

41076: Methods in Quantum Computing

‘Models of computation’ Module

Dr. Mária Kieferová and Dr. Min-Hsiu Hsieh

*Centre for Quantum Software & Information, Faculty of Engineering and Information Technology,
University of Technology Sydney*

Abstract

Contents to be covered in this lecture are

1. Motivation behind quantum computing
2. Models of computation
3. Quantum circuits

1 Why do we study quantum computing

The digital computer was one of the biggest, if not the biggest technological breakthroughs in the 20th century. Research on the theory of computation and information together with material science (development of transistors) led to the development of computers as we know them today. While there were many proposed models of computation, such as Turing machines (TM), context-free grammars and λ -calculus, they all led to models of computing that are more or less equivalent. An abstract version of our computers is also such a model. Specifically, all these models can compute (in an arbitrarily long time) the same class of problems. There are also problems that are uncomputable for all of these models of computers.

While computability is a fascinating theory, we are more often interested in what is efficiently computable. In this class, efficient will stand for polynomial in the size of the input. Surprisingly, it turned out that the class of problems that are efficiently computable is also the same for many models including TMs. For a long time, the only models stronger than TMs we models that are not physically realistic - for example, these models of computers would have the ability to exactly represent any real number or travel back in time. This made people think about computability as a physics law, similarly to the Third Law of thermodynamics - physically realistic models cannot be exponentially faster than TMs. Thus, any physically realistic model could be efficiently simulated by another physically realistic model.

As people started thinking about the physical nature of computation, a person named Feynman (you might have heard of him) pointed out that these models don't seem to be able to efficiently simulate quantum mechanics. He then conjectured that if we could build a quantum computer it would be different from TMs and better suited for simulating physical systems. This piqued interest of many people and quantum computing was born.

Exercise 1 (Discussion) *Why are you interested in quantum computing? What topics are you working on and why did you choose them?*

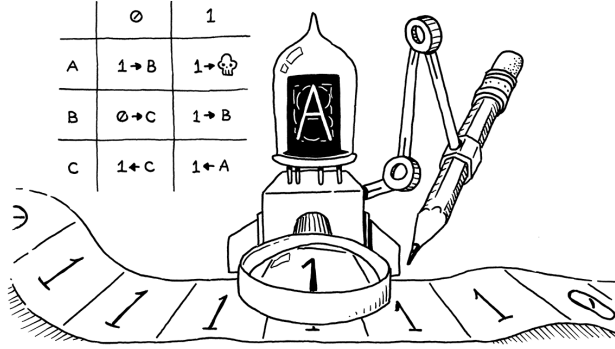


Figure 1: An illustration of a TM from www.craftinginterpreters.com. A computation is performed by the head moving around the tape according to the transition function.

2 Models of computation

To understand what makes quantum computing different from classical computers, we need to first understand the foundation of classical computation.

2.1 Turing machine

A Turing machine (TM) is a theoretical model of computation that on a high level describes how classical computers work. A TM performs computation by manipulating symbols on a tape according to a set of rules. A Turing machine is formally specified by a 7-tuple:

- Γ a non-empty alphabet, i.e. a set of allowed symbols
- $b \in \Gamma$ a blank symbol
- $\Sigma \subseteq \Gamma$ a set of symbols that initially appear on the tape
- Q a finite set of states of the machine
- $q_0 \in Q$ the initial state
- $F \subseteq Q$ set of accepting states. If the TM reaches one of these states, the computation finishes and the input is accepted ("yes").
- $\delta : Q \setminus F \times \Gamma \rightarrow Q \times \Gamma \times \{\text{left, right}\}$ is the transition function. Given the state of the machine and the symbol of the tape, TM changes its state, rewrites the symbol on the tape and the head moves left or right on the tape.

We can distinguish between a deterministic and a non-deterministic TM. The deterministic Turing machine (DTM) uses a fixed set of rules to determine its future actions; while the Non-deterministic Turing machine is allowed to explore multiple possible future actions from a given state. One might also consider a probabilistic Turing machine which is a deterministic Turing machine that has the ability to flip a coin (generate a random Boolean number) and perform deterministic action based on the outcome.

As a computational model, the Turing machine is able to compute an enormous variety of functions, such as basic arithmetical operations, searching, and simulation of all operations performed

on a modern computer. Surprisingly, many other theoretical models have been proven to be computationally equivalent to TMs¹. The belief that a Turing machine describes the computability of reasonable computational models is known as the Church-Turing thesis:

Claim 2 (Church-Turing thesis) *A Turing machine can simulate any realistic model of computation.*

We can add additional constraints on TMs. For example, we might require them to complete the computation in time polynomial in the size of the input or restrict the amount of space they can use on the tape. These restrictions then give rise to complexity classes such as **P** for deterministic polynomial time or **NP** for nondeterministic polynomial time. A stronger version of the Church-Turing thesis is concerned with the complexity of computation.

Theorem 3 (Extended Church–Turing thesis) *A probabilistic Turing machine can efficiently simulate any realistic model of computation.*

If quantum physics cannot be efficiently simulable on a classical computer, and we now believe that it is not after trying for many years, and quantum computers can be eventually built, it would mean that the extended Church-Turing thesis is false.

We will learn more about Turing machines and complexity classes in Lecture 7.

2.2 Logical circuits

Another computational model that is equivalent to Turing machines is logical circuits. Denote $\mathbb{B}^n := \mathbb{Z}_2^n$. Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be a Boolean function that takes an n -bit string as input and outputs an m -bit string. Let \mathcal{G} be a collection of basic logic gates. A *Boolean circuit* for f is a sequence of gates $\{g_1, \dots, g_L\} \in \mathcal{G}$ which converts an input $\mathbf{x} \in \mathbb{B}^n$ to the output $\mathbf{y} \in \mathbb{B}^m$ with a fixed size of K auxiliary bits (called memory cell). Here L is called the size of the circuit. A list of elementary Boolean logic gates is given in Table 1.

Important questions in the circuits model include

- whether there exists an *universal* set of elementary gates so that any function computable in this model can be computed by a sequence of basic gates from this set.
- how to efficiently simulate a larger circuit with gates from the basic gate sets.
- how big is the circuit size L required with the input size to the function f ?

These questions are also relevant for quantum circuits.

Unlike TMs when we have an infinite tape, every logical circuit operates only on inputs of a fixed size n . To make turn circuits into a computational model independent of input size, we need to introduce a family of circuits that all compute the same function for different-sized inputs. These circuit families must be *uniform* which means that it must be a way to efficiently compute the circuit for each input size from the description.

Exercise 4 *Show that the NAND and FANOUT (copy) are universal for computation, i.e. they can be used to express all possible truth tables.*

¹This is a statement about computability, not complexity. There are problems such as the halting problem that are not computable by TMs in any amount of time.

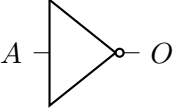
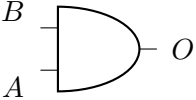
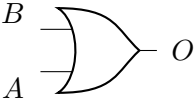
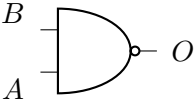
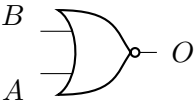
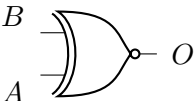
Name	Gates	Truth Table															
NOT		<table border="1"> <thead> <tr> <th><i>A</i></th> <th><i>O</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	<i>A</i>	<i>O</i>	0	1	1	0									
<i>A</i>	<i>O</i>																
0	1																
1	0																
AND		<table border="1"> <thead> <tr> <th><i>A</i></th> <th><i>B</i></th> <th><i>O</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	<i>A</i>	<i>B</i>	<i>O</i>	0	0	0	1	0	0	0	1	0	1	1	1
<i>A</i>	<i>B</i>	<i>O</i>															
0	0	0															
1	0	0															
0	1	0															
1	1	1															
OR		<table border="1"> <thead> <tr> <th><i>A</i></th> <th><i>B</i></th> <th><i>O</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	<i>A</i>	<i>B</i>	<i>O</i>	0	0	0	1	0	1	0	1	1	1	1	1
<i>A</i>	<i>B</i>	<i>O</i>															
0	0	0															
1	0	1															
0	1	1															
1	1	1															
NAND		<table border="1"> <thead> <tr> <th><i>A</i></th> <th><i>B</i></th> <th><i>O</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	<i>A</i>	<i>B</i>	<i>O</i>	0	0	1	1	0	1	0	1	1	1	1	0
<i>A</i>	<i>B</i>	<i>O</i>															
0	0	1															
1	0	1															
0	1	1															
1	1	0															
NOR		<table border="1"> <thead> <tr> <th><i>A</i></th> <th><i>B</i></th> <th><i>O</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	<i>A</i>	<i>B</i>	<i>O</i>	0	0	1	1	0	0	0	1	0	1	1	0
<i>A</i>	<i>B</i>	<i>O</i>															
0	0	1															
1	0	0															
0	1	0															
1	1	0															
XNOR		<table border="1"> <thead> <tr> <th><i>A</i></th> <th><i>B</i></th> <th><i>O</i></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	<i>A</i>	<i>B</i>	<i>O</i>	0	0	1	1	0	0	0	1	0	1	1	1
<i>A</i>	<i>B</i>	<i>O</i>															
0	0	1															
1	0	0															
0	1	0															
1	1	1															

Table 1: Boolean logic gates

- (a) Show that the NOT gate can be simulated using a single NAND gate.
- (b) Show that the AND gate can be simulated with a constant number of NAND gates.
- You might use additional bits initialized to 0 or 1.

2.3 Reversible circuits

Logical circuits often map multiple inputs on the same output. An example is the AND gate that maps $11 \rightarrow 1$ and all other inputs to 0. Consequently, such a gate cannot be inverted - if we only know that the output is 0, we cannot with certainty compute what was the input. Logical circuits that can be inverted are known as reversible circuits.

It is convenient to represent reversible circuits as matrices which can be obtained from truth tables. We can represent logical 0 and logical 1 as vectors

$$0_L = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad (1)$$

$$1_L = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \quad (2)$$

and similarly for 2 bits. With multiple qubit one can construct a basis such as

$$00_L = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, 01_L = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, 10_L = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, 11_L = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \quad (3)$$

Quantum computing builds on this notation.

Exercise 5 What operations in Table 1 are reversible? What are the inverse operations to the reversible gates in Tables 1 and 2?

Any function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ can be modified to a reversible function f_\oplus such that $f_\oplus : \mathbb{B}^n \times \mathbb{B}^m \rightarrow \mathbb{B}^n \times \mathbb{B}^m$ by

$$f_\oplus : (\mathbf{u}, \mathbf{v}) \rightarrow (\mathbf{u}, \mathbf{v} \oplus f(\mathbf{u})). \quad (4)$$

In other words, f_\oplus copies the output of the original Boolean function f to the second register. It is not difficult to verify that f_\oplus is a bijection. We will denote $\mathcal{G}_\oplus := \{f_\oplus : \forall f \in \mathcal{G}\}$.

While the function f_\oplus is a bijection, the computation might produce “garbage”, i.e., extra information which has to be forgotten after the computation is done. Specifically, in classical computers, extra auxiliary bits are allowed in the computation process, and these auxiliary bits might contain information related to the input and output bits. Let $X \in \mathbb{B}^n$ and $W \in \mathbb{B}^k$ be the input and memory registers, where $X \cap W = \emptyset$. Let $Y \subset X \cup W$ be the output register of size m , and let $Z = X \cup W \setminus Y$. A sequence of gates, represented by $\mathfrak{F} : (X, W) \rightarrow (Y, Z)$, that computes $\mathbf{y} = f(\mathbf{x})$ for some input \mathbf{x} in the register X and $\mathbf{0}$ bit string in the memory register W initially:

$$\mathfrak{F} : (\mathbf{x}, \mathbf{0}) \rightarrow (\mathbf{y}, \mathbf{g}), \quad (5)$$

where $\mathbf{g} \in Z \equiv X \cup W \setminus Y$ is some garbage bits that live outside the register Y .

We formally define the *reversible computation* so that these auxiliary bits have to be returned to their initial state.

Definition 6 (Reversible computation) Let $G : \mathbb{B}^n \rightarrow \mathbb{B}^n$ be any bijective function. A sequence of bijective operations $\{r_1, \dots, r_L\}$ is said to compute G reversibly if their composition $r_L \circ \dots \circ r_1$ maps $(\mathbf{x}, \mathbf{0})$ to $(G(\mathbf{x}), \mathbf{0})$ for every $\mathbf{x} \in \mathbb{B}^n$, where $\mathbf{0}$ is the ground state of some auxiliary register.

Exercise 7 Construct a reversible added that for $a, b \in \{0, 1\}$ computes $a+b$ in an additional register. Hint: ²

Theorem 8 Any logical operation can be implemented by a reversible circuit with constant overhead.

Proof: in the next exercise

Exercise 9 Show that the Toffoli gate can be used to simulate NOT and AND. You can use additional bits initialized to 0 or 1.

Exercise 10 Convince yourself that a logical circuit C such that C^2 (applying C twice) would create NOT ($C^2 = \text{NOT}$) does not exist.

3 Quantum circuits

In an abstract manner, a *circuit* is a model of computation that aims to compute a function, in which the output of the function is produced through a sequence of basic elements, called *gates*.

The following text assumes some level of familiarity with qubits and quantum circuits. For an introductory text, see for example <https://qiskit.org/textbook-beta/>.

4 Qubits

Qubits are a generalization of logical bits. Define

$$0_L \rightarrow |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6}$$

$$1_L \rightarrow |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{7}$$

A pure state on a qubit can be represented as a superposition of basis states

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{8}$$

where $|\alpha|^2 + |\beta|^2 = 1$. The numbers α and β are known as *amplitudes*. Note that this notation is superfluous because it allows for a global phase. We say that the global state is not physical because it cannot be detectable. Thus, states that differ only by a global phase, for example $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ and $\frac{|1\rangle - |0\rangle}{\sqrt{2}}$ correspond to the same physical state. Another way of representing a qubit state is

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle. \tag{9}$$


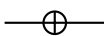
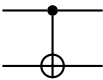
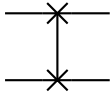
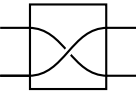
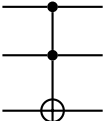
Name	Gates	Matrix
Pauli-X	 or 	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
CNOT		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
SWAP	 or 	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Table 2: Reversible gates

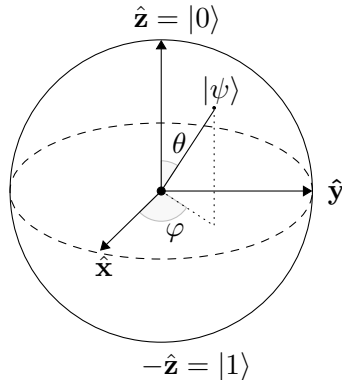


Figure 2: Bloch sphere of $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle$.

This representation has the advantage of ensuring that the quantum state is normalized. A qubit can be represented on a Bloch sphere as in Fig. 2.

With multiple qubits, one can use classical bit states from (3) as a basis. Any state will be a superposition basis states. An n qubit state is represented by a superposition of basis vectors in a 2^n dimensional Hilbert space. We will study quantum states more closely in the next lecture.

It is important to note that if we are given qubits in a state $|\psi\rangle$, we *do not* have access to its amplitudes. The best we can do to learn about its classical description is to measure the qubit which would lead to sampling from the probability distribution over squared absolute values of amplitudes in some basis.

5 Quantum Gates and Universal Gate Sets

We manipulate quantum states by applying quantum operations, often called gates, to them. Given an initial state $|\psi_0\rangle$, we can apply a gate U to obtain a new state $|\psi_1\rangle$

$$|\psi_1\rangle = U|\psi_0\rangle. \quad (10)$$

One can work out (at least for small enough systems) what the resulting state will be using linear algebra and representing states as vectors and operations as matrices. Collections of basic single-qubit gates and multiple-qubit gates are summarized in Table 3 and 4, respectively.

There are multiple ways to compute how a circuit composed of multiple gates would act. A straightforward but tedious way would be to multiply matrices corresponding to individual gates together. For pen-and-paper calculations, it is often more convenient to compute how the circuit acts on each basis state. Since quantum mechanics is linear, this is sufficient for computing what the circuit does on any input. The last approach expresses matrices in Dirac notation and it is useful when working with sparse operators.

Exercise 11 *Show that quantum operations must be unitary in order to preserve the norm of quantum states. An operation U is unitary if and only if it satisfies $U^{-1} = U^\dagger$.*

²Write a table first for all the options and focus on the first 3 cases first.

5.1 Single qubit gates

Single qubit gates are operations that map a (valid) single-qubit state of another single-qubit state. A list of common single-qubit gates is depicted in Table 3.

Given the simplicity of a single qubit, it is worth asking what is the most general single-qubit gate. We will find a general representation of a single qubit gate as an exercise.

Exercise 12 *Here we will find a general form of single-qubit gates.*

- a Show that for a unitary matrix U , $|\det(U)| = 1$. *Hint: in a footnote*³
- b A global phase of a quantum state is not detectable. In other words, states $|\psi\rangle$ and $e^{i\psi}|\psi\rangle$ represent the same physical state. What consequence will it have for single-qubit gates?
- c Write the most general single qubit gate U using the convention $\det(U) = 1$.

Exercise 13 *Use a Jupyter notebook/other software to understand what transformations single qubit quantum gates apply to a qubit.*

Exercise 14 (optional) *Gates H and S generate a finite group known as the Clifford group on a single qubit. What other well-known gates belong to this group and what else can we say about them?*

We can compute the outcome of a quantum gate on a given quantum state by matrix multiplication. For instance, assume that we apply the Hadamard gate on a qubit in state $|0\rangle$. The computation in vector notation gives:

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad (11)$$

In Dirac notation, this corresponds to the state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. The same applies to multi-qubit gates.

5.2 Quantum gates on multiple qubits

Similarly to single-qubit gates, multiple qubit gates are unitary transformations on qubits. While single-qubit gates are relatively easy to characterize and visualize, the intricacy of quantum computation rapidly increases with the number of qubits. Fortunately, it is possible to choose a small set of gates that are able to approximate arbitrary unitary on multiple qubits (often using exponentially many gates to do so). A set of gates that has this ability is known as a *universal gate set*. Denote $\mathbb{H}^n = \mathcal{H}_2^{\otimes n}$ the space of n copies of 2-dimensional Hilbert space \mathcal{H}_2 . Let \mathbb{U}^n be the collection of unitary operations on \mathbb{H}^n .

³These identities might be useful:

- $\det(A^{-1}) = \frac{1}{\det(A)}$
- $\det(A^\dagger) = (\det(A))^*$
- for a 2-by-2 matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ the determinant $\det(A) = ac - bd$





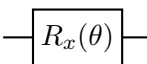
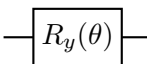
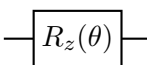
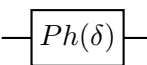



Name	Gates	Matrix
Pauli-X	 or 	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
$R_x(\theta)$		$\begin{bmatrix} \cos \frac{\theta}{2} & i \sin \frac{\theta}{2} \\ i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$
$R_y(\theta)$		$\begin{bmatrix} \cos \frac{\theta}{2} & \sin \frac{\theta}{2} \\ -\sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$
$R_z(\theta)$		$\begin{bmatrix} e^{i\theta/2} & 0 \\ 0 & e^{-i\theta/2} \end{bmatrix}$
$Ph(\delta)$		$\begin{bmatrix} e^{i\delta} & 0 \\ 0 & e^{i\delta} \end{bmatrix}$
Hadamard		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
T		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$

Table 3: Single-qubit gates

Definition 15 A set of gates \mathcal{G} is universal for a computational model⁴ if any function computable in this model can be computed by a circuit that uses only gates in \mathcal{G} .

For the classical circuit model, the $\{\text{NAND}\}$, $\{\text{NOR}\}$ and $\{\text{AND, NOT}\}$ are universal.

Denote by $SU(d)$ the special unitary group of degree d , i.e., collection of $d \times d$ unitary matrices with determinant 1. For the quantum circuit model, the universality in Definition 15 is equivalent to the following.

Definition 16 A set of quantum gates \mathcal{G} is universal if the group generated by \mathcal{G} is dense⁵ in $SU(d)$.

Examples of quantum universal gate sets include

- $\{\text{Toffoli}, H\}$.
- $\{H, T, \text{CNOT}\}$
- $\{\text{CNOT}\} \cup S_1$, where S_1 is the set of single-qubit gates.

Note that the existence of universal gate sets only implies that an arbitrary circuit can be approximated by a finite circuit from the gate set, with no bound on its length. In other words, universality did not promise the simulation of an arbitrary circuit can be done efficiently.

Theorem 17 (Solovay-Kitaev) Given any universal set of gates \mathcal{G} that is closed under inverse, any unitary operation $U \in SU(d)$ can be ε -approximated using $O(\log^c(\frac{1}{\varepsilon}))$ gates from \mathcal{G} for some constant c .

With the seminal result of Theorem 17, we know that a quantum circuit of m constant-qubit gates can be approximated to ε error by a quantum circuit of $O(m \log^c(\frac{m}{\varepsilon}))$ gates from a desired finite universal gate set. This guarantees the efficiency of the simulation. A proof of this theorem can be found in Ref. [3].

We already saw how quantum gates can be applied to qubits in vector notation. Dirac notation (bra-ket) is an alternative form for expressing linear algebra. Expressing vectors (states) in Dirac notation is very popular in quantum computing but it can be used for matrices as well. For single qubit operators, we can express individual matrix elements as follows:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = |0\rangle\langle 0|, \quad \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = |0\rangle\langle 1|, \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = |1\rangle\langle 0|, \quad \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = |1\rangle\langle 1| \quad (12)$$

One can easily prove that the above identities are correct by verifying that they provide the same results in vector and bracket notation. For one of the identities

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0 \quad \text{in bra-ket } |0\rangle \rightarrow 0 \quad (13)$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{in bra-ket } |1\rangle \rightarrow |0\rangle \quad (14)$$

⁴In computer science, a model of computation is a model which describes how an output of a mathematical function is computed given an input. A model describes how units of computations, memories, and communications are organized.

⁵In topology and related areas of mathematics, a subset A of a topological space X is called dense if every point $x \in X$ either belongs to A or is a limit point of A .

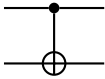
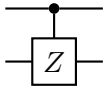
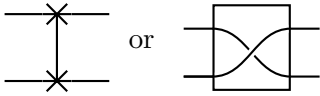
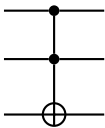
Name	Gates	Matrix
CNOT		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
CZ		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Table 4: Multiple-qubit gates

The operator $|0\rangle\langle 1|$ acts the same way

$$|0\rangle\langle 1||0\rangle = 0 \tag{15}$$

$$|0\rangle\langle 1||1\rangle = |0\rangle \tag{16}$$

We verified that the matrices and the ket-bra operators acts perform the same operation on the basis vectors, therefore they act the same.

We can express common single qubit gates using identities (12).

$$\mathbf{1} = |0\rangle\langle 0| + |1\rangle\langle 1| \text{ does nothing} \tag{17}$$

$$Z = |0\rangle\langle 0| - |1\rangle\langle 1| \text{ flips the sign for } |1\rangle \tag{18}$$

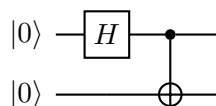
$$X = |0\rangle\langle 1| + |1\rangle\langle 0| \text{ switches } |0\rangle \text{ and } |1\rangle \tag{19}$$

$$H = \frac{1}{\sqrt{2}}(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| - |1\rangle\langle 1|) \text{ switches between } Z \text{ and } X \text{ basis} \tag{20}$$

We see that this notation is useful when the matrices are sparse since the number of ket-bra operators is the same as the number of non-zero matrix elements. Concatenating operators in this formalism is also very simple utilizing the the property $\langle i||j\rangle = \delta_{i,j}$ ⁶ We can extend the notation to multiple qubits using the tensor product/ Let us take the CNOT. If the control qubit is in state $|0\rangle$, CNOT acts as an identity and if it is in state $|1\rangle$ it performs NOT (Pauli X) on target. In the Dirac notation:

$$CNOT = |0\rangle\langle 0| \otimes \mathbf{1} + |1\rangle\langle 1| \otimes X = |0\rangle\langle 0| \otimes (|0\rangle\langle 0| + |1\rangle\langle 1|) + |1\rangle\langle 1| \otimes (|0\rangle\langle 1| + |1\rangle\langle 0|) \tag{21}$$

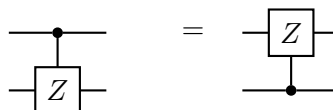
Exercise 18 *What is the state prepared by the following circuit?*



What outcomes would we get if we measured both qubits in the Z basis? What can you say about the correlation between them?

Exercise 19 *There are several identities that are useful when implementing quantum protocols. In this exercise, we will derive several of them.*

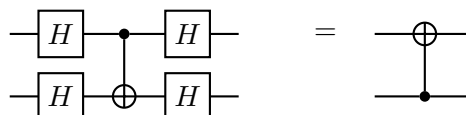
1. *We introduce controlled-Z in Table 4. Show that*



2. *Show that $HZH = X$ and $HXH = Z$.*

3. *How would one construct CZ out of CNOT and single-qubit gates?*

4. *Show that the following identity holds*



⁶ $\delta_{i,j}$ stands for Kronecker delta. $\delta_{i,j}$ is 1 if $i = j$ and 0 otherwise.

Further reading

The paper [2] established the equivalence between the quantum Turing machines and quantum circuit models. Specifically, Ref. [2] showed that quantum Turing machines can simulate, and be simulated by, uniform families of polynomial-size quantum circuits, with at most polynomial slowdown. Specific construction for building complicated circuits out of elementary gates can be found in [1].

References

- [1] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter, *Elementary gates for quantum computation*, Phys. Rev. A **52** (1995Nov), 3457–3467.
- [2] A. Chi-Chih Yao, *Quantum circuit complexity*, Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science, 1993, pp. 352–361.
- [3] Christopher M. Dawson and Michael A. Nielsen, *The Solovay-Kitaev algorithm*, 2005.