Classical computations on quantum computers

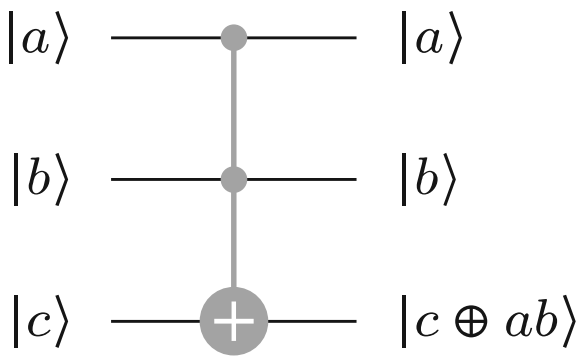
We'll now turn our attention to implementing classical algorithms on quantum computers. We'll see that any computation that can be performed with a classical Boolean circuit can also be performed by a quantum circuit with a similar asymptotic computational cost. Moreover, this can be done in a "clean" manner to be described shortly, which is an important requirement for using these computations as subroutines inside of larger quantum computations.

Simulating Boolean circuits with quantum circuits

Boolean circuits are composed of AND, OR, NOT, and FANOUT gates. To simulate Boolean circuits with quantum circuits, we'll begin by showing how each of these four gates can be simulated by quantum gates. Once that's done, converting a given Boolean circuit to a quantum circuit is a simple matter of simulating one gate at a time. We'll only need NOT gates, controlled-NOT gates, and Toffoli gates to do this, which are all deterministic operations in addition to being unitary.

Toffoli gates

Toffoli gates can alternatively be described as controlled-controlled-NOT gates, whose action on standard basis states is as shown in the following figure.

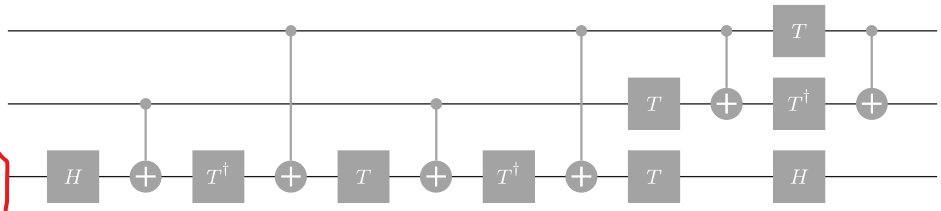


Bearing in mind that we're using Qiskit's ordering convention, where the qubits are ordered in increasing significance from top to bottom, the matrix representation of this gate is as follows.

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

Another way to think about Toffoli gates is that they're essentially query gates for the AND function, in the sense that they follow the pattern we saw in the previous lesson for unitary query gate implementations of arbitrary functions having binary string inputs and outputs.

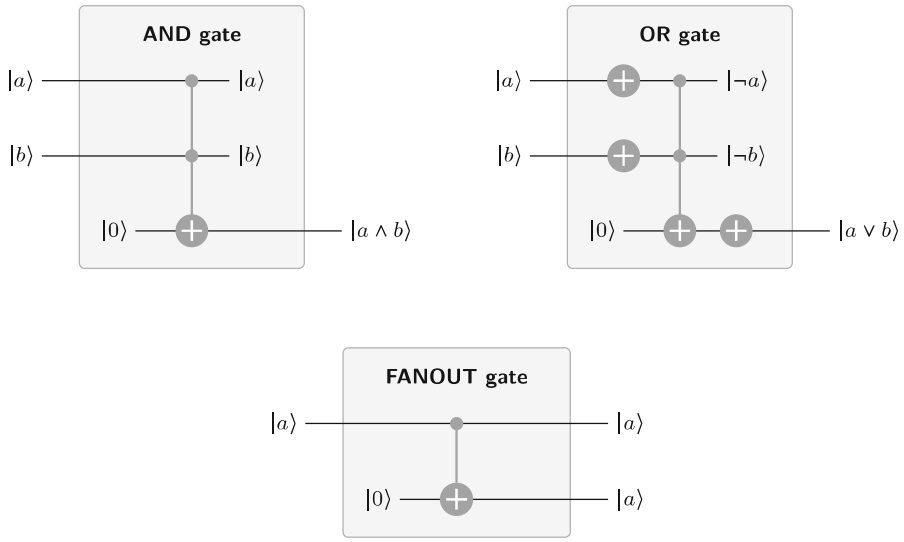
Toffoli gates are not included in the default gate set discussed earlier in the lesson, but it is possible to construct a Toffoli gate from $H, T, T^\dagger$, and CNOT gates as follows.



Simulating Boolean gates with Toffoli, controlled-NOT, and NOT gates

A single Toffoli gate, used in conjunction with a few NOT gates, can implement an AND and OR gate, and FANOUT gates can easily be

implemented using controlled-NOT gates, as the following diagrams suggest.



In all three cases, the qubits that the AND, OR, and FANOUT gates act upon come in from the left as inputs, and we also require one *workspace* qubit initialized to the zero state for each one. These workspace qubits appear inside of the boxes representing the gate implementations to suggest that they're new, and therefore part of the cost of these implementations.

For the AND and OR gates we also have two qubits left over, in addition to the output qubit. For example, inside the box in the diagram representing the simulation of an AND gate, the top two qubits are left in the states $|a\rangle$ and $|b\rangle$. These qubits are illustrated as remaining inside of the boxes because they're no longer needed and are not part of the output. They can be ignored for now, though we will turn our attention back to them shortly.

The remaining Boolean gate, the NOT gate, is included in our default set of quantum gates, so we don't require a simulation for this one.

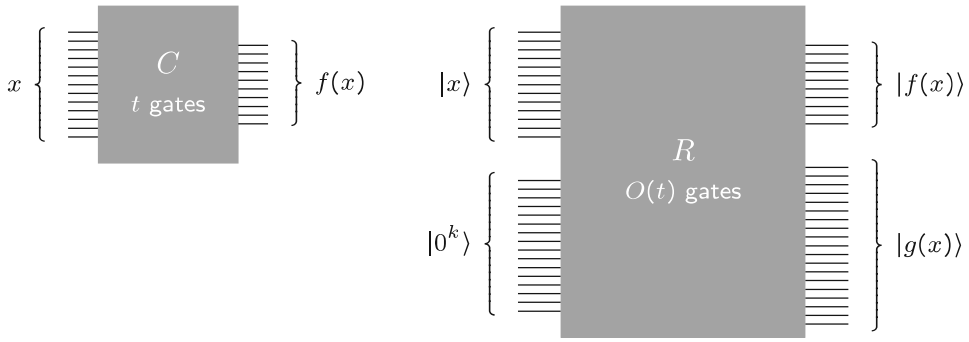
Gate by gate simulation of Boolean circuits

Now suppose that we have an ordinary Boolean circuit named C, composed of AND, OR, NOT, and FANOUT gates, and having n input bits and m of output bits. Let $t = \mathrm{size}(C)$ be the number of gates in C, and let's give the name f to the function that C computes, which takes the form

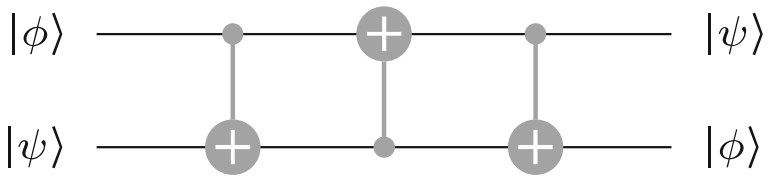
$$f : \Sigma^n \to \Sigma^m$$

for $\Sigma = \{0, 1\}$.

Now consider what happens when we go one at a time through the AND, OR, and FANOUT gates of C, replacing each one by the corresponding simulation described above, including the addition of the required workspace qubits. Let's name the resulting circuit R, and let's order the qubits of R so that the n input bits of C correspond to the top n qubits of R and the workspace qubits are on the bottom.



Here, k is the number of workspace qubits required — one for each AND, OR, and FANOUT gate of C — and g is a function of the form $g : \Sigma^n \to \Sigma^{n+k-m}$ that describes the states of the leftover qubits created by the gate simulations after R is run. In the figure, the qubits corresponding to the output $f(x)$ are on the top and the remaining, leftover qubits storing $g(x)$ are on the bottom. We can force this to happen if we wish by rearranging the qubits using SWAP gates, which can be implemented with three controlled-NOT gates like this:



As we'll see in the next section, it's not really essential to rearrange the output qubits like this, but it's easy enough to do it if we choose.

The function g that describes the classical states of the leftover qubits is determined by the circuit C, but we actually don't need to worry all that much about it; we don't care specifically what state these qubits are in when the computation finishes. The letter g comes after f, so it's a reasonable name for this function on that account, but there's a better reason to pick the name g — it's short for *garbage*.

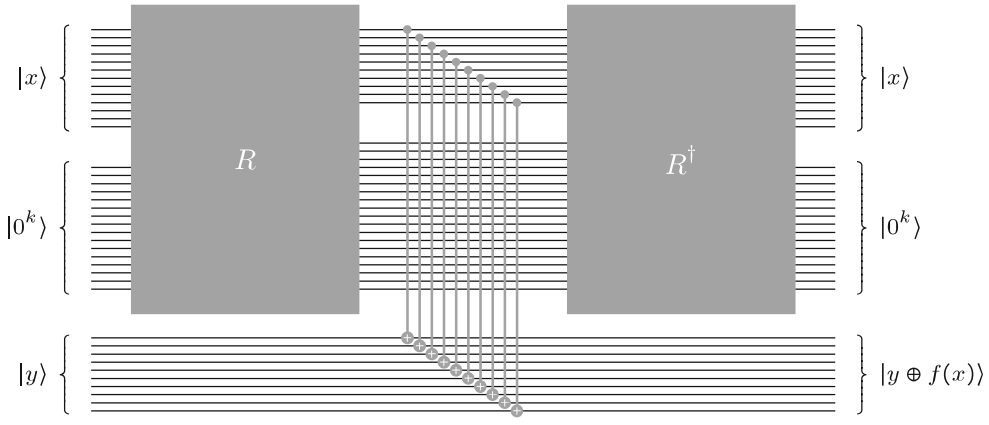
Cleaning up the garbage

If our only interest is in evaluating the function f computed by a given Boolean circuit C with a quantum circuit, we don't need to proceed any further than the gate-by-gate simulation just described. This means that, in addition to the output of the function, we'll have a bunch of garbage left over.

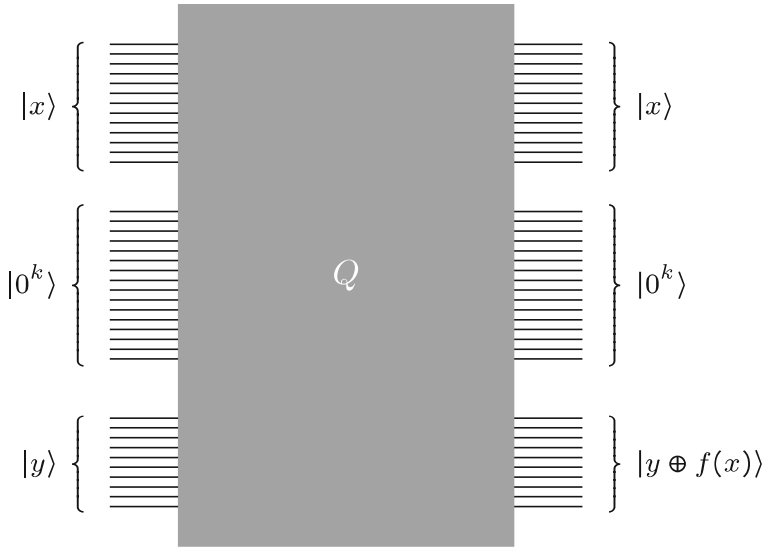
However, this is not good enough if we want to perform classical computations as subroutines within larger quantum computations, because those garbage qubits will cause problems. The phenomenon of *interference* is critically important to quantum algorithms, and garbage qubits can ruin the interference patterns needed to make quantum algorithms work.

Fortunately, it's not difficult to clean up the garbage, so to speak. The key is to use the fact that because R is a quantum circuit, we can run it in reverse, by simply replacing each gate with its inverse and applying them in the reverse order, thereby obtaining a quantum circuit for the operation $R^\dagger$. Toffoli gates, CNOT gates, and NOT gates are actually their own inverses, so running R in reverse is really just a matter of applying the gates in the reverse order — but more generally any quantum circuit can be reversed as just described.

Specifically, what we can do is to add m more qubits (recalling that the function f has m output bits), use CNOT gates to copy the output of R onto these qubits, and reverse R to clean up the garbage. The following figure illustrates the resulting circuit and describes its action on standard basis states.



If we put a box around the entire circuit and call it Q, it looks like this:



Given that C has t gates, the circuit Q will have $O(t)$ gates.

If we disregard the k additional workspace qubits, what we have is a circuit Q that functions exactly like a query gate for the function f. If we simply want to compute the function f on some string x, we can set $y = 0^m$ and the resulting value $f(x)$ will appear on the bottom m qubits — or we can feed in a different state to the bottom m qubits if we wish (perhaps to make use of a phase kickback, like in Deutsch's or the Deutsch-Jozsa algorithm).

This means that for any query algorithm, if we have a Boolean circuit that computes the input function, we can replace each query gate with a circuit implementation of it, and the query algorithm will function correctly.

Note that the workspace qubits are needed to make this process work, but they are returned to their initial states once the combined circuit is executed. This allows these qubits to be used again as workspace qubits for other purposes. There are also known strategies to reduce the number of workspace qubits required (which come at a cost of making the circuits larger), but we won't discuss those strategies here.

Implementing invertible functions

The construction just described allows us to simulate any Boolean circuit with a quantum circuit in a garbage-free manner. If C is a Boolean circuit implementing a function $f : \Sigma^n \to \Sigma^m$, then we obtain a quantum circuit Q that operates as follows on standard basis states.

$$Q\big(|y\rangle|0^k\rangle|x\rangle\big) = |y \oplus f(x)\rangle|0^k\rangle|x\rangle$$

The number k indicates how many workspace qubits are required in total. This is enough for the purposes of this course, but it is possible to take this methodology one step further when the function f itself is invertible.

To be precise, suppose that the function f takes the form $f : \Sigma^n \to \Sigma^n$, and also suppose that there exists a function f^{-1} such that $f^{-1}(f(x)) = x$ for every $x \in \Sigma^n$ (which is necessarily unique when it exists). This means that the operation that transforms $|x\rangle$ into $|f(x)\rangle$ for every $x \in \Sigma^n$ is unitary, so we might hope to build a quantum circuit that implements the unitary operation defined by

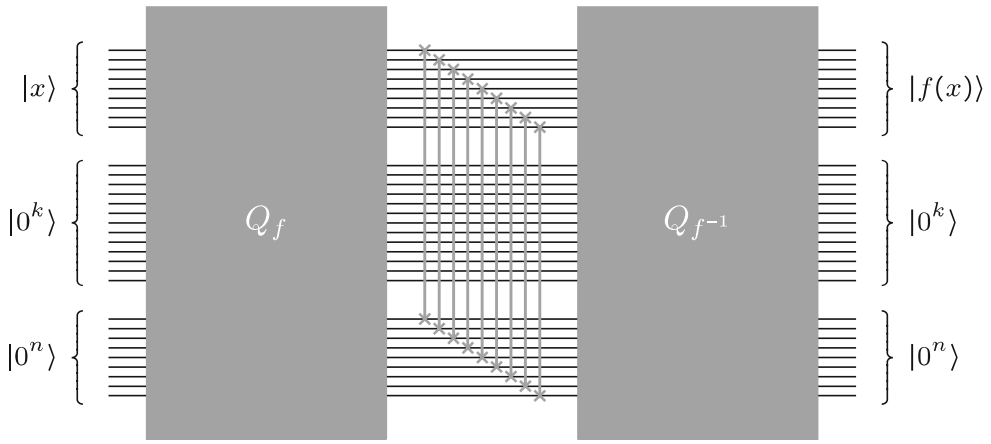
$$U|x\rangle = |f(x)\rangle$$

for every $x \in \Sigma^n$.

To be clear, the fact that this is a unitary operation relies on f being invertible — it's not unitary when f isn't invertible. Disregarding the workspace qubits, U is different from the operation that the circuit Q implements because we're not keeping a copy of the input around and XORing it to an arbitrary string, we're *replacing* x by $f(x)$.

The question is: when f is invertible, can we do this?

The answer is yes, provided that we're allowed to use workspace qubits and, in addition to having a Boolean circuit that computes f, we also have one that computes f^{-1}. So, this isn't a shortcut for computationally inverting functions when we don't already know how to do that! The following diagram illustrates how it can be done by composing two quantum circuits, Q_f and $Q_{f^{-1}}$, which are obtained individually for the functions f and f^{-1} through the method described above, along with n swap gates, taking k to be the maximum of the numbers of workspace qubits required by Q_f and $Q_{f^{-1}}$.



Was this page helpful?

Yes No

Report a bug, typo, or request content on GitHub ↗.

Previous page

Start the next lesson