6 Feynman' Contribution

In this chapter we review Richard Feynman's two articles from 1982 and 1985 [9, 69, 70], which were seminal for the field of quantum computation. Both papers originated from invited talks at conferences. Feynman's interest had been triggered by the notion of reversible computation brought up by Fredkin, Bennett, and Toffoli. The sections of this chapter bear the same titles as the original papers. This chapter is not necessary in order to understand the remainder of this book. It is there purely for entertainment, or, if you are more seriously minded, for historical interest. A recent, very positive assessment of his concept was published in February 2025 [71].

6.1 Simulating physics with computers

6.1.1 Discrete system representations

In his 1982 article (which was mentioned already in Section 1.3.1) Feynman discussed the ways in which different kinds of physical systems can be simulated by computers. A deterministic simulation of a quantum system on a classical computer runs into problems because the required resources grow exponentially with the system size. In Section 1.3.1 we saw that even for a few spin-1/2 particles without any other degrees of freedom, the size of the Hilbert space is forbidding. This situation worsens considerably if additional (continuous) degrees of freedom of the particles must be accounted for. Classical (deterministic) dynamics, on the other hand, is much easier to simulate because it is *local, causal*, and *reversible*. Of course such a simulation always involves some kind of discretization for the possible values of continuous variables such as time, coordinates, field values, etc. For example, the motion of N interacting classical point particles in three dimensions is determined by $3N$ equations of motion. The number of differential equations is proportional to the number of particles. A typical numerical algorithm for solving these equations of motion involves a discretization of time and an approximation of differentials by differences. This converts the set of differential equations to a set of algebraic equations. The resources necessary to solve this set of algebraic equations grows as a power of the number of particles, but not exponentially. This means that classical deterministic dynamics can be efficiently simulated on a computer.

This is no longer so for classical *probabilistic* dynamics; at least if a deterministic simulation is desired. To understand what is meant by a deterministic simulation, consider the classical diffusion equation

$$\frac{\partial p}{\partial t} = D\nabla^2 p$$

(where D is the diffusion constant) as an example. $p(\vec{r}, t)$ is the probability density of finding a single particle which undergoes Brownian motion. To simulate the diffusion equation, space and time can be discretized and the dynamics can be approximated by a set of transition rules determining how probability "jumps" back and forth between neighboring points in space in each time step. The continuous function $p(\vec{r}, t)$ is thus replaced by an array of numbers p_{ik}, the probabilities of finding the diffusing particle at the space point $\vec{r}_i$ at the instant of time t_k. The simulation keeps track of all these numbers, starting from a given initial configuration p_{i0} and ending up with the desired final configuration p_{iT}, where i always runs from 1 to S, the number of grid points into which $\vec{r}$ was discretized.

The trouble starts as the number of diffusing particles increases. For two particles $p(\vec{r}, t)$ then becomes $p(\vec{r}_1, \vec{r}_2, t)$, where $\vec{r}_1$ and $\vec{r}_2$ are the coordinates of the two particles. This discretizes into an array of numbers p_{ijk}, where the indices i, j, and k indicate the possible discrete values of the coordinates $\vec{r}_1$ and $\vec{r}_2$ and the time t, respectively. The simulation now has to keep track of S^2 numbers per time step. With N particles one has S^N numbers per time step, which quickly outgrows the capabilities of any classical computer. Of course there are situations where a description in terms of individual particle coordinates is unnecessarily complicated, for example if the particles do not interact with each other, but if they do there is no way around this description (or a similar one).

6.1.2 Probabilistic simulations

Deterministic simulations of probabilistic dynamics keep track of all possible (discretized) configurations of the system, important ones as well as very improbable ones. The aim of a *probabilistic simulation* is to avoid the waste of resources implied by the complete calculation of all possible configurations. The probabilistic simulation is constructed in such a way that it arrives at any possible final result (or configuration) with the same probability as the natural process. This can be done without exponential growth of resources as the number of particles increases. Of course for probabilistic simulations repeated simulation runs (plus some statistics to generate error bars for the results) are necessary. Probabilistic simulations of this kind are everyday business for scientists and engineers in various fields.

A probabilistic simulation of a quantum system on a classical computer, however, turns out to be impossible. The fundamental reason for this failure is related to the nature of correlations in quantum systems. The possibility of a probabilistic simulation of quantum systems would imply the existence of some "hidden" classical variables which are not accessible to the observer and have to be averaged over to arrive at a physical result. The existence of such variables in turn restricts the values of correlations of the system, by the Bell or CHSH inequalities discussed in Section 4.4.6. These inequalities are not obeyed by quantum theory, and they have been shown to be violated in a number of quantum experiments. Thus a consistent probabilistic simulation of a quantum system on a classical computer is impossible, as demonstrated in detail by Feynman. This impossibility led Feynman to the suggestion of investigating the possibilities of quantum simulations performed by quantum computers, a field that we will briefly discuss in Section 8.6.

6.2 Quantum mechanical computers

6.2.1 Simple gates

Feynman's second paper contains quite detailed suggestions for quantum implementations of classical computing tasks. We will discuss these suggestions up to a "Hamiltonian that adds" before turning to the genuine quantum applications in the following chapters. The paper also shows that Feynman was well aware of (and interested in) the problems inherent in the high sensitivity of quantum systems to small perturbations; nevertheless, he says: "This study is one of principle; our aim is to exhibit some Hamiltonian for a system which could serve as a computer. We are not concerned with whether we have the most efficient system, nor how we could best implement it."

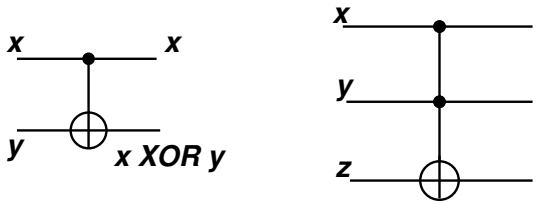

Figure 6.1: Left: Single CNOT gate. Right: CCNOT (Toffoli) gate.

From Chapter 3 we know some reversible gates on the 1-, 2-, and 3-bit levels. For single qubits:

$$\text{NOT} \quad \text{maps } x \longrightarrow 1 - x.$$

Figure 6.1 show controlled NOT gates for 2- and 3-qubit systems:

$$\begin{aligned} \text{CNOT} \quad \text{maps } (x, y) &\longrightarrow (x, x \text{ XOR } y) \\ &= \begin{cases} (x, y) \text{ if } x = 0 \\ (x, 1 - y) \text{ if } x = 1 \end{cases} , \end{aligned}$$

and the Toffoli gate, controlled controlled NOT or $\theta^{(3)}$ gate:

$$\begin{aligned} \text{CCNOT maps } (x, y, z) &\longrightarrow (x, y, xy \text{ XOR } z) \\ &= \begin{cases} (x, y, 1 - z) \text{ iff } x = y = 1 \\ (x, y, z) \text{ otherwise} \end{cases} , \end{aligned}$$

where "iff" is short for "if and only if", as usual, and the symbol $\oplus$ symbolizes XOR or equivalently addition modulo 2. Because for all three gates just one bit is flipped, all three are their own inverses, which will be important in what follows. Their matrix representations are unitary which allows us to consider them as quantum mechanical evolution operators that perform a logical gate operation.

6.2.2 Adder circuits

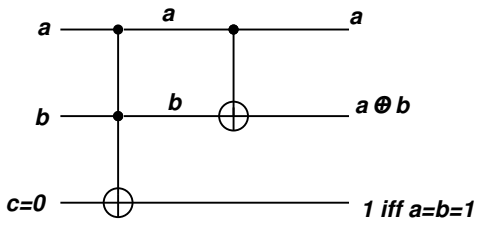

Figure 6.2: An adder (half-adder) circuit.

From these elements we can construct an adder (more precisely, a half-adder), which takes two input bits a and b and a carry bit c which is zero initially (Figure 6.2). The CCNOT changes the carry bit to 1 iff both a and b are 1. The output bit on the middle wire is 1 if $a = 1$ and $b = 0$ or

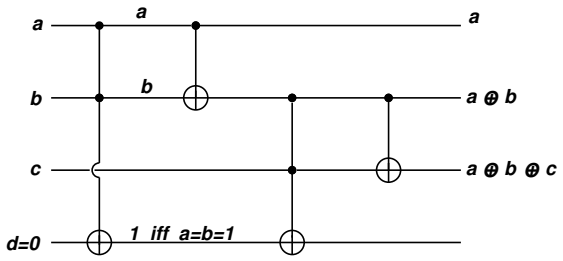

Figure 6.3: A full adder circuit.

if $a = 0$ and $b = 1$ and zero otherwise, and thus yields $a \oplus b$.

The next circuit (and the one for which we will construct a Hamiltonian) is a full adder (Figure 6.3). It takes two data bits a and b and a carry bit c from a previous calculation and calculates $a \oplus b \oplus c$, plus a carry bit d which is 1 if two or more of a, b, c are 1.

What is going on along the three top wires is quite clear, the "tricky bit" is the carry bit d, especially the action of the second CCNOT gate. If $a = b = 1$, $d = 1$ by the first CCNOT gate. The control bit $a \oplus b = 0$ of the second CCNOT gate then is zero so that d is not flipped back to 0 regardless of the value of c. The only case in which d is flipped (from 0 to 1) is $a \oplus b = 1$ and $c = 1$, such that indeed $d = 1$ if $a + b + c \geq 2$.

6.2.3 Qubit raising and lowering operators

We now change our point of view from classical to quantum. To this end we first map the bits to qubits of which we only use the basis states $|0\rangle = |\uparrow\rangle$ and $|1\rangle = |\downarrow\rangle$, since we are (at this point) not interested in the specific quantum properties arising from the superposition principle. We have to translate the gates and circuits discussed above into quantum mechanical operators. From Chapter 4 we know how to flip a qubit by the spin raising and lowering operators:

$$\mathbf{S}_a^+ |\downarrow\rangle_a = \hbar |\uparrow\rangle_a \quad ; \quad \mathbf{S}_a^- |\uparrow\rangle_a = \hbar |\downarrow\rangle_a. \quad (6.1)$$

The index a simply reminds us that we are manipulating the qubit a.

For the following discussion it is convenient to use a slightly different notation and language. We interpret the basis states $|0\rangle_a$ and $|1\rangle_a$ as absence and presence of a particle at position a, respectively. The number of particles at qubit a can be either zero or one, and this number can be changed by creating or annihilating an "a-type particle". These tasks are performed by the creation operator $\mathbf{a}^\dagger$ and by its adjoint, the annihilation operator $\mathbf{a}$:

$$\mathbf{a}^\dagger|0\rangle_a = |1\rangle_a \quad ; \quad \mathbf{a}|1\rangle_a = |0\rangle_a. \tag{6.2}$$

Comparing (6.2) to (6.1) we see that $\mathbf{a}^\dagger$ corresponds to $\mathbf{S}_a^-$ and $\mathbf{a}$ corresponds to $\mathbf{S}_a^+$. We stress that we will only use the language of creation and annihilation operators as a convenient way of discussing the states of qubits; we will not employ the full formal machinery of the "occupation number representation", also known as "second quantization" .

6.2.4 NOT and CNOT

Recalling the relation (4.13)

$$\mathbf{S}_x = \frac{\hbar}{2}\mathbf{X} = \frac{1}{2}(\mathbf{S}^+ + \mathbf{S}^-),$$

we can express the NOT operation on qubit a in terms of the a-particle creation and annihilation operators:

$$\text{NOT } (a) = (\mathbf{a} + \mathbf{a}^\dagger).$$

Since the qubit a may be used as a control qubit in a CNOT or CCNOT gate, we need a convenient way of checking the state of a without changing it. In our newly adopted language this means "counting the number of a-particles", and it is achieved by the particle number operator $\mathbf{a}^\dagger\mathbf{a}$, as can be verified:

$$\mathbf{a}^\dagger\mathbf{a}|x\rangle_a = x|x\rangle_a \quad (x = 0, 1).$$

In order to take care of other qubits b, c, etc., in addition to the qubit a, we introduce b-type, c-type, etc. particles with corresponding creation and annihilation operators $\mathbf{b}^\dagger$ and $\mathbf{b}$, $\mathbf{c}^\dagger$ and $\mathbf{c}$, etc. This can be used to write down the operator corresponding to the CNOT gate with a as control qubit. This operator is supposed to flip b if $a = 1$ and to do nothing if $a = 0$:

$$\begin{aligned} \text{CNOT } (a, b) &= (\mathbf{b} + \mathbf{b}^\dagger)\mathbf{a}^\dagger\mathbf{a} + \mathbf{1}_b(\mathbf{1}_a - \mathbf{a}^\dagger\mathbf{a}) \\ &= (\mathbf{b} + \mathbf{b}^\dagger - \mathbf{1}_b)\mathbf{a}^\dagger\mathbf{a} + \mathbf{1}_b\mathbf{1}_a. \end{aligned}$$

The operators acting on different sites belong to different Hilbert spaces and therefore commute with each other. This is a property reminiscent of Bose particles (Bosons), while the "on-site" (anti-)commutation relation

$$\mathbf{a}^\dagger\mathbf{a} + \mathbf{a}\mathbf{a}^\dagger = \mathbf{1}_a$$

is typical for Fermi particles (Fermions). Thus the particles employed here are neither Bosons nor Fermions, which would cause some complications if we intended to use standard many-particle calculational techniques. As mentioned already, however, we are not going to do this.

To continue the construction of a "Hamiltonian that adds" we need to code the Toffoli gate $\theta^{(3)}$ or CCNOT as an operator, similar to CNOT:

$$\boldsymbol{\theta}^{(3)}(a, b, c) = \mathbf{1}_a\mathbf{1}_b\mathbf{1}_c + (\mathbf{c}^\dagger + \mathbf{c} - \mathbf{1}_c)\mathbf{a}^\dagger\mathbf{a}\mathbf{b}^\dagger\mathbf{b}.$$

The operator for the full adder can be written down reading the diagram in Fig. 6.3 starting from the left and writing down the elementary operators starting from the right:

$$\begin{aligned} &\text{CNOT } (b, c)\boldsymbol{\theta}^{(3)}(b, c, d) \text{ CNOT } (a, b) \\ &\qquad \boldsymbol{\theta}^{(3)}(a, b, d)|a, b, c, 0\rangle \\ &\qquad =: \mathbf{A}_4\mathbf{A}_3\mathbf{A}_2\mathbf{A}_1|a, b, c, 0\rangle \\ &\qquad = \exp\left(-i\frac{\mathcal{H}t}{\hbar}\right)|a, b, c, 0\rangle \end{aligned}$$

(with obvious definitions of the operators $\mathbf{A}_1 \ldots \mathbf{A}_4$). Is there a Hamiltonian $\mathcal{H}$ and a time t which both satisfy this equation? Obviously this is no easy question, since

$$\begin{aligned} \exp\left(-i\frac{\mathcal{H}t}{\hbar}\right) &= \mathbf{1} + \left(-i\frac{\mathcal{H}t}{\hbar}\right) + \frac{1}{2}\left(-i\frac{\mathcal{H}t}{\hbar}\right)^2 \\ &\quad + \frac{1}{6}\left(-i\frac{\mathcal{H}t}{\hbar}\right)^3 + \cdots \end{aligned}$$

and thus the right-hand side of the above equation for the full adder will be a superposition of states where $\mathcal{H}$ has acted any number of times, from zero to infinity. Nevertheless, it turns out that it is possible:

- to construct an $\mathcal{H}$ such that the desired final state is present (among others) and
- to separate the desired state from the others.

The trick is to keep a record of which of the $\mathbf{A}$ operators have already acted on the input state. This bookkeeping is done by auxiliary (or "slave") particles. Suppose we want to calculate

$$|\psi_f\rangle = \mathbf{A}_k\mathbf{A}_{k-1}\cdots\mathbf{A}_1|\psi_i\rangle$$

(in our example $k = 4$) for an n-qubit state $|\psi_i\rangle$ ($n = 4$ in our example). We introduce a "chain" of $k+1$ new "program counter qubits" named $i = 0\cdots k$, with corresponding creation and annihilation operators $\mathbf{q}_i$,$\mathbf{q}_i^\dagger$.

6.2.5 Adder Hamiltonian

The desired Hamiltonian then reads

$$\begin{aligned}\mathcal{H} &= \sum_{i=0}^{k-1}(\mathbf{q}_{i+1}^\dagger\mathbf{q}_i\mathbf{A}_{i+1} + \text{ h.c.})\\ &= \sum_{i=0}^{k-1}(\mathbf{q}_{i+1}^\dagger\mathbf{q}_i\mathbf{A}_{i+1} + A_{i+1}^\dagger\mathbf{q}_i^\dagger\mathbf{q}_{i+1})\\ &= \sum_{i=0}^{k-1}(\mathbf{q}_{i+1}^\dagger\mathbf{q}_i + \mathbf{q}_i^\dagger\mathbf{q}_{i+1})\mathbf{A}_{i+1}.\end{aligned}$$

Here, "h.c." denotes the Hermitian conjugate (to make $\mathcal{H}$ Hermitian). We have used the fact that the $\mathbf{A}$ operators are Hermitian and the $\mathbf{q}$ operators are assumed to commute among themselves and with all gate operators $\mathbf{A}_i$. Note that the number of "q particles" $\sum_{i=0}^k \mathbf{q}_i^\dagger\mathbf{q}_i$ is a constant; we will be interested exclusively in the case of a single particle. The action of the Hamiltonian is represented pictorially in Figure 6.4:

Whenever the "program counter particle" moves from site i to $i+1$ or vice versa the operator $\mathbf{A}_{i+1}$

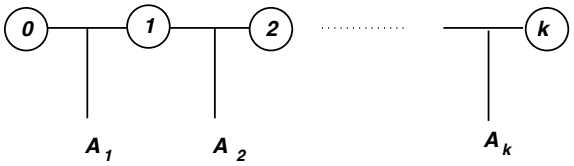

Figure 6.4: A Hamiltonian with register operations $\mathbf{A}_i$.

acts on the "register qubits" where the calculation is performed. The calculation starts with the register qubits in the input state $|\psi_i\rangle$ and a single program counter particle at site 0. The action of $\mathcal{H}^\nu$ then yields

$$\begin{aligned}\mathcal{H}^\nu|1000\cdots0\rangle|\psi_i\rangle &= \mathcal{H}^{\nu-1}|0100\cdots0\rangle\mathbf{A}_1|\psi_i\rangle\\ &= \mathcal{H}^{\nu-2}(|0010\cdots0\rangle\mathbf{A}_2\mathbf{A}_1|\psi_i\rangle\\ &\quad +|1000\cdots0\rangle\underbrace{\mathbf{A}_1\mathbf{A}_1}_{\mathbf{1}}|\psi_i\rangle)\end{aligned}$$

where we have used that the gates $\mathbf{A}_i$ are their own inverses. We see that if the program counter particle is at site l, the last operator which has been active is $\mathbf{A}_l$:

$$|0\cdots\underbrace{1}_{l}\cdots0\rangle\mathbf{A}_l\cdots|\psi_i\rangle.$$

The next application of $\mathcal{H}$ then leads to two possibilities:

- $l \to l-1; \mathbf{A}_l$ is squared (and thus erased because it is its own inverse)
- $l \to l+1; \mathbf{A}_{l+1}$ is prepended to the string of $\mathbf{A}$ operators.

(This argument can of course be transformed into a rigorous proof by induction.) We conclude that if our final state contains a component with the counter particle at site k, we are finished. We only have to project out the desired component:

$$\begin{aligned}&\alpha|00\cdots01\rangle|\psi_f\rangle\\ &= \mathbf{q}_k^\dagger\mathbf{q}_k\exp\left(-i\frac{\mathcal{H}t}{\hbar}\right)|100\cdots0\rangle|\psi_i\rangle,\end{aligned}$$

where α is a normalization factor whose size may be important in practice. The operator $\mathbf{q}_k^\dagger\mathbf{q}_k$,

which detects the presence of a particle at the last site, indicates that the computation has terminated.

After showing how to construct the full adder Hamiltonian, Feynman in his paper then goes on to discuss the influence of imperfections (for example not perfectly equal "bond strengths" in the program counter qubit chain), simplifications of the implementation and more complicated tasks like implementing loops which perform a piece of code a given number of times. We recommend the original paper [70] to readers who want to discover more details.

Problem

Construct the Hamiltonian for the full adder. Calculate (for example numerically, with your classical PC) the amplitude of the desired output state as a function of time. Does this amplitude depend on the contents of the register qubits? Can you see how it will depend on the number of program steps k for more general programs?

This problem is more advanced than others in this book. We have not solved it ourselves, but we are confident that it is feasible and that it will basically reduce to finding the eigenvalues and eigenstates of a single quantum mechanical particle moving on an open-ended chain of five sites, which is a typical (and solvable) exercise in many courses on condensed matter theory.