\boldsymbol{U}|0\rangle$. The first problem we face is that finite size classical circuits do integer arithmetic, while the matrix elements $\langle y|\boldsymbol{U}_t|x\rangle$ need not be rational numbers. The classical simulator must therefore settle for an approximate calculation to reasonable accuracy. Each term in the sum is a product of T complex factors, and there are $2^{n(T-1)}$ terms in the sum. The accumulated errors are sure to be small if we express the matrix elements to m bits of accuracy, with m large compared to $nT \log T$. Therefore, we can replace each complex matrix element by pairs of signed integers — the binary expansions, each m bits long, of the real and imaginary parts of the matrix element.

Our simulator will need to compute each term in the sum eq.(5.72) and accumulate a total of all the terms. But each addition requires only a modest amount of scratch space, and furthermore, since only the accumulated subtotal need be stored for the next addition, not much space is needed to sum all the terms, even though there are exponentially many.

So it only remains to consider the evaluation of a typical term in the sum, a product of T matrix elements. We will require a classical circuit that evaluates

$$\langle y|\boldsymbol{U}_t|x\rangle; \tag{5.73}$$

this circuit receives the $2n$-bit input (x, y), and outputs the $2m$-bit value of the (complex) matrix element. Given a circuit that performs this function, it will be easy to build a circuit that multiplies the complex numbers together without using much space.

This task would be difficult if $\boldsymbol{U}_t$ were an arbitrary $2^n \times 2^n$ unitary transformation. But now we may appeal to the properties we have demanded of our quantum gate set — the gates from a discrete set, and each gate acts on a bounded number of qubits. Because there are a fixed finite number of gates, there are only a fixed number of gate subroutines that our simulator needs to be able to call. And because the gate acts on only a few qubits, nearly all of its matrix elements vanish (when n is large),

and the value $\langle y|\boldsymbol{U}_t|x\rangle$ can be determined (to the required accuracy) by a simple circuit requiring little memory.

For example, in the case of a single-qubit gate acting on the first qubit, we have

$$\langle y_{n-1}\dots y_1y_0|\boldsymbol{U}_t|x_{n-1}\dots x_1x_0\rangle = 0 \text{ if } y_{n-1}\dots y_1 \neq x_{n-1}\dots x_1. \quad (5.74)$$

A simple circuit can compare x_1 with y_1, x_2 with y_2, *etc.*, and output zero if the equality is not satisfied. In the event of equality, the circuit outputs one of the four complex numbers

$$\langle y_0|\boldsymbol{U}_t|x_0\rangle, \quad (5.75)$$

to m bits of precision. A simple classical circuit can encode the $8m$ bits of this 2×2 complex-valued matrix. Similarly, a simple circuit, requiring only space polynomial in m, can evaluate the matrix elements of any gate of fixed size.

We see, then, that a classical computer with memory space scaling like $nT\log T$ suffices to simulate a quantum circuit with T gates acting on n qubits. If we wished to consider quantum circuits with superpolynomial size T, we would need a lot of memory, but for a quantum circuit families with size poly(n), a polynomial amount of space is enough. We have shown that BQP $\subseteq$ PSPACE.

But it is also evident that the simulation we have described requires exponential time, because we need to evaluate the sum of $2^{n(T-1)}$ complex numbers (where each term in the sum is a product of T complex numbers). Though most of these terms vanish, there are still an exponentially large number of nonvanishing terms to sum.

5.3.3 Most unitary transformations require large quantum circuits

We saw that any Boolean function can be computed by an exponential-size classical circuit, and also that exponential-size circuits are needed to compute most functions. What are the corresponding statements about unitary transformations and quantum circuits? We will postpone for now consideration of how large a quantum circuit *suffices* to reach any unitary transformation, focusing instead on showing that exponential-size quantum circuits are *required* to reach most unitaries.

The question about quantum circuits is different than the corresponding question about classical circuits because there is a finite set of Boolean functions acting on n input bits, and a continuum of unitary transformations acting on n qubits. Since the quantum circuits are countable (if the quantum computer's gate set is finite), and the unitary transformations

are not, we can't reach arbitrary unitaries with finite-size circuits. We'll be satisfied to accurately *approximate* an arbitrary unitary.

As noted in our discussion of quantum circuit accuracy, to ensure that we have a good approximation in the L^1 norm to the probability distribution for any measurement performed after applying a unitary transformation, it suffices for the actual unitary $\tilde{\boldsymbol{U}}$ to be close to the ideal unitary $\boldsymbol{U}$ in the sup norm. Therefore we will say that $\tilde{\boldsymbol{U}}$ is δ-close to $\boldsymbol{U}$ if $\|\tilde{\boldsymbol{U}} - \boldsymbol{U}\|_{\sup} \leq \delta$. How large should the circuit size T be if we want to approximate any n-qubit unitary to accuracy δ?

If we imagine drawing a ball of radius δ (in the sup norm) centered at each unitary achieved by some circuit with T gates, we want the balls to cover the unitary group $U(N)$, where $N = 2^n$. The number N_{balls} of balls needed satisfies

$$N_{\text{balls}} \geq \frac{\text{Vol}(U(N))}{\text{Vol}(\delta-\text{ball})}, \tag{5.76}$$

where $\text{Vol}(U(N))$ means the total volume of the unitary group and $\text{Vol}(\delta-\text{ball})$ means the volume of a single ball with radius δ. The geometry of $U(N)$ is actually curved, but we may safely disregard that subtlety — all we need to know is that $U(N))$ contains a ball centered at the identity element with a small but constant radius C (independent of N). Ignoring the curvature, because $U(N)$ has real dimension N^2, the volume of this ball (a lower bound on the volume of $U(N)$) is $\Omega_{N^2} C^{N^2}$, where Ω_{N^2} denotes the volume of a unit ball in flat space; likewise, the volume of a δ-ball is $\Omega_{N^2}\delta^{N^2}$. We conclude that

$$N_{\text{balls}} \geq \left(\frac{C}{\delta}\right)^{N^2}. \tag{5.77}$$

On the other hand, if our universal set contains a constant number of quantum gates (independent of n), and each gate acts on no more than k qubits, where k is a constant, then the number of ways to choose the quantum gate at step t of a circuit is no more than constant $\times \binom{n}{k} = \text{poly}(n)$. Therefore the number N_T of quantum circuits with T gates acting on n qubits is

$$N_T \leq (\text{poly}(n))^T. \tag{5.78}$$

We conclude that if we want to reach every element of $U(N)$ to accuracy δ with circuits of size T, hence $N_T \geq N_{\text{balls}}$, we require

$$T \geq 2^{2n} \frac{\log(C/\delta)}{\log(\text{poly}(n))}; \tag{5.79}$$

the circuit size must be exponential. With polynomial-size quantum circuits, we can achieve a good approximation to unitaries that occupy only an exponentially small fraction of the volume of $U(2^n)$!

Reaching any desired quantum state by applying a suitable quantum circuit to a fixed initial (*e.g.*, product) state is easier than reaching any desired unitary, but still hard, because the volume of the 2^n-dimensional n-qubit Hilbert space is exponential in n. Hence, circuits with size exponential in n are required. Future quantum engineers will know the joy of exploring Hilbert space, but no matter how powerful their technology, most quantum states will remain far beyond their grasp. It's humbling.

5.4 Universal quantum gates

We must address one more fundamental question about quantum computation; how do we construct an adequate set of quantum gates? In other words, what constitutes a universal quantum computer?

We will find a pleasing answer. Any generic two-qubit gate suffices for universal quantum computation. That is, for all but a set of measure zero of 4×4 unitary matrices, if we can apply that matrix to any pair of qubits, then we can construct a circuit acting on n qubits which computes a transformation coming as close as we please to any element of $U(2^n)$.

Mathematically, this is not a particularly deep result, but physically it is significant. It means that, in the quantum world, as long as we can devise a generic interaction between any two qubits, and we can implement that interaction accurately, we can build up any quantum computation, no matter how complex. Nontrivial computation is ubiquitous in quantum theory.

Aside from this general result, it is also of some interest to exhibit particular universal gate sets that might be particularly easy to implement physically. We will discuss a few examples.

5.4.1 Notions of universality

In our standard circuit model of quantum computation, we imagine that our circuit has a finite set of "hard-wired" quantum gates

$$\mathcal{G} = \{\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_m\}, \tag{5.80}$$

where $\boldsymbol{U}_j$ acts on k_j qubits, and $k_j \leq k$ (a constant) for each j. Normally we also assume that the gate $\boldsymbol{U}_j$ can be applied to any k_j of the n qubits in the computer. Actually, placing some kind of geometric locality constraints on the gates would not drastically change our analysis of complexity, as long as we can construct (a good approximation to a) a **SWAP** gate (which swaps the positions of two neighboring qubits) using our gate set. If we want to perform $\boldsymbol{U}_j$ on k_j qubits that are widely separated, we may first perform a series of **SWAP** gates to bring the qubits together,

then perform the gate, and finally perform **SWAP** gates to return the qubits to their original positions.

When we say the gate set $\mathcal{G}$ is *universal* we mean that the unitary transformations that can be constructed as quantum circuits using this gate set are *dense* in the unitary group $U(2^n)$, up to an overall phase. That is for any $V \in U(2^n)$ and any $\delta > 0$, there is a unitary $\tilde{V}$ achieved by a finite circuit such that

$$\|\tilde{V} - e^{i\phi} V\|_{\sup} \;\leq\; \delta \tag{5.81}$$

for some phase $e^{i\phi}$. (It is natural to use the sup norm to define the deviation of the circuit from the target unitary, but we would reach similar conclusions using any reasonable topology on $U(2^n)$.) Sometimes it is useful to relax this definition of universality; for example we might settle for *encoded universality*, meaning that the circuits are dense not in $U(2^n)$ but rather some subgroup $U(N)$, where N is exponential (or at least superpolynomial) in n.

There are several variations on the notion of universality that are noteworthy, because they illuminate the general theory or are useful in applications.

(1) *Exact universality.* If we are willing to allow uncountable gate sets, then we can assert that for certain gate sets we can construct a circuit that achieves an arbitrary unitary transformation *exactly*. We will see that two-qubit gates are exactly universal — any element of $U(2^n)$ can be constructed as a finite circuit of two qubit gates. Another example is that the two-qubit CNOT gate, combined with arbitrary single-qubit gates, is exactly universal.

In fact the CNOT gate is not special in this respect. *Any* "entangling" two-qubit gate, when combined with arbitrary single-qubit gates, is universal (Exercise 5.6). We say a two-qubit gate is entangling if it maps some product state to a state which is not a product state.

An example of a two-gate which is not entangling is a "local gate" — a product unitary $\boldsymbol{V} = \boldsymbol{A} \otimes \boldsymbol{B}$; another example is the **SWAP** gate, or any gate "locally equivalent" to **SWAP**, *i.e.*, of the form

$$\boldsymbol{V} = (\boldsymbol{A} \otimes \boldsymbol{B})\,(\mathbf{SWAP})\,(\boldsymbol{C} \otimes \boldsymbol{D})\,. \tag{5.82}$$

In fact these are the only non-entangling two-qubit gates. Every two-qubit unitary which is *not* local or locally equivalent to **SWAP** is entangling, and hence universal when combined with arbitrary single-qubit gates.

(2) *Generic universality.* Gates acting on two or more qubits which which are not local are typically universal. For example, almost any two-qubit gate is universal, if the gate can be applied to any pair of the n qubits. By "almost any" we mean except for a set of measure zero in $U(4)$.

(3) *Particular finite universal gate sets.* It is shown in the Exercises that each one of the following gate sets is universal:

$$\mathcal{G} = \{\boldsymbol{H}, \Lambda(\boldsymbol{S})\}, \quad \{\boldsymbol{H}, \boldsymbol{T}, \Lambda(\boldsymbol{X})\}, \quad \{\boldsymbol{H}, \boldsymbol{S}, \Lambda^2(\boldsymbol{X})\}, \tag{5.83}$$

where $\boldsymbol{H}$, $\boldsymbol{S}$, $\boldsymbol{T}$ are the single-qubit gates

$$\boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \ \boldsymbol{S} = \begin{pmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}, \ \boldsymbol{T} = \begin{pmatrix} e^{-i\pi/8} & 0 \\ 0 & e^{i\pi/8} \end{pmatrix}. \tag{5.84}$$

In Bloch sphere language, the "Hadamard gate" $\boldsymbol{H} = \frac{1}{\sqrt{2}}(\boldsymbol{X} + \boldsymbol{Z})$ is a rotation by π about the axis $\hat{x} + \hat{z}$, $\boldsymbol{S} = \exp\left(-i\frac{\pi}{4}\boldsymbol{Z}\right)$ is a rotation by $\pi/2$ about the $\hat{z}$ axis, and $\boldsymbol{T} = \exp\left(-i\frac{\pi}{8}\boldsymbol{Z}\right)$ is a rotation by $\pi/4$ about the $\hat{z}$ axis. By $\Lambda(\boldsymbol{S})$ we mean the two-qubit in which $\boldsymbol{S}$ is applied to the target qubit iff the control qubit is $|1\rangle$. More generally, we use the notation $\Lambda(\boldsymbol{U})$, where $\boldsymbol{U}$ is a single-qubit gate, to denote the two-qubit gate

$$\Lambda(\boldsymbol{U}) = |0\rangle\langle 0| \otimes \boldsymbol{I} + |1\rangle\langle 1| \otimes \boldsymbol{U}; \tag{5.85}$$

likewise we use $\Lambda^2(\boldsymbol{U})$ to denote the three-qubit gate

$$\Lambda^2(\boldsymbol{U}) = (\boldsymbol{I} - |11\rangle\langle 11|) \otimes \boldsymbol{I} + |11\rangle\langle 11| \otimes \boldsymbol{U}, \tag{5.86}$$

etc.

That particular finite gates sets are universal is especially important in the theory of quantum fault tolerance, in which highly accurate logical gates acting on encoded quantum states are constructed from noisy physical gates. Only a discrete set of logical gates can be well protected against noise, where the set depends on how the quantum information is encoded. The goal of fault-tolerant gate constructions is to achieve a universal set of such protected gates.

(4) *Efficient circuits of universal gates.* The above results concern only the "reachability" of arbitrary n-qubit unitaries; they say nothing about the circuit size needed for a good approximation. Yet the circuit size is highly relevant if we want to approximate one universal gate set by using another one, or if we want to approximate the steps in an ideal quantum algorithm to acceptable accuracy.

We already know that circuits with size exponential in n are needed to approximate arbitrary n-qubit unitaries using a finite gate set. However, we will see that, for any fixed k, a k-qubit unitary can be approximated to accuracy ε using a circuit whose size scales with the error like $\text{polylog}(1/\varepsilon)$. This result, the *Solovay-Kitaev Theorem*, holds for any universal gate set which is "closed under inverse" — that is, such that the inverse of each gate in the set can be constructed exactly using a finite circuit.

The Solovay-Kitaev Theorem tells us that one universal gate set can accurately approximate another one at a modest cost; therefore a characterization of the complexity of a computation based on quantum circuit size is not very sensitive to how the universal gate set is chosen. For example, suppose I build a unitary transformation $\boldsymbol{U}$ using T gates chosen from gate set $\mathcal{G}_1$, and I want to approximate $\boldsymbol{U}$ to constant accuracy ε using gates chosen from gate set $\mathcal{G}_2$. It will suffice to approximate each gate from $\mathcal{G}_1$ to accuracy ε/T, which can be achieved using a circuit of polylog(T/ε) gates from $\mathcal{G}_2$. Therefore $\boldsymbol{U}$ can be approximated with all together $O(T \text{ polylog}(T))$ $\mathcal{G}_2$ gates.

Another consequence of the Solovay-Kitaev Theorem concerns our conclusion that polynomial-size circuits can reach (to constant accuracy) only a tiny fraction of $U(2^n)$. How is the conclusion modified if we build circuits using arbitrary k-qubit unitaries (where k is constant) rather than gates chosen from a finite gate set? Because approximating the k-qubit unitaries using the finite gate set inflates the circuit size by only a polylog(T) factor, if we can achieve an accuracy-δ approximation using a circuit of size T built from arbitrary k-qubit unitaries, then we can also achieve an accuracy-(2δ) approximation using a circuit of size T polylog(T/δ) built from a finite gate set. Thus the criterion eq.(5.79) for reaching all unitaries to accuracy δ using circuits of size T constructed from the finite gate set is replaced by

$$T \text{ polylog}(T/\delta) \geq 2^{2n} \frac{\log(C/2\delta)}{\log n}. \tag{5.87}$$

if we use circuits constructed from arbitrary k-qubit gates. The required circuit size is smaller than exponential by only a poly(n) factor. The group $U(2^n)$ is unimaginably vast not because we are limited to a discrete set of gates, but rather because we are unable to manipulate more than a constant number of qubits at a time.

5.4.2 Two-qubit gates are exactly universal

We will show in two steps that an arbitrary element of $U(2^n)$ can be achieved by a finite circuit of two-qubit gates. First we will show how to express an element of $U(N)$ as a product of "2×2" unitaries; then we we will show how to obtain any 2×2 unitary from a circuit of two-qubit unitaries.

What is a 2×2 unitary? Fix a standard orthonormal basis $\{|0\rangle, |1\rangle, |2\rangle, \ldots |N-1\rangle\}$ for an N-dimensional space. We say a unitary transformation $\boldsymbol{U}$ is 2×2 if it acts nontrivially only in the two-dimensional subspace spanned by two basis elements $|i\rangle$ and $|j\rangle$; that is, $\boldsymbol{U}$ decomposes

as a direct *sum*

$$\boldsymbol{U} = \boldsymbol{U}^{(2)} \oplus \boldsymbol{I}^{(N-2)}, \tag{5.88}$$

where $\boldsymbol{U}^{(2)}$ is a 2×2 unitary matrix acting on the span of $|i\rangle$ and $|j\rangle$, and $\boldsymbol{I}^{(N-2)}$ is the identity matrix acting on the complemenary $(N-2)$-dimensional subspace.

We should be careful not to confuse a 2×2 unitary with a two-qubit unitary acting on the n-qubit space of dimension $N = 2^n$. A two-qubit unitary $\boldsymbol{U}$ decomposes as a tensor *product*

$$\boldsymbol{U} = \boldsymbol{U}^{(4)} \otimes \boldsymbol{I}^{(2^{n-2})}, \tag{5.89}$$

where $\boldsymbol{U}^{(4)}$ is a 4×4 unitary matrix acting on a pair of qubits, and $\boldsymbol{I}^{(2^{n-2})}$ is the identity matrix acting on the remaining $n-2$ qubits. We can regard the two-qubit unitary as a direct sum of 2^{n-2} 4×4 blocks, with each block labeled by a basis state of the $(n-2)$-qubit Hilbert space, and $\boldsymbol{U}^{(4)}$ acting on each block.

Let's see how to express $\boldsymbol{U} \in U(N)$ as a product of 2×2 unitaries. Consider the action of $\boldsymbol{U}$ on the basis state $|0\rangle$:

$$\boldsymbol{U}|0\rangle = \sum_{i=0}^{N-1} a_i |i\rangle. \tag{5.90}$$

We can see that $\boldsymbol{U}|0\rangle$ can be written as $\boldsymbol{W}_0|0\rangle$, where $\boldsymbol{W}_0$ is a product of $(N-1)$ 2×2 unitaries which act as follows:

$$\begin{aligned}
|0\rangle &\mapsto a_0|0\rangle + b_0|1\rangle, \\
b_0|1\rangle &\mapsto a_1|1\rangle + b_1|2\rangle, \\
b_1|2\rangle &\mapsto a_2|2\rangle + b_2|3\rangle, \\
&\cdots \\
b_{N-2}|N-2\rangle &\mapsto a_{N-2}|N-2\rangle + a_{N-1}|N-1\rangle.
\end{aligned} \tag{5.91}$$

Next define $\boldsymbol{U}_1 = \boldsymbol{W}_0^{-1}\boldsymbol{U}$, and note that $\boldsymbol{U}_1|0\rangle = |0\rangle$, so $\boldsymbol{U}_1$ acts nontrivially only in the $(N-1)$-dimensional span of $\{|1\rangle, |2\rangle, \ldots |N-1\rangle\}$. By the same construction as above, we construct $\boldsymbol{W}_1$ as a product of $(N-2)$ 2×2 unitaries such that $\boldsymbol{W}_1|0\rangle = |0\rangle$ and $\boldsymbol{W}_1|1\rangle = \boldsymbol{U}_1|1\rangle$, then define $\boldsymbol{U}_2 = \boldsymbol{W}_1^{-1}\boldsymbol{U}_1$ such that $\boldsymbol{U}_2$ preserves both $|0\rangle$ and $|1\rangle$. Proceeding in this way we construct $\boldsymbol{W}_2$, $\boldsymbol{W}_3$, $\ldots$ $\boldsymbol{W}_{N-2}$ such that

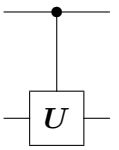
$$\boldsymbol{W}_{N-2}^{-1}\boldsymbol{W}_{N-3}^{-1}\ldots\boldsymbol{W}_1^{-1}\boldsymbol{W}_0^{-1}\boldsymbol{U} = \boldsymbol{I}; \tag{5.92}$$

that is, we may express $\boldsymbol{U}$ as

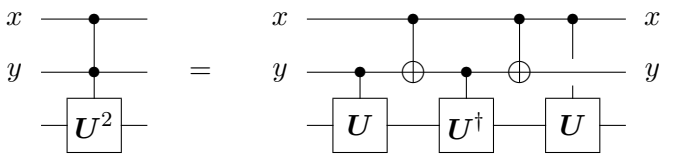
$$\boldsymbol{U} = \boldsymbol{W}_0\boldsymbol{W}_1\ldots\boldsymbol{W}_{N-3}\boldsymbol{W}_{N-2}\ , \tag{5.93}$$

a product of $(N-1)+(N-2)+\cdots+2+1=\frac{1}{2}N(N-1)$ 2×2 unitaries.

Now it remains to show that we can construct any 2×2 unitary as a circuit of two-qubit unitaries. It will be helpful to notice that the three-qubit gate $\Lambda^2(\boldsymbol{U}^2)$ can be constructed as a circuit of $\Lambda(\boldsymbol{U})$, $\Lambda(\boldsymbol{U}^\dagger)$, and $\Lambda(\boldsymbol{X})$ gates. Using the notation



for the $\Lambda(\boldsymbol{U})$ gate, the circuit



does the job. We can check that the power of $\boldsymbol{U}$ applied to the third qubit is

$$y-(x\oplus y)+x=y-(x+y-2xy)+x=2xy. \tag{5.94}$$

That is, $\boldsymbol{U}^2$ is applied if $x=y=1$, and the identity is applied otherwise; hence this circuit achieves the $\Lambda^2(\boldsymbol{U}^2)$ gate. Since every unitary $\boldsymbol{V}$ has a square root $\boldsymbol{U}$ such that $\boldsymbol{V}=\boldsymbol{U}^2$, the construction shows that, using two-qubit gates, we can achieve $\Lambda^2(\boldsymbol{V})$ for any single-qubit $\boldsymbol{V}$.

Generalizing this construction, we can find a circuit that constructs $\Lambda^m(\boldsymbol{U}^2)$ using $\Lambda^{m-1}(\boldsymbol{U})$, $\Lambda^{m-1}(\boldsymbol{X})$, $\Lambda(\boldsymbol{U})$, and $\Lambda(\boldsymbol{U}^\dagger)$ gates. If we replace the $\Lambda(\boldsymbol{X})$ gates in the previous circuit by $\Lambda^{m-1}(\boldsymbol{X})$ gates, and replace the last $\Lambda(\boldsymbol{U})$ gate by $\Lambda^{n-1}(\boldsymbol{U})$, then, if we denote the m control bits by $x_1, x_2, x_3, \ldots x_m$, the power of $\boldsymbol{U}$ applied to the last qubit is

$$\begin{aligned}
&x_m + x_1x_2x_3\ldots x_{m-1} - (x_m \oplus x_1x_2x_3\ldots x_{m-1})\\
&= x_m + x_1x_2x_3\ldots x_{m-1}\\
&- (x_m + x_1x_2x_3\ldots x_{m-1} - 2x_1x_2x_3\ldots x_{m-1}x_m)\\
&= 2x_1x_2x_3\ldots x_{m-1}x_m,
\end{aligned} \tag{5.95}$$

where we have used the identity $x\oplus y = x+y-2xy$. Now $\boldsymbol{U}^2$ is applied if $x_1=x_2=\cdots=x_m=1$ and the identity is applied otherwise; this circuit achieves the $\Lambda^m(\boldsymbol{U}^2)$ gate.

Using the construction recursively, we see that with two-qubit gates we can construct $\Lambda^2(\boldsymbol{V})$ for any $\boldsymbol{V}$, then with these gates and two-qubit gates we can construct $\Lambda^3(\boldsymbol{V})$ for any $\boldsymbol{V}$, which allows us to construct $\Lambda^4(\boldsymbol{V})$ for any $\boldsymbol{V}$ and so on. We have shown, therefore, how to construct the n-qubit gate $\Lambda^{n-1}(\boldsymbol{V})$ for any $\boldsymbol{V}$ using a circuit of two-qubit gates.

To complete the argument showing that any element of $U(2^n)$ is a product of two-qubit unitaries, it will suffice to show that arbitrary 2×2 unitaries can be constructed from $\Lambda^{n-1}(\boldsymbol{V})$ and two-qubit gates. Note that $\Lambda^{n-1}(\boldsymbol{V})$ is, in fact, a 2×2 unitary — it applies $\boldsymbol{V}$ in the two-dimensional space spanned by the two computational basis states

$$\{|111\ldots110\rangle\ , \quad |111\ldots111\rangle\}\ . \tag{5.96}$$

If we wish to apply $\boldsymbol{V}$ in the space spanned by computational states $\{|x\rangle, |y\rangle\}$ instead, we can use a permutation $\boldsymbol{\Sigma}$ of the computational basis states with the action

$$\begin{aligned} \boldsymbol{\Sigma}: \quad & |x\rangle \mapsto |111\ldots110\rangle\ , \\ & |y\rangle \mapsto |111\ldots111\rangle\ , \end{aligned} \tag{5.97}$$

constructing

$$\boldsymbol{\Sigma}^{-1} \circ \Lambda^{n-1}(\boldsymbol{V}) \circ \boldsymbol{\Sigma}\ . \tag{5.98}$$

This is to be read from right to left, with $\boldsymbol{\Sigma}$ acting first and $\boldsymbol{\Sigma^{-1}}$ acting last. But we have already seen in §5.2.2 how to construct an arbitrary permutation of computational basis states using $\Lambda^{n-1}(\boldsymbol{X})$ gates and (single-qubit) NOT gates, and we now know how to construct $\Lambda^{n-1}(\boldsymbol{X})$ (a special case of $\Lambda^{n-1}(\boldsymbol{U})$) from two-qubit gates. Therefore, using two-qubit gates, we have constructed the general 2×2 unitary (in the computational basis) as in eq.(5.98). That completes the proof that any element of $U(2^n)$ can be achieved by a circuit of two-qubit gates.

5.4.3 Finite universal gate sets

Denseness on the circle. A finite gate set is universal if circuits constructed using that gate set are *dense* in $U(2^n)$. As a first simple example of denseness, consider the group $U(1)$ of rotations of the circle, *e.g.* the rotations of the Bloch sphere about the $\hat{z}$ axis:

$$\left\{\boldsymbol{U}(\theta) = \exp\left(i\frac{\theta}{2}\boldsymbol{\sigma}_3\right), \quad \theta \in [0, 4\pi)\right\}\ . \tag{5.99}$$

We claim that the positive integer powers of $\boldsymbol{U}(4\pi\alpha)$ are dense in $U(1)$ if $\alpha \in [0, 1)$ is irrational. Equivalently, the points

$$\{n\alpha \pmod 1, \quad n = 1, 2, 3, \ldots,\} \tag{5.100}$$

are dense in the unit interval.

To see why, first note that the points $\{n\alpha \pmod 1\}$ are all distinct, since $n\alpha = m\alpha + k$ for integers k and $n \neq m$ would imply that α is a rational number $\alpha = k/(n-m)$. Now consider open intervals of width ε centered on each of the N points $\{n\alpha \pmod 1,\ n = 1, 2, 3, \ldots, N\}$. For $N\varepsilon > 1$, at least two of these intervals must intersect — if all intervals were disjoint then their total length $N\varepsilon$ would exceed the length of the interval. Hence there exist distinct positive integers n and m less than $1/\varepsilon$ such that $|n-m|\alpha \pmod 1 < \varepsilon$; in other words, the positive integer $r = |n-m| < 1/\varepsilon$ satisfies $r\alpha \pmod 1 < \varepsilon$. Now the positive integer multiples of $r\alpha \pmod 1$ are equally spaced points on the unit interval separated by less than ε. Therefore, for sufficiently large M, the intervals of width ε centered on the points $\{kr\alpha \pmod 1,\ k = 1, 2, 3, \ldots, M\}$ fill the unit interval. Since ε can be any positive real number, we conclude that the points $\{n\alpha \pmod 1,\ n = 1, 2, 3, \ldots\}$ are dense in the interval.

Powers of a generic gate. Generalizing this argument, consider the positive integer powers of a generic element of $U(N)$. In a suitable basis, $\boldsymbol{U} \in U(N)$ is diagonal, with eigenvalues

$$\{e^{i\theta_1/2}, e^{i\theta_2/2}, \ldots, e^{i\theta_N/2}\}. \tag{5.101}$$

Since rational numbers are countable and real numbers are not, for a generic $\boldsymbol{U}$ (that is, for all elements of $U(N)$ except for a set of measure zero) each θ_i/π and θ_i/θ_j is an irrational number. For each positive integer k, the eigenvalues $\{e^{-ik\theta_i/2}, i = 1, 2, \ldots, N\}$ of $\boldsymbol{U}^k$ define a point on the N-dimensional torus (the product of N circles), and as k ranges over all positive integers, these points densely fill the whole N-torus. We conclude that for any generic $\boldsymbol{U}$, the elements $\{\boldsymbol{U}^k, k = 1, 2, 3, \ldots\}$ are dense in the group $U(1)^N$, *i.e.*, come as close as we please to every unitary matrix which is diagonal in the same basis as $\boldsymbol{U}$.

Note that this argument does not provide any upper bound on how large k must be for $\boldsymbol{U}^k$ to be ε-close to any specified element of $U(1)^N$. In fact, the required value of k could be extremely large if, for some m and i, $|m\theta_i \pmod{4\pi}| << \varepsilon$. It might be hard (that is, require many gates) to approximate a specified unitary transformation with circuits of commuting quantum gates, because the unitary achieved by the circuit only depends on how many times each gate is applied, not on the order in which the gates are applied. It is much easier (requires fewer gates) to achieve a good approximation using circuits of noncommuting gates. If the gates are noncommuting, then the order in which the gates are applied matters, and many more unitaries can be reached by circuits of specified size than if the gates are noncommuting.

Reaching the full Lie algebra. Suppose we can construct the two gates $\boldsymbol{U} = \exp(i\boldsymbol{A}), \boldsymbol{V} = \exp(i\boldsymbol{B}) \in U(N)$, where $\boldsymbol{A}$ and $\boldsymbol{B}$ are $N \times N$

Hermitian matrices. If these are generic gates, positive powers of $\boldsymbol{U}$ come as close as we please to $e^{i\alpha \boldsymbol{A}}$ for any real α and positive powers of $\boldsymbol{V}$ come as close as we please to $e^{i\beta \boldsymbol{B}}$ for any real β. That is enough to ensure that there is a finite circuit constructed from $\boldsymbol{U}$ and $\boldsymbol{V}$ gates that comes as close as we please to $e^{i\boldsymbol{C}}$, where $\boldsymbol{C}$ is any Hermitian element of the Lie algebra generated by $\boldsymbol{A}$ and $\boldsymbol{B}$.

We say that a unitary transformation $\boldsymbol{U}$ is *reachable* if for any $\varepsilon > 0$ there is a finite circuit achieving $\tilde{\boldsymbol{U}}$ which is ε-close to $\boldsymbol{U}$ in the sup norm. Noting that

$$\begin{aligned}\lim_{n\to\infty} (e^{i\alpha \boldsymbol{A}/n} e^{i\beta \boldsymbol{B}/n})^n &= \lim_{n\to\infty} \left(1 + \frac{i}{n}(\alpha \boldsymbol{A} + \beta \boldsymbol{B}) + O\left(\frac{1}{n^2}\right)\right)^n \\ &= e^{i(\alpha \boldsymbol{A} + \beta \boldsymbol{B})}, \end{aligned} \tag{5.102}$$

we see that any $e^{i(\alpha \boldsymbol{A} + \beta \boldsymbol{B})}$ is reachable if each $e^{i\alpha \mathbf{A}/\mathbf{n}}$ and $e^{i\beta \boldsymbol{B}/n}$ is reachable. Furthermore, because

$$\begin{aligned}&\lim_{n\to\infty} \left(e^{i\boldsymbol{A}/\sqrt{n}} e^{i\boldsymbol{B}/\sqrt{n}} e^{-i\boldsymbol{A}/\sqrt{n}} e^{-i\boldsymbol{B}/\sqrt{n}}\right)^n \\ &= \lim_{n\to\infty} \left(1 - \frac{1}{n}(\boldsymbol{A}\boldsymbol{B} - \boldsymbol{B}\boldsymbol{A}) + O\left(\frac{1}{n^{3/2}}\right)\right)^n = e^{-[\boldsymbol{A},\boldsymbol{B}]}, \end{aligned} \tag{5.103}$$

we see that $e^{-[\boldsymbol{A},\boldsymbol{B}]}$ is also reachable.

For example, positive integer powers of a generic element of $SU(2)$ allow us to reach a $U(1)$ subgroup; if we orient our axes on the Bloch sphere appropriately, this is the subgroup generated by the Pauli operator $\boldsymbol{Z}$. Positive integer powers of a second generic element of $SU(2)$ allow us to reach a different $U(1)$ subgroup, generated by $\tilde{\boldsymbol{X}} = \boldsymbol{X} + \gamma \boldsymbol{Z}$ (for some real γ) with an appropriate choice of axes. Because $[\boldsymbol{Z}, \boldsymbol{X}] = i\boldsymbol{Y}$, the elements $\{\boldsymbol{Z}, \tilde{\boldsymbol{X}}, -i[\boldsymbol{Z}, \tilde{\boldsymbol{X}}]\}$ span the three-dimensional $SU(2)$ Lie algebra. It follows that circuits built from any two generic elements of $SU(2)$ suffice to reach any element of $SU(2)$.

This observation applies to higher-dimensional Lie algebras as well. For example, the $SU(4)$ Lie algebra is 15 dimensional. It contains various lower-dimensional subalgebras, such as the Lie algebras of the $SU(4)$ subgroups $U(1)^3$, $SU(2) \times SU(2) \times U(1)$, $SU(3) \times U(1)$, *etc.* But two generic elements of the $SU(4)$ Lie algebra already suffice to general the full Lie algebra. The generated algebra closes on one of the lower-dimensional subalgebras only if some nested commutators vanish "by accident," a criterion satisfied by only a set of measure zero among all pairs of $SU(4)$ generators. Actually, as we have already noted, a generic element of $SU(4)$ allows us to reach the torus $U(1)^3$, and no nontrivial subgroup of $SU(4)$ contains two generic $U(1)^3$ subgroups. Therefore, circuits built from two generic two-qubit gates suffice to reach any two-qubit gate (up

to an overall phase). And since we can reach any element of $U(2^n)$ with two-qubit gates, a pair of generic two-qubit gates provides a universal gate set, assuming we can apply the gates to any pair of qubits.

But in fact just one generic two-qubit gate is already enough, if we are free to choose not just the pair of qubits on which the gate acts but also the ordering of the qubits. That is, a generic two-qubit gate does not commute with the the operator **SWAP** which interchanges the two qubits. If $\boldsymbol{U}$ is a generic two-qubit gate, then

$$\boldsymbol{V} = \mathbf{SWAP} \circ \boldsymbol{U} \circ \mathbf{SWAP} \tag{5.104}$$

(the same gate applied to the same two qubits, but in the opposite order) is another two-qubit gate not commuting with $\boldsymbol{U}$. Positive powers of $\boldsymbol{U}$ reach one $U(1)^3$ subgroup of $SU(4)$ while positive powers of $\boldsymbol{V}$ reach a different $U(1)^3$, so that circuits built from $\boldsymbol{U}$ and $\boldsymbol{V}$ reach all of $SU(4)$.

Even nongeneric universal gates, in particular gates whose eigenvalues are all rational multiples of π, can suffice for universality. One example discussed in the homework is the gate set $\{\text{CNOT}, \boldsymbol{H}, \boldsymbol{T}\}$, where $\boldsymbol{H}$ rotates the Bloch sphere by the angle π about the axis $\frac{1}{\sqrt{2}}(\hat{x} + \hat{x})$, and $\boldsymbol{T}$ rotates the Bloch sphere by the angle $\pi/4$ about the $\hat{z}$ axis. If we replaced $\boldsymbol{T}$ with the $\pi/2$ rotation $\boldsymbol{T}^2$, then the gate set would not be universal; in that case the only achievable single-qubit rotations would be those in a finite subgroup of $SU(2)$, the symmetry group of the cube. But $SU(2)$ has few such finite nonabelian subgroups (the only finite nonabelian subgroups of the rotation group $SO(3)$ are the symmetry groups of regular polygons and of regular three-dimensional polyhedra, the platonic solids). If the gate set reaches beyond these finite subgroups it will reach either a $U(1)$ subgroup of $SU(2)$ or all of $SU(2)$.

5.4.4 The Solovay-Kitaev approximation

Up until now our discussion of universal gates has focused on reachability and has ignored complexity. But when we have a finite universal gate set, we want to know not only *whether* we can approximate a desired unitary transformation to accuracy ε, but also *how hard* it is to achieve that approximation. How large a circuit suffices? The question really has two parts. (1) Given a unitary transformation $\boldsymbol{U}$, how large a quantum circuit is needed to construct $\tilde{\boldsymbol{U}}$ such that $\|\tilde{\boldsymbol{U}} - e^{i\phi}\boldsymbol{U}\|_{\text{sup}} \leq \varepsilon$? (2) How large a *classical* circuit is needed to find the quantum circuit that achieves $\tilde{\boldsymbol{U}}$? We will see that, for any universal set of gates (closed under inverse) used to approximate elements of a unitary group of constant dimension, the answer to both questions is $\text{polylog}(1/\varepsilon)$. We care about the answer to the second question because it would not be very useful to know that

$\boldsymbol{U}$ can be well approximated by small quantum circuits if these circuits are very hard to find.

We will prove this result by devising a recursive algorithm which achieves successively better and better approximations. We say that a finite repertoire of unitary transformations $\mathcal{R}$ is an "ε-net" in $U(N)$ if every element of $U(N)$ is no more than distance ε away (in the sup norm) from some element of $\mathcal{R}$, and we say that $\mathcal{R}$ is "closed under inverse" if the inverse of every element of $\mathcal{R}$ is also in $\mathcal{R}$. The key step of the recursive algorithm is to show that if $\mathcal{R}$ is an ε-net, closed under inverse, then we can construct a new repertoire $\mathcal{R}'$, also closed under inverse, with the following properties: (1) each element of $\mathcal{R}'$ is achieved by a circuit of at most 5 gates from $\mathcal{R}$. (2) $\mathcal{R}'$ is an ε'-net, where

$$\varepsilon' = C\varepsilon^{3/2} \ , \tag{5.105}$$

and C is a constant.

Before explaining how this step works, let's see why it ensures that we can approximate any unitary using a quantum circuit with size polylogarithmic in the accuracy. Suppose to start with that we have found an ε_0-net $\mathcal{R}_0$, closed under inverse, where each element of $\mathcal{R}_0$ can be achieved by a circuit with no more than L_0 gates chosen from our universal gate set. If $\varepsilon_0 < 1/C^2$, then we can invoke the recursive step to find an ε_1-net $\mathcal{R}_1$, where $\varepsilon_1 < \varepsilon_0$, and each element of $\mathcal{R}_1$ can be achieved by a circuit of $L_1 = 5L_0$ gates. By repeating this step k times, we can make the error ε_k much smaller than the level-0 error ε_0. Iterating the relation

$$C^2\varepsilon_k = \left(C^2\varepsilon_{k-1}\right)^{3/2} \tag{5.106}$$

k times we obtain

$$C^2\varepsilon_k = \left(C^2\varepsilon_0\right)^{(3/2)^k} \ , \tag{5.107}$$

and by taking logs of both sides we find

$$\left(\frac{3}{2}\right)^k = \frac{\log\left(1/C^2\varepsilon_k\right)}{\log\left(1/C^2\varepsilon_0\right)} \ . \tag{5.108}$$

After k recursive steps the circuit size for each unitary in the ε_k-net $\mathcal{R}_k$ is no larger than L_k where

$$L_k/L_0 = 5^k = \left(\left(\frac{3}{2}\right)^k\right)^{\log 5/\log(3/2)} = \left(\frac{\log\left(1/C^2\varepsilon_k\right)}{\log\left(1/C^2\varepsilon_0\right)}\right)^{\log 5/\log(3/2)} \ . \tag{5.109}$$

Thus the circuit size scales with the accuracy ε_k as $[\log(1/\varepsilon_k)]^{3.97}$.

Now let's see how the ε'-net $\mathcal{R}'$ is constructed from the ε-net $\mathcal{R}$. For any $\boldsymbol{U} \in SU(N)$ there is an element $\tilde{\boldsymbol{U}} \in \mathcal{R}$ such that $\|\boldsymbol{U} - \tilde{\boldsymbol{U}}\|_{\sup} \leq \varepsilon$, or equivalently $\|\boldsymbol{U}\tilde{\boldsymbol{U}}^{-1} - \boldsymbol{I}\|_{\sup} \leq \varepsilon$. Now we will find $\boldsymbol{W}$, constructed as a circuit of 4 elements of $\mathcal{R}$, such that $\|\boldsymbol{U}\tilde{\boldsymbol{U}}^{-1} - \boldsymbol{W}|_{\sup} \leq \varepsilon'$, or equivalently $\|\boldsymbol{U} - \boldsymbol{W}\tilde{\boldsymbol{U}}|_{\sup} \leq \varepsilon'$. Thus $\boldsymbol{U}$ is approximated to accuracy ε' by $\boldsymbol{W}\tilde{\boldsymbol{U}}$, which is achieved by a circuit of 5 elements of $\mathcal{R}$.

We may write $\boldsymbol{U}\tilde{\boldsymbol{U}}^{-1} = e^{i\boldsymbol{A}}$, where $\boldsymbol{A} = O(\varepsilon)$. (By $\boldsymbol{A} = O(\varepsilon)$ we mean $\|\boldsymbol{A}\|_{\sup} = O(\varepsilon)$, *i.e.*, $\|\boldsymbol{A}\|_{\sup}$ is bounded above by a constant times ε for ε sufficiently small.) It is possible to find Hermitian $\boldsymbol{B}$, $\boldsymbol{C}$, both $O(\varepsilon^{1/2})$, such that $[\boldsymbol{B}, \boldsymbol{C}] = -i\boldsymbol{A}$. Furthermore, because $\mathcal{R}$ is an ε-net, there is an element $e^{i\tilde{\boldsymbol{B}}}$ of $\mathcal{R}$ which is ε-close to $e^{i\boldsymbol{B}}$, and an element $e^{i\tilde{\boldsymbol{C}}}$ of $\mathcal{R}$ which is ε-close to $e^{i\boldsymbol{C}}$. It follows that $\boldsymbol{B} - \tilde{\boldsymbol{B}} = O(\varepsilon)$ and $\boldsymbol{C} - \tilde{\boldsymbol{C}} = O(\varepsilon)$.

Now we consider the circuit

$$\boldsymbol{W} = e^{i\tilde{\boldsymbol{B}}} e^{i\tilde{\boldsymbol{C}}} e^{-i\tilde{\boldsymbol{B}}} e^{-i\tilde{\boldsymbol{C}}} = \boldsymbol{I} - [\tilde{\boldsymbol{B}}, \tilde{\boldsymbol{C}}] + O(\varepsilon^{3/2}); \tag{5.110}$$

the remainder term is cubic order in $\tilde{\boldsymbol{B}}$ and $\tilde{\boldsymbol{C}}$, hence $O(\varepsilon^{3/2})$. First note that the inverse of this circuit, $e^{i\tilde{\boldsymbol{C}}} e^{i\tilde{\boldsymbol{B}}} e^{-i\tilde{\boldsymbol{C}}} e^{-i\tilde{\boldsymbol{B}}}$, can also be constructed as a size-4 circuit of gates from $\mathcal{R}$. Furthermore,

$$\begin{aligned} \boldsymbol{W} &= I - [\boldsymbol{B} + O(\varepsilon), \boldsymbol{C} + O(\varepsilon)] + O(\varepsilon^{3/2}) = I + i\boldsymbol{A} + O(\varepsilon^{3/2}) \\ &= e^{i\boldsymbol{A}} + O(\varepsilon^{3/2}); \end{aligned} \tag{5.111}$$

thus $\boldsymbol{W}$, a circuit of 4 gates from $\mathcal{R}$, approximates $\boldsymbol{U}\tilde{\boldsymbol{U}}^{-1}$ to $O(\varepsilon^{3/2})$ accuracy, as we wanted to show.

Finally, let's consider the classical computational cost of *finding* the quantum circuit which approximates a unitary transformation. The classical algorithm receives a unitary transformation $\boldsymbol{U}$ as input, and produces as output a quantum circuit evaluating $\tilde{\boldsymbol{U}}$, which approximates $\boldsymbol{U}$ to accuracy ε. To improve the accuracy of the approximation to ε', we need to call the accuracy-ε algorithm three times, to find circuits evaluating $\tilde{\boldsymbol{U}}$, $e^{i\tilde{\boldsymbol{B}}}$, and $e^{i\tilde{\boldsymbol{C}}}$. Therefore, if the classical cost of the accuracy-ε algorithm is t, the the classical cost of the improved accuracy-ε' algorithm is $t' = 3t + \text{constant}$, where the additive constant is needed to cover the cost of tasks that do not scale with ε, such as finding the matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ satisfying $[\boldsymbol{B}, \boldsymbol{C}] = -i\boldsymbol{A}$. After k iterations, the classical cost scales like

$$O(3^k) = O\left([\log(1/\varepsilon_k)]^{\log 3/\log(3/2)}\right) = O\left([\log(1/\varepsilon_k)]^{2.71}\right), \tag{5.112}$$

polylogarithmic in the accuracy achieved by the level-k version of the algorithm.

What we have accomplished is a bit surprising. By composing unitary transformations with $O(\varepsilon)$ errors we have obtained unitary transformations with smaller $O(\varepsilon^{3/2})$ errors. How could we achieve such sharp results with such blunt tools? The secret is that we have constructed our circuit so that the $O(\varepsilon)$ errors cancel, leaving only the higher-order errors. This would not have worked if $\mathcal{R}$ had not been closed under inverse. If instead of the inverses of $e^{i\tilde{\boldsymbol{B}}}$ and $e^{i\tilde{\boldsymbol{C}}}$ we had been forced to use $O(\varepsilon)$ approximations to these inverses, the cancellations would not have occured, and our quest for an improved approximation would have failed. But if our universal gate set allows us to construct the exact inverse of each element of the gate set, then we can use the Solovay-Kitaev approach to recursively improve the approximation.

This scheme works for any universal gate set that is closed under inverse. For particular gate sets improved approximations are possible. For example, for the case of the universal gate set $\{\mathrm{CNOT}, \boldsymbol{H}, \boldsymbol{T}\}$, an approximation with accuracy ε can be achieved with $O(\log(1/\varepsilon))$ gates, a substantial improvement over $O([\log(1/\varepsilon)]^{3.97})$ established by the general argument, and the circuits achieving this improved overhead cost can be efficiently constructed.

5.5 Summary

Classical circuits. The complexity of a problem can be characterized by the size of a uniform family of logic circuits that solve the problem: The problem is hard if the size of the circuit is a superpolynomial function of the size of the input, and easy otherwise. One classical universal computer can simulate another efficiently, so the classification of complexity is machine independent. The 3-bit Toffoli gate is universal for classical reversible computation. A reversible computer can simulate an irreversible computer without a significant slowdown and without unreasonable memory resources.

Quantum Circuits. Although there is no proof, it seems likely that polynomial-size quantum circuits cannot be simulated in general by polynomial-size randomized classical circuits (BQP $\neq$ BPP); however, polynomial space is sufficient (BQP $\subseteq$ PSPACE). A noisy quantum circuit can simulate an ideal quantum circuit of size T to acceptable accuracy if each quantum gate has an accuracy of order $1/T$. Any n-qubit unitary transformation can be constructed from two-qubit gates. A generic two-qubit quantum gate, if it can act on any two qubits in a device, is sufficient for universal quantum computation. One universal quantum computer can simulate another to accuracy ε with a polylog$(1/\varepsilon)$ overheard cost; therefore the complexity class BQP is machine independent.

5.6 Exercises

5.1 Linear simulation of Toffoli gate.

In class we constructed the n-bit Toffoli gate $\Lambda^{n-1}(\boldsymbol{X})$ from 3-bit Toffoli gates ($\Lambda^2(\boldsymbol{X})$'s). The circuit required only one bit of scratch space, but the number of gates was exponential in n. With more scratch, we can substantially reduce the number of gates.

a) Find a circuit family with $2n - 5$ $\Lambda^2(\boldsymbol{X})$'s that evaluates $\Lambda^{n-1}(\boldsymbol{X})$. (Here $n - 3$ scratch bits are used, which are set to 0 at the beginning of the computation and return to the value 0 at the end.)

b) Find a circuit family with $4n - 12$ $\Lambda^2(\boldsymbol{X})$'s that evaluates $\Lambda^{n-1}(\boldsymbol{X})$, which works irrespective of the initial values of the scratch bits. (Again the $n-3$ scratch bits return to their initial values, but they don't need to be set to zero at the beginning.)

5.2 A universal quantum gate set.

The purpose of this exercise is to complete the demonstration that the controlled-NOT and arbitrary one-qubit gates constitute a universal set.

a) If $\boldsymbol{U}$ is any unitary 2×2 matrix with determinant one, find unitary $\boldsymbol{A}, \boldsymbol{B}$, and $\boldsymbol{C}$ such that

$$\boldsymbol{ABC} = \boldsymbol{I} \tag{5.113}$$

$$\boldsymbol{AXBXC} = \boldsymbol{U}. \tag{5.114}$$

Hint: From the Euler angle construction, we know that

$$\boldsymbol{U} = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\phi), \tag{5.115}$$

where, *e.g.*, $\boldsymbol{R}_z(\phi)$ denotes a rotation about the z-axis by the angle ϕ. We also know that, *e.g.*,

$$\boldsymbol{X}\boldsymbol{R}_z(\phi)\boldsymbol{X} = \boldsymbol{R}_z(-\phi). \tag{5.116}$$

b) Consider a two-qubit *controlled phase gate*: it applies $\boldsymbol{U} = e^{i\alpha}\boldsymbol{1}$ to the second qubit if the first qubit has value $|1\rangle$, and acts trivially otherwise. Show that it is actually a one-qubit gate.

c) Draw a circuit using $\Lambda(\boldsymbol{X})$ gates and single-qubit gates that implements controlled-$\boldsymbol{U}$, where $\boldsymbol{U}$ is an arbitrary 2×2 unitary transformation.

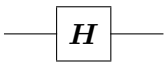
5.3 Universal quantum gates I

In this exercise and the two that follow, we will establish that several simple sets of gates are universal for quantum computation.

The *Hadamard transformation* $\boldsymbol{H}$ is the single-qubit gate that acts in the standard basis $\{|0\rangle, |1\rangle\}$ as

$$\boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \; ; \tag{5.117}$$

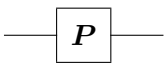
in quantum circuit notation, we denote the Hadamard gate as



The single-qubit *phase gate* $\boldsymbol{P}$ acts in the standard basis as

$$\boldsymbol{P} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \; , \tag{5.118}$$

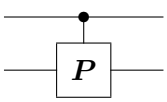
and is denoted



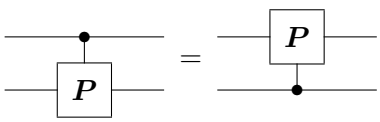
A two-qubit *controlled phase gate* $\Lambda(\boldsymbol{P})$ acts in the standard basis $\{|00\rangle, 01\rangle, |10\rangle, |11\rangle\}$ as the diagonal 4×4 matrix

$$\Lambda(\boldsymbol{P}) = \mathrm{diag}(1, 1, 1, i) \tag{5.119}$$

and can be denoted

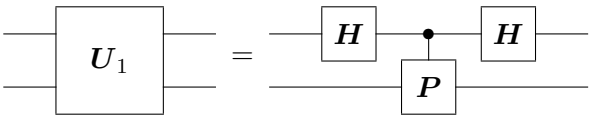


Despite this misleading notation, the gate $\Lambda(\boldsymbol{P})$ actually acts symmetrically on the two qubits:

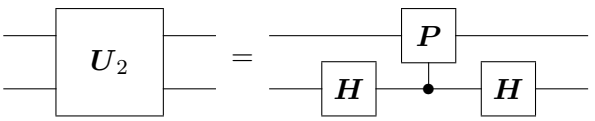


We will see that the two gates $\boldsymbol{H}$ and $\Lambda(\boldsymbol{P})$ comprise a *universal gate set* – any unitary transformation can be approximated to arbitrary accuracy by a quantum circuit built out of these gates.

a) Consider the two-qubit unitary transformations $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ defined by quantum circuits



and



Let $|ab\rangle$ denote the element of the standard basis where a labels the upper qubit in the circuit diagram and b labels the lower qubit. Write out $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ as 4×4 matrices in the standard basis. Show that $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ both act trivially on the states

$$|00\rangle, \quad \frac{1}{\sqrt{3}}\left(|01\rangle + |10\rangle + |11\rangle\right) \ . \tag{5.120}$$

b) Thus $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ act nontrivially only in the two-dimensional space spanned by

$$\left\{ \frac{1}{\sqrt{2}}\left(|01\rangle - |10\rangle\right), \frac{1}{\sqrt{6}}\left(|01\rangle + |10\rangle - 2|11\rangle\right) \right\} \ . \tag{5.121}$$

Show that, expressed in this basis, they are

$$\boldsymbol{U}_1 = \frac{1}{4}\begin{pmatrix} 3+i & \sqrt{3}(-1+i) \\ \sqrt{3}(-1+i) & 1+3i \end{pmatrix} \ , \tag{5.122}$$

and

$$\boldsymbol{U}_2 = \frac{1}{4}\begin{pmatrix} 3+i & \sqrt{3}(1-i) \\ \sqrt{3}(1-i) & 1+3i \end{pmatrix} \ . \tag{5.123}$$

c) Now express the action of $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ on this two-dimensional subspace in the form

$$\boldsymbol{U}_1 = \sqrt{i}\left(\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}\hat{n}_1 \cdot \vec{\boldsymbol{\sigma}}\right) \ , \tag{5.124}$$

and

$$\boldsymbol{U}_2 = \sqrt{i}\left(\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}\hat{n}_2 \cdot \vec{\boldsymbol{\sigma}}\right) \ . \tag{5.125}$$

What are the unit vectors $\hat{n}_1$ and $\hat{n}_2$?

d) Consider the transformation $\boldsymbol{U}_2^{-1}\boldsymbol{U}_1$ (Note that $\boldsymbol{U}_2^{-1}$ can also be constructed from the gates $\boldsymbol{H}$ and $\Lambda(\boldsymbol{P})$.) Show that it performs a rotation with half-angle $\theta/2$ in the two-dimensional space spanned by the basis eq.(5.121), where $\cos(\theta/2) = 1/4$.

5.4 Universal quantum gates II

We have now seen how to compose our fundamental quantum gates to perform, in a two-dimensional subspace of the four-dimensional Hilbert space of two qubits, a rotation with $\cos(\theta/2) = 1/4$. In this exercise, we will show that the angle θ is not a rational multiple of π. Equivalently, we will show that

$$e^{i\theta/2} \equiv \cos(\theta/2) + i\sin(\theta/2) = \frac{1}{4}\left(1 + i\sqrt{15}\right) \tag{5.126}$$

is not a root of unity: there is no finite integer power n such that $(e^{i\theta/2})^n = 1$.

Recall that a *polynomial of degree* n is an expression

$$P(x) = \sum_{k=0}^{n} a_k x^k \tag{5.127}$$

with $a_n \neq 0$. We say that the polynomial is *rational* if all of the a_k's are rational numbers, and that it is *monic* if $a_n = 1$. A polynomial is *integral* if all of the a_k's are integers, and an integral polynomial is *primitive* if the greatest common divisor of $\{a_0, a_1, \ldots, a_n\}$ is 1.

a) Show that the monic rational polynomial of minimal degree that has $e^{i\theta/2}$ as a root is

$$P(x) = x^2 - \frac{1}{2}x + 1 \ . \tag{5.128}$$

The property that $e^{i\theta/2}$ is not a root of unity follows from the result (*a*) and the

Theorem *If a is a root of unity, and $P(x)$ is a monic rational polynomial of minimal degree with $P(a) = 0$, then $P(x)$ is integral.*

Since the minimal monic rational polynomial with root $e^{i\theta/2}$ is not integral, we conclude that $e^{i\theta/2}$ is not a root of unity. In the rest of this exercise, we will prove the theorem.

b) By "long division" we can prove that if $A(x)$ and $B(x)$ are rational polynomials, then there exist rational polynomials $Q(x)$ and $R(x)$ such that

$$A(x) = B(x)Q(x) + R(x) \ , \tag{5.129}$$

where the "remainder" $R(x)$ has degree less than the degree of $B(x)$. Suppose that $a^n = 1$, and that $P(x)$ is a rational polynomial of minimal degree such that $P(a) = 0$. Show that there is a rational polynomial $Q(x)$ such that

$$x^n - 1 = P(x)Q(x) \ . \tag{5.130}$$

c) Show that if $A(x)$ and $B(x)$ are both primitive integral polynomials, then so is their product $C(x) = A(x)B(x)$. **Hint**: If $C(x) = \sum_k c_k x^k$ is not primitive, then there is a prime number p that divides all of the c_k's. Write $A(x) = \sum_l a_l x^l$, and $B(x) = \sum_m b_m x^m$, let a_r denote the coefficient of lowest order in $A(x)$ that is not divisible by p (which must exist if $A(x)$ is primitive), and let b_s denote the coefficient of lowest order in $B(x)$ that is not divisible by p. Express the product $a_r b_s$ in terms of c_{r+s} and the other a_l's and b_m's, and reach a contradiction.

d) Suppose that a monic integral polynomial $P(x)$ can be factored into a product of two monic rational polynomials, $P(x) = A(x)B(x)$. Show that $A(x)$ and $B(x)$ are integral. **Hint:** First note that we may write $A(x) = (1/r) \cdot \tilde{A}(x)$, and $B(x) = (1/s) \cdot \tilde{B}(x)$, where r, s are positive integers, and $\tilde{A}(x)$ and $\tilde{B}(x)$ are primitive integral; then use (*c*) to show that $r = s = 1$.

e) Combining (*b*) and (*d*), prove the theorem.

What have we shown? Since $\boldsymbol{U}_2^{-1}\boldsymbol{U}_1$ is a rotation by an irrational multiple of π, the powers of $\boldsymbol{U}_2^{-1}\boldsymbol{U}_1$ are dense in a $U(1)$ subgroup. Similar reasoning shows that $\boldsymbol{U}_1\boldsymbol{U}_2^{-1}$ is a rotation by the same angle about a different axis, and therefore its powers are dense in another $U(1)$ subgroup. Products of elements of these two noncommuting $U(1)$ subgroups are dense in the $SU(2)$ subgroup that contains both $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$.

Furthermore, products of $\Lambda(\boldsymbol{P})\boldsymbol{U}_2^{-1}\boldsymbol{U}_1\Lambda(\boldsymbol{P})^{-1}$ and $\Lambda(\boldsymbol{P})\boldsymbol{U}_1\boldsymbol{U}_2^{-1}\Lambda(\boldsymbol{P})^{-1}$ are dense in another $SU(2)$, acting on the span of

$$\left\{ \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle), \frac{1}{\sqrt{6}}(|01\rangle + |10\rangle - 2i|11\rangle) \right\} \ . \tag{5.131}$$

Together, these two $SU(2)$ subgroups close on the $SU(3)$ subgroup that acts on the three-dimensional space orthogonal to $|00\rangle$. Conjugating this $SU(3)$ by $\boldsymbol{H} \otimes \boldsymbol{H}$ we obtain another $SU(3)$ acting on the three dimensional space orthogonal to $|+,+\rangle$, where

$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. The only subgroup of $SU(4)$ that contains both of these $SU(3)$ subgroups is $SU(4)$ itself.

Therefore, the circuits constructed from the gate set $\{\boldsymbol{H}, \Lambda(\boldsymbol{P})\}$ are dense in $SU(4)$ — we can approximate any two-qubit gate to arbitrary accuracy, which we know suffices for universal quantum computation. Whew!

5.5 Universal quantum gates III

We have shown that the gate set $\{\boldsymbol{H}, \Lambda(\boldsymbol{P})\}$ is universal. Thus any gate set from which both $\boldsymbol{H}$ and $\Lambda(\boldsymbol{P})$ can be constructed is also universal. In particular, we can see that $\{\boldsymbol{H}, \boldsymbol{P}, \Lambda^2(\boldsymbol{X})\}$and $\{\boldsymbol{H}, \boldsymbol{T}, \Lambda(\boldsymbol{X})\}$ are universal gates sets, where $\boldsymbol{T} = \exp\left(-i\frac{\pi}{8}\boldsymbol{Z}\right)$.

a) It is sometimes convenient to characterize a quantum gate by specifying the action of the gate when it conjugates a Pauli operator. Show that $\boldsymbol{H}$ and $\boldsymbol{P}$ have the properties

$$\boldsymbol{HXH} = \boldsymbol{Z} \ , \quad \boldsymbol{HYH} = -\boldsymbol{Y} \ , \quad \boldsymbol{HZH} = \boldsymbol{X} \ , \quad (5.132)$$

and

$$\boldsymbol{PXP}^{-1} = \boldsymbol{Y} \ , \quad \boldsymbol{PYP}^{-1} = -\boldsymbol{X} \ , \quad \boldsymbol{PZP}^{-1} = \boldsymbol{Z} \ . (5.133)$$

b) Note that, since $\boldsymbol{P}^{-1} = \boldsymbol{P}^3$, the gate $\boldsymbol{K} = \boldsymbol{HP}^{-1}\boldsymbol{HPH}$ can be constructed using $\boldsymbol{H}$ and $\boldsymbol{P}$. Show that

$$\boldsymbol{KXK} = \boldsymbol{Y} \ , \quad \boldsymbol{KYK} = \boldsymbol{X} \ , \quad \boldsymbol{KZK} = -\boldsymbol{Z} \ . \quad (5.134)$$

c) Construct circuits for $\Lambda^2(\boldsymbol{Y})$ and $\Lambda^2(\boldsymbol{Z})$ using the gate set $\{\boldsymbol{H}, \boldsymbol{P}, \Lambda^2(\boldsymbol{X})\}$. Then complete the proof of universality for this gate set by constructing $\Lambda(\boldsymbol{P}) \otimes \boldsymbol{I}$ using $\Lambda^2(\boldsymbol{X})$, $\Lambda^2(\boldsymbol{Y})$, and $\Lambda^2(\boldsymbol{Z})$.

d) Show that $\{\boldsymbol{H}, \boldsymbol{T}, \Lambda(\boldsymbol{X})\}$ is a universal gate set by constructing a circuit for $\Lambda(\boldsymbol{P})$ from $\Lambda(\boldsymbol{X})$ and $\boldsymbol{T}$. **Hint**: Observe that $\boldsymbol{T}^2 = e^{-i\pi/4}\boldsymbol{P}$, then use the construction suggested in Exercise 5.2, noting that $\boldsymbol{T}^{-1}\boldsymbol{T}^{-1}\boldsymbol{T}^2 = \boldsymbol{I}$ and $\boldsymbol{T}^{-1}\boldsymbol{XT}^{-1}\boldsymbol{XT}^2 = \boldsymbol{T}^2$.

The Toffoli gate $\Lambda^2(\boldsymbol{X})$ is universal for reversible classical computation. What must be added to realize the full power of quantum computing? We have just seen that the single-qubit gates $\boldsymbol{H}$ and $\boldsymbol{P}$, together with the Toffoli gate, are adequate for reaching any unitary transformation. But in fact, just $\boldsymbol{H}$ and $\Lambda^2(\boldsymbol{X})$ suffice to efficiently simulate any quantum computation.

Of course, since $\boldsymbol{H}$ and $\Lambda^2(\boldsymbol{X})$ are both real orthogonal matrices, a circuit composed from these gates is necessarily real — there are complex n-qubit unitaries that cannot be constructed with these tools. But a 2^n-dimensional complex vector space is isomorphic to a 2^{n+1}–dimensional real vector space. A complex vector can be encoded by a real vector according to

$$|\psi\rangle = \sum_x \psi_x |x\rangle \mapsto |\tilde{\psi}\rangle = \sum_x (\text{Re } \psi_x)|x,0\rangle + (\text{Im } \psi_x)|x,1\rangle \ , \quad (5.135)$$

and the action of the unitary transformation $\boldsymbol{U}$ can be represented by a real orthogonal matrix $\tilde{U}_R$ defined as

$$\begin{aligned} U_R : \qquad & |x,0\rangle \mapsto (\text{Re } U)|x\rangle \otimes |0\rangle + (\text{Im } U)|x\rangle \otimes |1\rangle \ , \\ & |x,1\rangle \mapsto -(\text{Im } U)|x\rangle \otimes |0\rangle + (\text{Re } U)|x\rangle \otimes |1\rangle \ . \end{aligned} \quad (5.136)$$

To show that the gate set $\{\boldsymbol{H}, \Lambda^2(\boldsymbol{X})\}$ is "universal," it suffices to demonstrate that the real encoding $\Lambda(\boldsymbol{P})_R$ of $\Lambda(\boldsymbol{P})$ can be constructed from $\Lambda^2(\boldsymbol{X})$ and $\boldsymbol{H}$.

d) Verify that $\Lambda(\boldsymbol{P})_R = \Lambda^2(\boldsymbol{XZ})$.

e) Use $\Lambda^2(\boldsymbol{X})$ and $\boldsymbol{H}$ to construct a circuit for $\Lambda^2(\boldsymbol{XZ})$.

Thus, the classical Toffoli gate does not need much help to unleash the power of quantum computing. In fact, *any* nonclassical single-qubit gate (one that does not preserve the computational basis), combined with the Toffoli gate, is sufficient.

5.6 Universality from any entangling two-qubit gate

We say that a two-qubit unitary quantum gate is *local* if it is a tensor product of single-qubit gates, and that the two-qubit gates $\boldsymbol{U}$ and $\boldsymbol{V}$ are *locally equivalent* if one can be transformed to the other by local gates:

$$\boldsymbol{V} = (\boldsymbol{A} \otimes \boldsymbol{B})\boldsymbol{U}(\boldsymbol{C} \otimes \boldsymbol{D}) \ . \quad (5.137)$$

It turns out (you are not asked to prove this) that every two-qubit gate is locally equivalent to a gate of the form:

$$\boldsymbol{V}(\theta_x, \theta_y, \theta_z) = \exp\left[i\left(\theta_x \boldsymbol{X} \otimes \boldsymbol{X} + \theta_y \boldsymbol{Y} \otimes \boldsymbol{Y} + \theta_z \boldsymbol{Z} \otimes \boldsymbol{Z}\right)\right] \ , \quad (5.138)$$

where

$$-\pi/4 < \theta_x \leq \theta_y \leq \theta_z \leq \pi/4 \ . \quad (5.139)$$

a) Show that $\boldsymbol{V}(\pi/4, \pi/4, \pi/4)$ is (up to an overall phase) the **SWAP** operation that interchanges the two qubits:

$$\textbf{SWAP}\,(|\psi\rangle \otimes |\phi\rangle) = |\phi\rangle \otimes |\psi\rangle \ . \tag{5.140}$$

b) Show that $\boldsymbol{V}(0, 0, \pi/4)$ is locally equivalent to the CNOT gate $\Lambda(\boldsymbol{X})$.

As discussed in the lecture notes, the CNOT gate $\Lambda(\boldsymbol{X})$ together with arbitrary single-qubit gates form a universal gate set. But in fact there is nothing special about the the CNOT gate in this regard. *Any* two-qubit gate $\boldsymbol{U}$, when combined with arbitrary single-qubit gates, suffices for universality *unless* $\boldsymbol{U}$ is either local or locally equivalent to **SWAP**.

To demonstrate that $\boldsymbol{U}$ is universal when assisted by local gates it suffices to construct $\Lambda(\boldsymbol{X})$ using a circuit containing only local gates and $\boldsymbol{U}$ gates.

Lemma *If* $\boldsymbol{U}$ *is locally equivalent to* $\boldsymbol{V}(\theta_x, \theta_y, \theta_z)$*, then* $\Lambda(\boldsymbol{X})$ *can be constructed from a circuit using local gates and* $\boldsymbol{U}$ *gates except in two cases: (1)* $\theta_x = \theta_y = \theta_z = 0$ *(*$\boldsymbol{U}$ *is local), (2)* $\theta_x = \theta_y = \theta_z = \pi/4$ *(*$\boldsymbol{U}$ *is locally equivalent to* **SWAP***)..*

You will prove the Lemma in the rest of this exercise.

c) Show that:

$$\begin{aligned}
(\boldsymbol{I}\otimes\boldsymbol{X})\boldsymbol{V}(\theta_x,\theta_y,\theta_z)(\boldsymbol{I}\otimes\boldsymbol{X})\boldsymbol{V}(\theta_x,\theta_y,\theta_z) &= \boldsymbol{V}(2\theta_x,0,0)\ ,\\
(\boldsymbol{I}\otimes\boldsymbol{Y})\boldsymbol{V}(\theta_x,\theta_y,\theta_z)(\boldsymbol{I}\otimes\boldsymbol{Y})\boldsymbol{V}(\theta_x,\theta_y,\theta_z) &= \boldsymbol{V}(0,2\theta_y,0)\ ,\\
(\boldsymbol{I}\otimes\boldsymbol{Z})\boldsymbol{V}(\theta_x,\theta_y,\theta_z)(\boldsymbol{I}\otimes\boldsymbol{Z})\boldsymbol{V}(\theta_x,\theta_y,\theta_z) &= \boldsymbol{V}(0,0,2\theta_z)\ .
\end{aligned} \tag{5.141}$$

d) Show that $\boldsymbol{V}(0, 0, \theta)$ is locally equivalent to the controlled rotation $\Lambda[\boldsymbol{R}(\hat{n}, 4\theta)]$, where $\boldsymbol{R}(\hat{n}, 4\theta) = \exp[-2i\theta(\hat{n}\cdot\boldsymbol{\sigma})]$, for an arbitrary axis of rotation $\hat{n}$. (Here $\boldsymbol{\sigma} = (\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})$.)

e) Now use the results of (*c*) and (*d*) to prove the Lemma.

5.7 BQP is contained in PP

We have seen that BQP (the class of decision problems that can be solved efficiently by a quantum computer) is contained in the classical complexity class PSPACE (the decision problems that can be solved using a polynomial-size memory). The purpose of this problem is to establish an inclusion that is presumed to be stronger: BQP is contained in PP. A decision problem is in PP ("probabilistic

polynomial time") if it can be solved in polynomial time by a randomized classical computation with probability of success greater than 1/2. (In contrast to the class BPP, the success probability is not required to exceed 1/2 by a positive constant independent of the input size.)

When a decision problem is solved by a quantum computer, one particular qubit may be designated as the "answer qubit" — the qubit that is measured to decide the answer. If the quantum circuit builds the unitary transformation $\boldsymbol{U}$, which acts on the input state $|0\rangle$, then the probability distribution governing the answer bit x can be expressed as

$$P(x) = \sum_y |\langle x, y|\boldsymbol{U}|0\rangle|^2 \ , \tag{5.142}$$

where $|y\rangle$ denotes a basis state for the "junk" output qubits that are not measured. If the problem is in BQP, then there is a polynomial-size uniform quantum circuit family such that $P(x) \geq 2/3$ when x is the correct value of the output. For convenience, we are assuming here that the input to the Boolean function being evaluated is encoded by choosing the unitary U to depend on the this input; hence we may suppose that the unitary always acts on the fixed quantum state $|0\rangle$.

a) Show that if a problem is in BQP then there is a polynomial-size uniform circuit family (where the circuit depends on the input to the problem) such that $|\langle 0|\boldsymbol{U}|0\rangle|^2 \geq 2/3$ if the correct output is 0 and $|\langle 0|\boldsymbol{U}|0\rangle|^2 < 1/3$ if the correct output is 1. **Hint**: We want to avoid summing over the state of the unmeasured "junk." Recall the trick we used to remove the junk produced by a reversible classical circuit.

If we want to simulate on a classical computer the quantum computation that solves the problem, then, it suffices to estimate the single matrix element $|\langle 0|\boldsymbol{U}|0\rangle|$ to reasonable accuracy.

Now recall that we saw in Exercise 3.3 that the quantum gate set $\{\boldsymbol{H}, \Lambda^2(\boldsymbol{X})\}$, where $\boldsymbol{H}$ denotes the Hadamard gate and $\Lambda^2(\boldsymbol{X})$ is the Toffoli gate, is universal for quantum computation. The Toffoli gate is classical, but the Hadamard gate takes the computational basis states of a qubit to superpositions of basis states:

$$\boldsymbol{H} : |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y\in\{0,1\}} (-1)^{xy}|y\rangle \ . \tag{5.143}$$

Note that, since $\boldsymbol{H}^2 = I$, we are free to insert a pair of Hadamard gates acting on all qubits following each Toffoli gate, without changing anything.

Suppose that $\boldsymbol{U}$ is expressed as a circuit constructed from Toffoli gates and Hadamard gates, where the circuit contains h Hadamard gates, which we label by $i = 0, 1, 2, \ldots, h-1$. By inserting the partition of unity $I = \sum_{x_i \in \{0,1\}} |x_i\rangle\langle x_i|$ following each Hadamard gate, we may write the matrix element $\langle 0|\boldsymbol{U}|0\rangle$ as a sum of 2^h terms, with each term arising from a "computational path" indexed by the bit string $x = x_{h-1}x_{h-2}\ldots x_1 x_0$. Each term has absolute value $2^{-h/2}$, arising from the factor $2^{-1/2}$ that accompanies each Hadamard gate, and a phase ± 1 depending on x that counts modulo 2 the number of Hadamard gates for which the input and output both have the value 1. Note that each output bit from every Toffoli gate, and therefore the input to each Hadamard gate, can be expressed as a polynomial in the $\{x_i\}$ that depends on the circuit.

b) Show that, if $\boldsymbol{U}$ is a unitary transformation constructed from a circuit of size L, built from Toffoli and Hadamard gates, then there is a polynomial function $\phi(x)$ of $h \leq 3L$ binary variables $\{x_{h-1}, x_{h-2}, \ldots x_1, x_0\}$ such that

$$\langle 0|\boldsymbol{U}|0\rangle = \frac{1}{\sqrt{2^h}} \left(N_0 - N_1\right) \ , \tag{5.144}$$

where N_0 is the number of values of x for which $\phi(x) = 0 \pmod 2$ and N_1 is the number of values of x for which $\phi(x) = 1 \pmod 2$. Furthermore, $\phi(x)$ is of degree at most three, is a sum of at most $2h$ monomials, and can be computed efficiently from a description of the circuit.

Thus, if the circuit is polynomial size, the function $\phi(x)$ can be evaluated efficiently for any of the 2^h possible values of its input x. The difference $N_0 - N_1$ is at most $\sqrt{2^h}$, and we may simulate the quantum circuit (solving a problem in BQP by distinguishing $|\langle 0|U|0\rangle|^2 \geq 2/3$ from $|\langle 0|U|0\rangle|^2 < 1/3$) by estimating $N_0 - N_1$ to sufficient accuracy.

Up until now, the computational problems we have encountered have typically involved determining whether a solution to an equation exists, exhibiting a solution, or verifying the correctness of a solution. Here we have encountered a problem that appears to be intrinsically harder: *counting* (approximately) the number of solutions.

We could attempt to do the counting by a randomized computation, estimating $N_0 - N_1$ by evaluating $\phi(x)$ for a sample of randomly selected values of x. Unfortunately, the number of inputs mapped to 0 and to 1 are nearly in balance, which makes it hard to estimate $N_0 - N_1$. We can think of $\phi(x)$ as a coin with an exponentially small bias; determining the basis to reasonable accuracy requires an exponentially large number of trials.

But the task of simulating BQP within the class PP is easier than that — it is enough to be able to distinguish $|\langle 0|\boldsymbol{U}|0\rangle|^2 \geq 2/3$ from $|\langle 0|\boldsymbol{U}|0\rangle|^2 < 1/3$ with success probability $1/2 + \delta$ as long as δ is positive, even if it is exponentially small. This crude estimate can be achieved with a polynomial number of trials. Therefore, BQP $\subseteq$ PP.

The class PP is certainly contained in PSPACE, because we can determine the probability of acceptance for a randomized computation by simulating all of the possible computational paths that occur for all possible outcomes of the coin flips. There may be an exponentially large number of paths, but we can run through the paths one at a time, while maintaining a count of how many of the computations have accepted. This can be done with polynomial space.