F Universal quantum computers

Hitherto we have used a practical definition of universality: since conventional digital computers are implemented in terms of binary digital logic, we have taken the ability to implement binary digital logic as sufficient for universality. This leave open the question of the relation of quantum computers to the theoretical standard of computational universality: the Turing machine. Therefore, a natural question is: What is the power of a quantum computer? Is it super-Turing or sub-Turing? Another question is: What is its efficiency? Can it solve NP problems efficiently? There are a number of universal quantum computing models for both theoretical and practical purposes.

F.1 Feynman on quantum computation

F.1.a Simulating quantum systems

In 1982 Richard Feynman discussed what would be required to simulate a quantum mechanical system on a digital computer.[22] First he considered a classical probabilistic physical system. Suppose we want to use a conventional computer to calculate the probabilities as the system evolves in time. Further suppose that the system comprises R particles that are confined to N locations in space, and that each configuration c has a probability $p(c)$. There are N^R possible configurations, since a configuration assigns a location N to each of the R particles (i.e., the number of functions $R \to N$). Therefore to simulate all the possibilities would require keeping track of a number of quantities (the probabilities) that grows exponentially with the size of the system. This is infeasible.

So let's take a weaker goal: we want a simulator that exhibits the same probabilistic behavior as the system. Our goal is that if we run both of them over and over, we will see the same distribution of behaviors. This we can do. You can implement this by having a nondeterministic computer that has the same state transition probabilities as the primary system.

Let's try the same trick with quantum systems, that is, have a conventional computer that exhibits the same probabilities as the quantum system. If you do the math (which we won't), it turns out that this is impossible. The reason is that, in effect, some of the state transitions would have to have

[22] This section is based primarily on Feynman (1982).

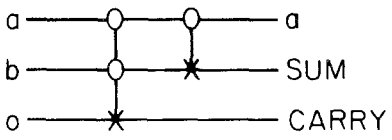

Figure III.40: Simple adder using reversible logic. [fig. from Feynman (1986)]

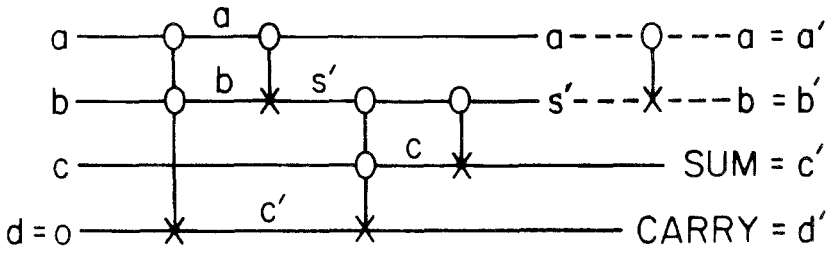

Figure III.41: Full adder using reversible logic. [fig. from Feynman (1986)]

what amount to negative probabilities, and we don't know how to do this classically. We've seen how in quantum mechanics, probabilities can in effect cancel by destructive interference of the wavefunctions. The conclusion is that no conventional computer can efficiently simulate a quantum computer. Therefore, if we want to (efficiently) simulate any physical system, we need a quantum computer.

F.1.b Universal quantum computer

In 1985 Feynman described several possible designs for a universal quantum computer.[23] He observes that NOT, CNOT, and CCNOT are sufficient for any logic gate, as well as for COPY and EXCHANGE, and therefore for universal computation. He exhibits circuits for a simple adder (Fig. III.40) and a full adder (Fig. III.41).

The goal is to construct a Hamiltonian to govern the operation of a quantum computer. Feynman describes quantum logic gates in terms of two primitive operations, which change the state of an "atom" (two-state system or "wire"). Letters near the beginning of the alphabet $(a, b, c, \ldots)$ are used for

[23] This section is based primarily on Feynman (1986).

data or *register atoms*, and those toward the end $(p, q, r, \ldots)$ for *program atoms* (which are used for sequencing operations). In this simple sequential computer, only one program atom is set at a time.

For a single line a, the *annihilation operator* is defined:

$$a = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = |0\rangle\langle 1|.$$

The *annihilator* changes the state $|1\rangle$ to $|0\rangle$; typically it lowers the energy of a quantum system. Applied to $|0\rangle$, it leaves the state unchanged and returns the *zero vector* $\mathbf{0}$ (which is not a meaningful quantum state). It *matches* $|1\rangle$ and *resets* it to $|0\rangle$. The operation is not unitary (because not norm preserving). It is a "partial NOT" operation.

Its conjugate transpose it the *creation operation*:[24]

$$a^* = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = |1\rangle\langle 0|.$$

The *creator* transforms $|0\rangle$ to $|1\rangle$, but leaves $|1\rangle$ alone, returning $\mathbf{0}$; typically it raises the energy of a quantum system. It matches $|0\rangle$ and sets it to $|1\rangle$. This is the other half of NOT $= a + a^*$.

Feynman also defines a *number operation* or *1-test*: Consider[25]

$$N_a = a^* a = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = |1\rangle\langle 1|.$$

This has the effect of returning $|1\rangle$ for input $|1\rangle$, but $\mathbf{0}$ for $|0\rangle$:

$$N_a = a^* a = |1\rangle\langle 0|\ |0\rangle\langle 1| = |1\rangle\langle 0 \mid 0\rangle\langle 1| = |1\rangle\langle 1|.$$

Thus it's a test for $|1\rangle$. (This is a partial identity operation.)

Similarly, the *0-test* is defined:[26]

$$\mathbf{1} - N_a = aa^* = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = |0\rangle\langle 0|.$$

[24]Note that Feynman uses a^* for the adjoint (conjugate transpose) of a.

[25]This matrix is not the same as that given in Feynman (1982, 1986), since Feynman uses the basis $|1\rangle = (1,0)^{\mathrm{T}}, |0\rangle = (0,1)^{\mathrm{T}}$, whereas we use $|0\rangle = (1,0)^{\mathrm{T}}$ and $|1\rangle = (0,1)^{\mathrm{T}}$.

[26]This matrix is different from that given in Feynman (1982, 1986), as explained in the previous footnote.

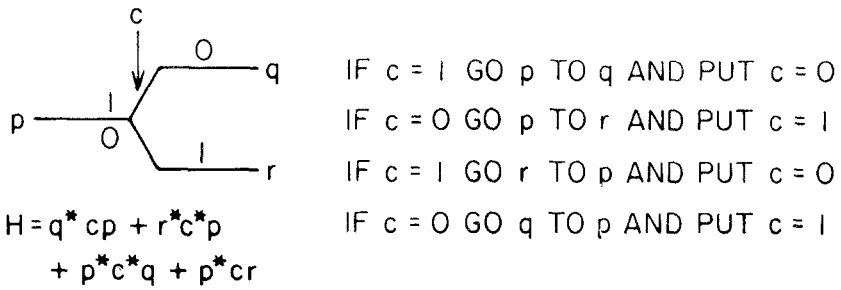

Figure III.42: Switch element. 0/1 annotations on the wires show the c values. [fig. from Feynman (1986)]

(Feynman writes this $\mathbf{1} - N_a$ because he writes $\mathbf{1} = I$.) This has the effect of returning $|0\rangle$ for input $|0\rangle$, but $\mathbf{0}$ for $|1\rangle$. This is test for $|0\rangle$. (It is the rest of the identity operation.)

The two operations a and a^* are sufficient for creating all 2×2 matrices, and therefore all transformations on a single qubit. Note that

$$\begin{pmatrix} w & x \\ y & z \end{pmatrix} = waa^* + xa + ya^* + za^*a.$$

Feynman writes A_a for the negation (NOT) operation applied to a. Obviously, $A_a = a + a^*$ (it annihilates $|1\rangle$ and creates from $|0\rangle$) and $\mathbf{1} = aa^* + a^*a$ (it passes $|0\rangle$ and passes $|1\rangle$. We can prove that $A_aA_a = \mathbf{1}$ (Exer. III.55).

Feynman writes $A_{a,b}$ for the CNOT operation applied to lines a and b. Observe, $A_{a,b} = a^*a(b + b^*) + aa^*$. Notice that this is a tensor product on the register $|a, b\rangle$: $A_{a,b} = a^*a \otimes (b + b^*) + aa^* \otimes \mathbf{1}$. You can write this formula $N_a \otimes A_b + (\mathbf{1} - N_a) \otimes \mathbf{1}$. That is, if N_a detects $|1\rangle$, then it negates b. If $\mathbf{1} - N_a$ detects $|0\rangle$, then it leaves b alone.

Feynman writes $A_{ab,c}$ for the CCNOT operation applied to lines a, b, and c. Note that $A_{ab,c} = \mathbf{1} + a^*ab^*b(c + c^* - \mathbf{1})$ (Exer. III.56). This formula is more comprehensible in this form:

$$A_{ab,c} = \mathbf{1} + N_aN_b(A_c - \mathbf{1}).$$

One of Feynman's universal computers is based on only two logic gates, NOT and SWITCH (Fig. III.42). If $c = |1\rangle$, then the "cursor" (locus of control)

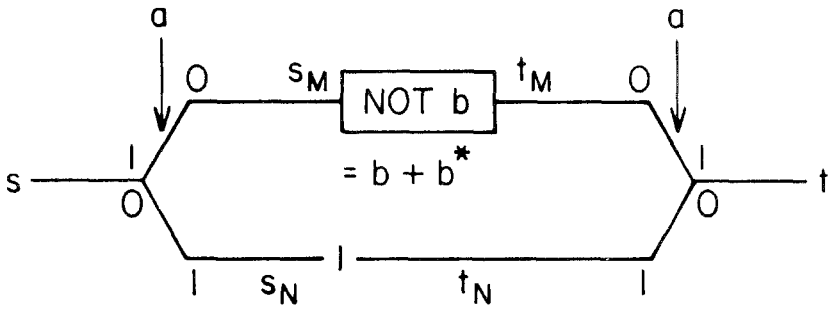

Figure III.43: CNOT implemented by switches. 0/1 annotations on the wires show the a values. [fig. from Feynman (1986)]

---

at p moves to q, but if $c = |0\rangle$ it moves to r. It also negates c in the process. It's also reversible (see Fig. III.42).

The switch is a tensor product on $|c, p, q, r\rangle$:

$$q^* cp + r^* c^* p + [p^* c^* q + p^* cr].$$

(The bracketed expression is just the complex conjugate of the first part, required for reversibility.) Read the factors in each term from right to left:
(1) $q^* cp$: if p and c are set, then unset them and set q.
(2) $r^* c^* p$: if p is set and c is not set, the unset p and set c and r.

Fig. III.43 shows CNOT implemented by switches. This is the controlled-NOT applied to data a, b and sequenced by cursor atoms s, t (= start, terminate). If $a = 1$ the cursor state moves along the top line, and if $a = 0$ along the bottom. If it moves along the top, then it applies $b + b^*$ to negate b (otherwise leaving it alone). In either case, the cursor arrives at the reversed switch, where sets the next cursor atom t. We can write it

$$H_{a,b}(s,t) = s_M^* as + t^* a^* t_M + t_M^*(b + b^*)s_M + s_N^* a^* s + t^* a t_N + t_N^* s_N + \text{c.c},$$

where "c.c" means to add the complex conjugates of the preceding terms. Read the factors in each term from right to left:
(1) $s_M^* as$: if s and a are set, then unset them and set s_M.
(4) $s_N^* a^* s$: if s is set and a in unset, then unset s and set s_N and a.
(6) $t_N^* s_N$: if s_N is set, then unset it and set t_N.
(3) $t_M^*(b + b^*)s_M$: if s_M is set, then unset it, negate b and set t_M.

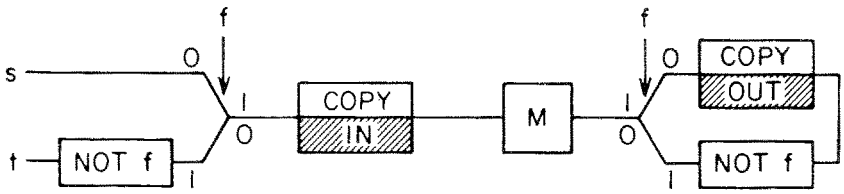

Figure III.44: Garbage clearer. 0/1 annotations on the wires show the f values. [fig. from Feynman (1986)]

---

(5) $t^* a t_N$: if t_N and a are set (as a must be to get here), then unset them and set t.
(2) $t^* a^* t_M$: if t_M is set and a is unset (as it must be to get here), then reverse their states and set t.
(The $t_N^* s_N$ term can be eliminated by setting $t_N = s_N$.)

F.1.c Garbage clearer

Instead of having a separate copy of the machine to clear out the garbage, it's possible to run the same machine backwards (Fig. III.44). An external register In contains the input, and the output register Out and all machine registers are initialized to 0s. Let s be the starting program atom. The flag f is initially 0.

The $f = 0$ flag routes control through the reversed switch (setting $f = 1$) to Copy. The Copy box uses CNOTs to copy the external input into M. Next M operates, generating the result in an internal register. At the end of the process M contains garbage.

The $f = 1$ flag directs control into the upper branch (resetting $f = 0$), which uses CNOTs to copy the result into the external output register Out. Control then passes out from the upper branch of the switch down and back into the lower branch, which negates f, setting it to $f = 1$. Control passes back into the machine through the lower switch branch (resetting $f = 0$), and backwards through M, clearing out all the garbage, restoring all the registers to 0s. It passes backwards through the Copy box, copying the input back from M to the external input register In. This restores the internal registers to 0s. Control finally passes out through the lower branch of the left switch (setting $f = 1$), but it negates f again, so $f = 0$. It arrives at the terminal

program atom t. At the end of the process, everything is reset to the initial conditions, except that we have the result in the OUT register. Feynman also discusses how to do subroutines and other useful programming constructs, but we won't go into those details.

F.2 Benioff's quantum Turing machine

In 1980 Paul Benioff published the first design for a universal quantum computer, which was based on the Turing machine (Benioff, 1980). The tape is represented by a finite lattice of quantum spin systems with eigenstates corresponding to the tape symbols. (Therefore, he cannot implement an open-ended TM tape, but neither can an ordinary digital computer.) The read/write head is a spinless system that moves along the lattice. The state of the TM was represented by another spin system. Benioff defined unitary operators for doing the various operations (e.g., changing the tape). In 1982 he extended his model to erase the tape, as in Bennett's model (Benioff, 1982). Each step was performed by measuring both the tape state under the head and the internal state (thus collapsing them) and using this to control the unitary operator applied to the tape and state. As a consequence, the computer does not make much use of superposition.

F.3 Deutsch's universal quantum computer

Benioff's computer is effectively classical; it can be simulated by a classical Turing machine. Moreover, Feynman's construction is not a true universal computer, since you need to construct it for each computation, and it's not obvious how to get the required dynamical behavior. Deutsch sought a broader definition of quantum computation, and a universal quantum computer $\mathcal{Q}$.[27] M binary observables are used to represent the processor, $\{\check{n}_i\}$ for $i \in \mathbf{M}$, where $\mathbf{M} = \{0, \ldots, M-1\}$. Collectively they are called $\check{\mathbf{n}}$. An infinite sequence of binary observables implements the memory, $\{\check{m}_i\}$ $(i \in \mathbb{Z})$ Collectively the sequence is called $\check{\mathbf{m}}$. An observable $\check{x}$, with spectrum $\mathbb{Z}$, represents the tape position (address) of the head. The *computational basis states* have the form:

$$|x; \mathbf{n}; \mathbf{m}\rangle \stackrel{\text{def}}{=} |x; n_0, n_1, \ldots, n_{M-1}; \ldots, m_{-1}, m_0, m_1, \ldots\rangle.$$

[27] This section is based on Deutsch (1985).

Here the eigenvectors are labeled by their eigenvalues $x, \mathbf{n}$, and $\mathbf{m}$.

The dynamics of computation is described by a unitary operator U, which advances the computation by one step of duration T:

$$|\psi(nT)\rangle = U^n|\psi(0)\rangle.$$

Initially, only a finite number of the memory elements are prepared in a non-zero state.

$$|\psi(0)\rangle = \sum_m \lambda_m|0;\mathbf{0};\mathbf{m}\rangle, \text{ where } \sum_m |\lambda_m|^2 = 1,$$

That is, only finitely many $\lambda_m \neq 0$, and in particular $\lambda_m = 0$ when an infinite number of the $\mathbf{m}$ are non-zero. The non-zero entries are the program and its input. Note that the initial state may be a superposition of initial tapes.

The matrix elements of U (relating the new state to the current state) have the form:

$$\begin{aligned}&\langle x';\mathbf{n}';\mathbf{m}' \mid U \mid x;\mathbf{n};\mathbf{m}\rangle \\ &= [\delta_{x'}^{x+1}U^+(\mathbf{n}',m'_x|\mathbf{n},m_x) + \delta_{x'}^{x-1}U^-(\mathbf{n}',m'_x|\mathbf{n},m_x)]\prod_{y\neq x}\delta_{m'_y}^{m_y}.\end{aligned}$$

U^+ and U^- represent moves to the right and left, respectively. The first two δ functions ensure that the tape position cannot move by more than one position in a step. The final product of deltas ensures that all the other tape positions are unchanged; it's equivalent to: $\forall y \neq x : m'_y = m_y$. The U^+ and U^- functions define the actual transitions of the machine in terms of the processor state and the symbol under the tape head. Each choice defines a quantum computer $\mathcal{Q}[U^+, U^-]$.

The machine cannot be observed before it has halted, since that would generally alter its state. Therefore one of the processor's bits is chosen as a halt indicator. It can be observed from time to time without affecting the computation.

$\mathcal{Q}$ can simulate TMs, but also any other quantum computer to arbitrary precision. In fact, it can simulate any finitely realizable physical system to arbitrary precision, and it can simulate some physical systems that go beyond the power of TMs (hypercomputation).