Lecture 9

Instantiating quantum oracles.

Fact 2. Any $f\colon \{0,1\}^m \to \{0,1\}^n$ can be implemented as a classical circuit involving FANOUT, NOT, AND, OR gates.

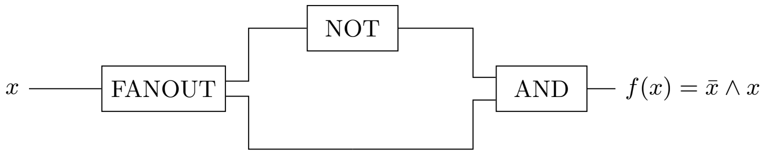
Proof. Exercise. □

Definition 2. For $f\colon \{0,1\}^m \to \{0,1\}^n$, the quantum oracle of f is the unitary (in fact, permutation) matrix $O_f \in \mathbb{C}^{2^{n+m} \times 2^{n+m}}$ defined by

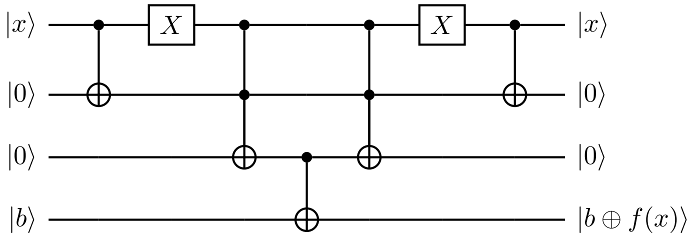
$$O_f\colon \, |x\rangle\,|y\rangle \mapsto |x\rangle\,|y \oplus f(x)\rangle\,. \tag{9}$$

Fact 3. Suppose we have the description of a size-s classical circuit implementing the function $f\colon \{0,1\}^n \to \{0,1\}^m$ involving FANOUT, NOT, AND, OR gates.[1] Then we can efficiently implement the quantum oracle for f using a size-$O(s)$ circuit involving X, CNOT, Toffoli gates.

Example 1. *Consider the following circuit involving three gates* FANOUT, NOT, AND *which implements* $f\colon \{0,1\}^n \to \{0,1\}$, $f(x) = x \wedge (\neg x)$.



The quantum oracle for f is implemented by the following quantum circuit.



The rule is you use CNOT *for* FANOUT, X *for* NOT, Toffoli *for* AND. *Then you use one extra* CNOT *to copy the output into a fresh qubit and reverse the computation by "reflecting" the previous gates about that* CNOT. *(Exercise: why does this reverse the computation?)*

Remark 1. Fact 3 is a rigorous proof that quantum computation subsumes deterministic classical computation. In fact, quantum computation also subsumes randomized classical computation since any coin flipping gates used to generate the randomness can be simulated by Hadamard gates (exercise).

Proof sketch of Fact 3. The example essentially gives the proof: we see FANOUT can be simulated using CNOT, NOT using X, AND using Toffoli. Note that OR can be simulated using X and Toffoli (think de Morgan's laws). For more details, see Section 6.3 of Watrous notes and accompanying video (in particular, starting at `45:37`). □

Remark 2. Does DJ mean we have a real (unconditional) provable exponential quantum speedup? **No.** For the quantum oracle of $f\colon \{0,1\}^n \to \{0,1\}$ to be reasonable, we need to instantiate f as a circuit.[2] But once we have a circuit description of f, we may be able to solve the problem faster than time $2^{n-1}+1$ by analyzing the circuit rather than only evaluating (aka querying) the circuit. Indeed in the example above, we could classically determine that f is constant by performing *Boolean algebra* on f's circuit rather than querying f.

If we are *forced* to only query f, then yes, DJ gives an exponential quantum speedup between quantum and randomized query complexity (if we also demand certain correctness). Being forced to only query f is the essential assumption of the query model of computation. The real-world model of computation (aka Turing model) does not make this assumption, so DJ does *not* constitute a provable exponential quantum speedup in the real world.

Comment: Pierre asked the interesting question whether P $\neq$ NP implies there exists a circuit for f such that the best thing to do is to query it. I thought more about this question after class and think that it is equivalent to asking if P $\neq$ NP implies black-box obfuscation. The answer to that is no as the latter has been proven to be impossible unconditionally. Note that what I threw out in class, indistinguishability obfuscation, is a weaker form of black-box obfuscation, which was recently shown to be possible under widely accepted cryptographic assumptions, e.g., hardness of the so-called LWE problem (but not P $\neq$ NP). (The colored text is clickable.)

[1] The size of a circuit is the total number of gates it contains.

[2] If a circuit sounds abstract, think of this as a program or description. These become circuits once you compile them to run on hardware.