Quantum Algorithms
Lecture 10
Correspondence between classical and quantum computation

Zhejiang University

Reminder of Chapter 6

Arbitrary quantum state

An arbitrary state of the system may be represented in the form

$$|\psi\rangle = \sum_{(x_1,\ldots,x_n)\in B^n} c_{x_1,\ldots,x_n}|x_1, \quad \ldots, x_n >$$

where $\sum_{(x_1,\ldots,x_n)\in B^n}\left|c_{x_1,\ldots,x_n}\right|^2 = 1$.

The state space for such a system is a linear space of dimension 2^n over the field $\mathbb{C}$ of complex numbers.

Amplitudes

The coefficients $c_{x_1,\ldots,x_n}$ of the decomposition of a vector $|\psi\rangle$ relative to this basis are called amplitudes. Their physical meaning is that the square of the absolute value $\left|c_{x_1,\ldots,x_n}\right|^2$ of the amplitude is interpreted as the probability of finding the system in the given state of the basis.

$$\sum_{(x_1,\ldots,x_n)\in B^n}\left|c_{x_1,\ldots,x_n}\right|^2 = 1.$$

Tensor product - example

$|\psi\rangle = a|0\rangle + b|1\rangle$
$|\varphi\rangle = c|0\rangle + d|1\rangle$
$|\psi\rangle \otimes |\varphi\rangle = (ac)|0\rangle \otimes |0\rangle + (ad)|0\rangle \otimes |1\rangle +$
$(bc)|1\rangle \otimes |0\rangle + (bd)|1\rangle \otimes |1\rangle$

Tensor product – how to represent two systems as one combined system.

Operators as matrices

Operators can be specified as matrices relative to the classical basis (or any other orthonormal basis):

$A = \sum_{j,k} a_{j,k} |j\rangle \langle k|$, where $a_{j,k} = \langle j|A|k\rangle$.

$|j\rangle\langle k|$ is a linear operator $(|j\rangle\langle k|)|\xi\rangle = \langle k|\xi\rangle|j\rangle$.

Operator applied to register

Let $U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$. Then the operators $U[1]$ and $U[2]$, acting on the space $B^{\otimes 2}$, are represented by these matrices:

$$U[1] = \begin{pmatrix} u_{00} & 0 & u_{01} & 0 \\ 0 & u_{00} & 0 & u_{01} \\ u_{10} & 0 & u_{11} & 0 \\ 0 & u_{10} & 0 & u_{11} \end{pmatrix}, \qquad U[2] = \begin{pmatrix} u_{00} & u_{01} & 0 & 0 \\ u_{10} & u_{11} & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{pmatrix}$$

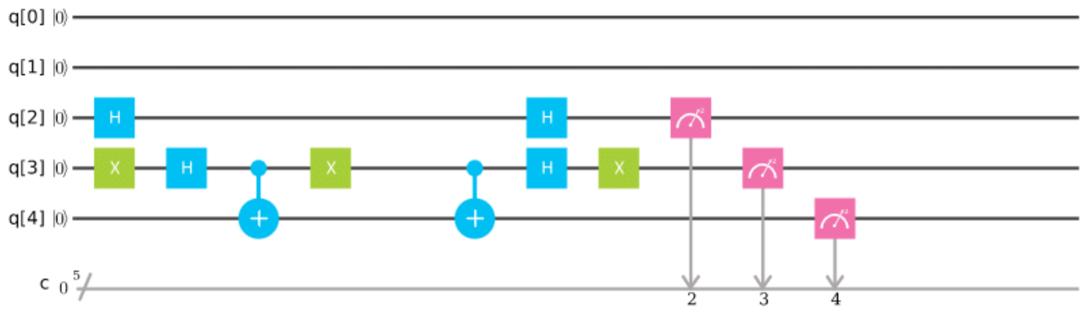
The rows and columns are associated with the basis vectors arranged in the lexicographic order:

|00>, |01>, |10>, |11>.

Quantum circuit

Let A be a fixed set of unitary operators. (We call A a basis, or a gate set, whereas its elements are called gates.) A quantum circuit over the basis A is a sequence $U_1[A_1], \ldots, U_L[A_L]$, where $U_j \in A$, and A_j is an ordered set of qubits.

The operator realized by the circuit is $U = U_L[A_L] \cdots U_1[A_1]$ ($U\colon B^{\otimes n} \to B^{\otimes n}$). The number L is called the size of the circuit.



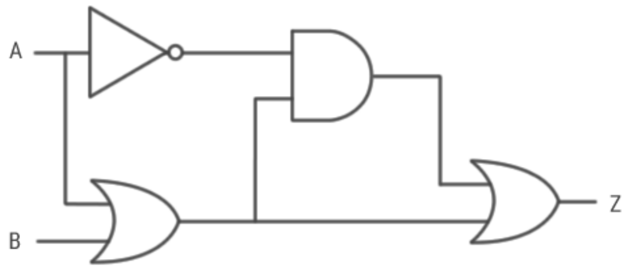
QC with ancillas

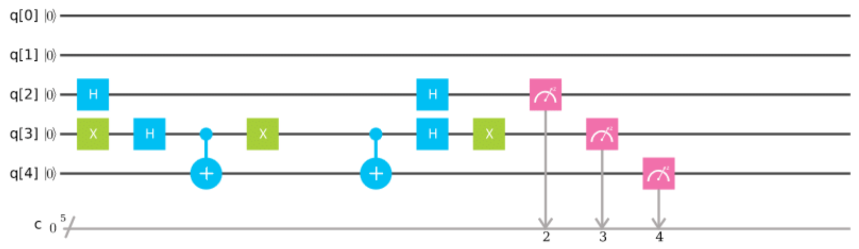
Operator realized by a quantum circuit using ancillas is an operator $U\colon B^{\otimes n} \to B^{\otimes n}$ such that the product $\mathrm{W} = U_L[A_L] \cdots U_1[A_1]$, acting on N qubits ($N \geq n$), satisfies the condition $W(|\xi\rangle \otimes |0^{N-n}\rangle) = (U|\xi\rangle) \otimes |0^{N-n}\rangle$ for any vector $|\xi\rangle \in B^{\otimes n}$.

Introduction

Boolean circuits in quantum

Quantum computation is supposed to be more general than classical computation. However, quantum circuits do not include Boolean circuits as a special case. Therefore some work is required to specialize the definition of a quantum circuit and prove that the resulting computational model is equivalent to the Boolean circuit model.





Reversible circuits

The classical analogue of a unitary operator is an invertible map on a finite set, i.e., a permutation. An arbitrary permutation $\mathrm{G}\colon B^{\otimes k} \to B^{\otimes k}$ corresponds naturally to a unitary operator $\hat{G}$ on the space $B^{\otimes k}$ acting according to the rule $\hat{G}|x\rangle = |Gx\rangle$.

We may define reversible classical circuits, which realize permutations.

Permutations

Permutations

A permutation of a set is, loosely speaking, an arrangement of its members into a sequence or linear order, or if the set is already ordered, a rearrangement of its elements.

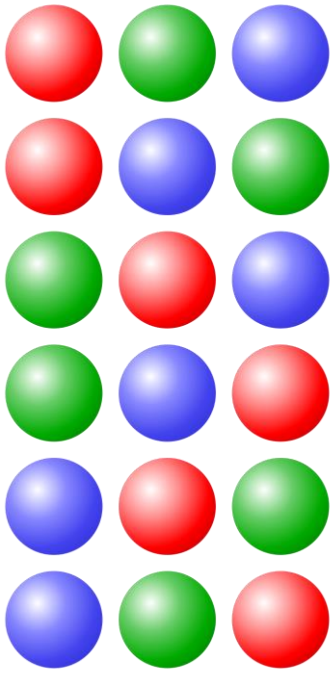
Example for set $\{1,2,3\}$

Permutation f:

$f(1) = 3$

$f(2) = 1$

$f(3) = 2$



The image is an example for permutation of 3 circles. We see that there are 6 total possibilities – permutations.

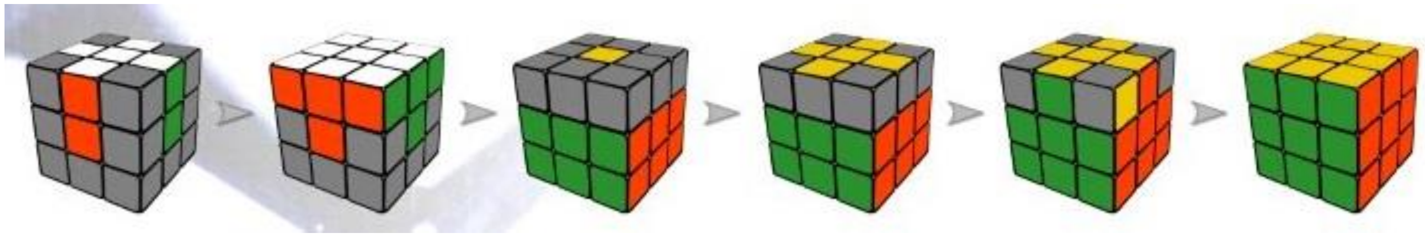
Reversible classical circuit

Let A be a set of permutations of the form G: $B^{\otimes k} \to B^{\otimes k}$. (The set A is called a basis; its elements are called gates.) A reversible classical circuit over the basis A is a sequence of permutations $G_1[A_1], \ldots, G_l[A_l]$, where A_j is a set of bits and $G_j \in A$.

Permutation realized by a reversible circuit

This is the product of permutations $G_l[A_l] \cdots G_1[A_1]$.

Example: $\{1,2,3\} \to \{3,1,2\} \to \{2,3,1\}$.



Permutation with ancillas

This is a permutation G such that the product of permutations

$$W = G_l[A_l] \cdots G_1[A_1]$$

(acting on N bits, $N \geq n$) satisfies the condition $W(x, 0^{N-n}) = (G_x, 0^{N-n})$ for arbitrary $x \in B^n$.

Reversible Boolean function

In what cases a function given by a Boolean circuit can be realized by a reversible circuit? Reversible circuits realize only permutations, i.e., invertible functions. This difficulty can be overcome in this way: instead of computing a general Boolean function $\mathrm{F}\colon B^n \to B^m$, we compute the permutation $F_{\oplus}\colon B^{n+m} \to B^{n+m}$ given by the formula $F_{\oplus}(x, y) = (x, y \oplus F(x))$. Then $F_{\oplus}(x, 0) = (x, F(x))$ contains the value of $F(x)$ we need.

$\oplus$ denotes bitwise addition modulo 2.

Permutations on two bits

Note that two-bit permutation gates do not allow to realize all functions of the form $F_{\bigoplus}$. It turns out that any permutation on two-bit states, $g: B^2 \to B^2$, is a linear function (under the natural identification of the set B with the two-element field F_2, i.e., with elements 0 and 1): $g(x, y) = (ax \oplus by \oplus c, dx \oplus ey \oplus f)$, where $a, b, c, d, e, f \in F_2$. Therefore, all functions realized by reversible circuits over the basis of permutations on two bits, are linear.

Permutations on three bits

However, permutations on three bits already suffice to realize any permutation. In fact, the following two functions form a complete basis for reversible circuits: negation $\neg$ and the Toffoli gate, $\Lambda_{\bigoplus} : (x, y, z) \rightarrow (x, y, z \bigoplus xy)$. Here, we mean realization using ancillas, i.e., it is allowed to borrow bits in the state 0 under the condition that they return to the same state after the computation is done.

Lemma for reversible circuit

Lemma

Let a function $\mathrm{F}\colon B^n \to B^m$ be realized by a Boolean circuit of size L and depth d over some basis A (the fan-in and fan-out being bounded by a constant). Then we can realize a map of the form $(x, 0) \to (F(x), G(x))$ by a reversible circuit of size $O(L)$ and depth $O(d)$ over the basis $A_\oplus$ consisting of the functions $f_\oplus$ $(f \in A)$ and the function $\circledcirc$: $(x, y) \to (x, x \oplus y)$.

Remark

In addition to the “useful” result $F(x)$, the indicated map produces some “garbage” $G(x)$.

$$(x, 0) \to (F(x), G(x))$$

Remark

The gate ⨷ is usually called “Controlled NOT” for reasons that will become clear later. Note that ⨷$= I_{\oplus}$, where I is the identity map on a single bit. The essential meaning of the operation ⨷ is reversible copying of the bit x (if the initial value of y is 0).

$$⨷\colon\ (x, y) \rightarrow (x, x \oplus y)$$

Remark

The gate $\circledcirc\!\!\!\otimes$ allows one to interchange bits in memory, since the function $(\leftrightarrow): (a,b) \to (b,a)$ can be represented as follows:

$$(\leftrightarrow)[j,k] \ = \circledcirc\!\!\!\otimes\, [j,k] \circledcirc\!\!\!\otimes\, [k,j] \circledcirc\!\!\!\otimes\, [j,k]$$

Proof of Lemma

Consider the Boolean circuit that computes F. Let the input variables be $x_1, \dots, x_n$, and the auxiliary variables (including the result bits) $x_{n+1}, \dots, x_{n+L}$. A reversible circuit we are to construct will also have $n+L$ bits; the bits $x_{n+1}, \dots, x_{n+L}$ are initialized by 0.

Proof of Lemma

Each assignment in the original (Boolean) circuit has the form $x_{n+k} = f_k(x_{j_k}, \dots, x_{l_k})$, $f_k \in A$, $j_k, \dots, l_k < n + k$. In the corresponding reversible circuit, the analogue of this assignment will be the action of the permutation $(f_k)_{\oplus}$, i.e., $x_{n+k} = x_{n+k} \oplus f_k(x_{j_k}, \dots, x_{l_k})$.

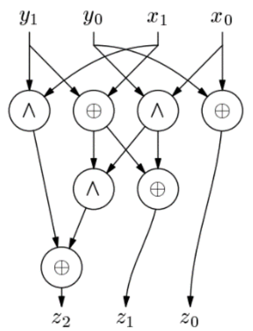
Proof of Lemma

Since the initial values of the auxiliary variables were equal to 0, their final values will be just as in the original circuit. In order to obtain the required form of the result, it remains to change positions of the bits.

Final values: $f_k(x_{j_k}, \dots, x_{l_k})$.

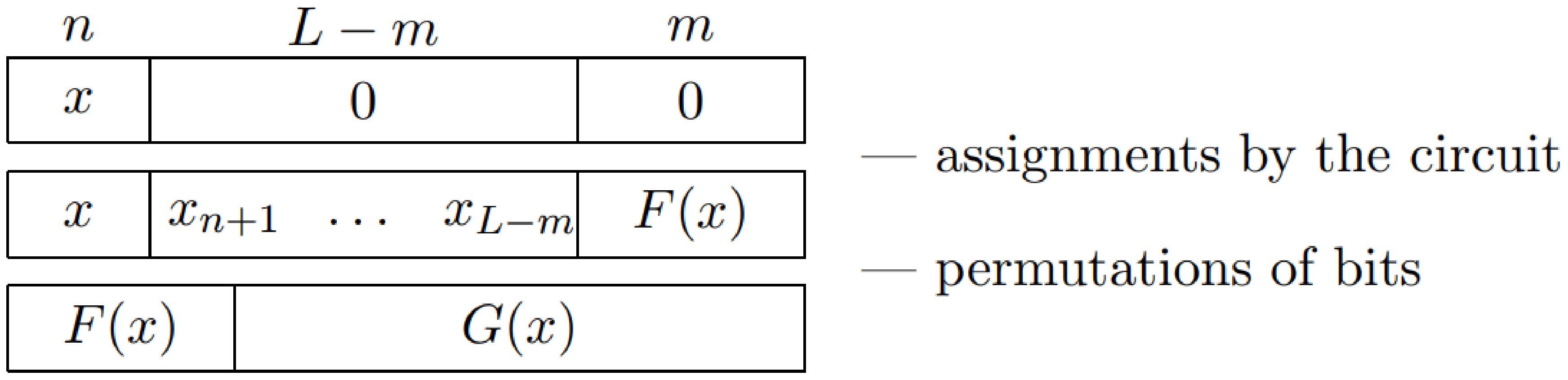
Proof of Lemma

In this argument, we may assume that the original circuit has a layered structure, so that several assignments can occur simultaneously. However, the concurrent assignments should not share their input variables. If this is not the case, we need to insert explicit copy gates between the layers; each copy gate will be replaced by $\circledcirc$ in the reversible circuit. This results in depth increase by at most constant factor, due to the bounded fan-out condition.



Proof of Lemma

The entire computational process is conveniently represented by the following diagram (above the rectangles is written the number of bits and, inside, their content).


n
$L-m$
m
x
0
0
x
$x_{n+1} \quad \ldots \quad x_{L-m}$
$F(x)$
$F(x)$
$G(x)$
— assignments by the circuit
— permutations of bits

Garbage removal

Garbage removal

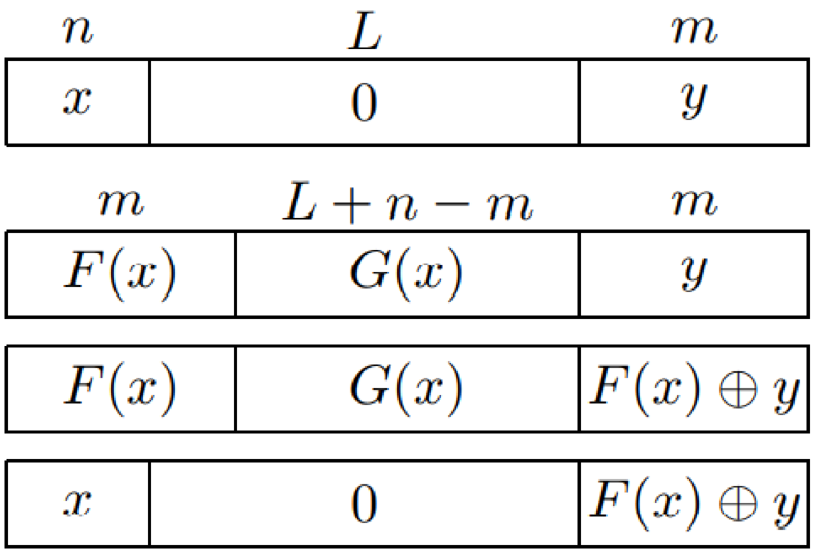
Under the conditions of previous Lemma, one can realize the function $F_{\oplus}$ by a reversible circuit of size $O(L+n+m)$ and depth $O(d)$ using ancillas.

Computation that we performed:

$$(x,0) \to (F(x), G(x))$$

Garbage removal - proof

We perform the computation $(x, 0) \to (F(x), G(x))$, add each bit of the result to the corresponding bit of y with $\otimes$, and undo the above computation.



— computation by the circuit from the proof of Lemma 7.1

— addition of $F(x)$ to y modulo 2

— reversal of the computation that was done in the first step

Remark – energy cost

Reversible computation provides an answer to the following question: how much energy is required to compute a given Boolean function? Theoretically, reversible operations can be performed at no energy cost.

Remark – irreversible operations

On the other hand, irreversible operations, like bit erasure, pose a fundamental problem. When such an operation is performed, two different logical states (0 and 1) become identical (0). However, physical laws on a micro-scale are reversible. The solution to this apparent paradox is that the difference between the initial states, 0 and 1, is converted into a difference between two physical states that both correspond to the same logical value 0.

Remark - heat

This may be interpreted as an increase in disorder (entropy) in physical degrees of freedom beyond our control, which eventually appears in the surrounding environment in the form of heat. The amount of energy required to erase a single bit is very small ($kT \ln 2$), but still nonzero.

Remark – amount of energy

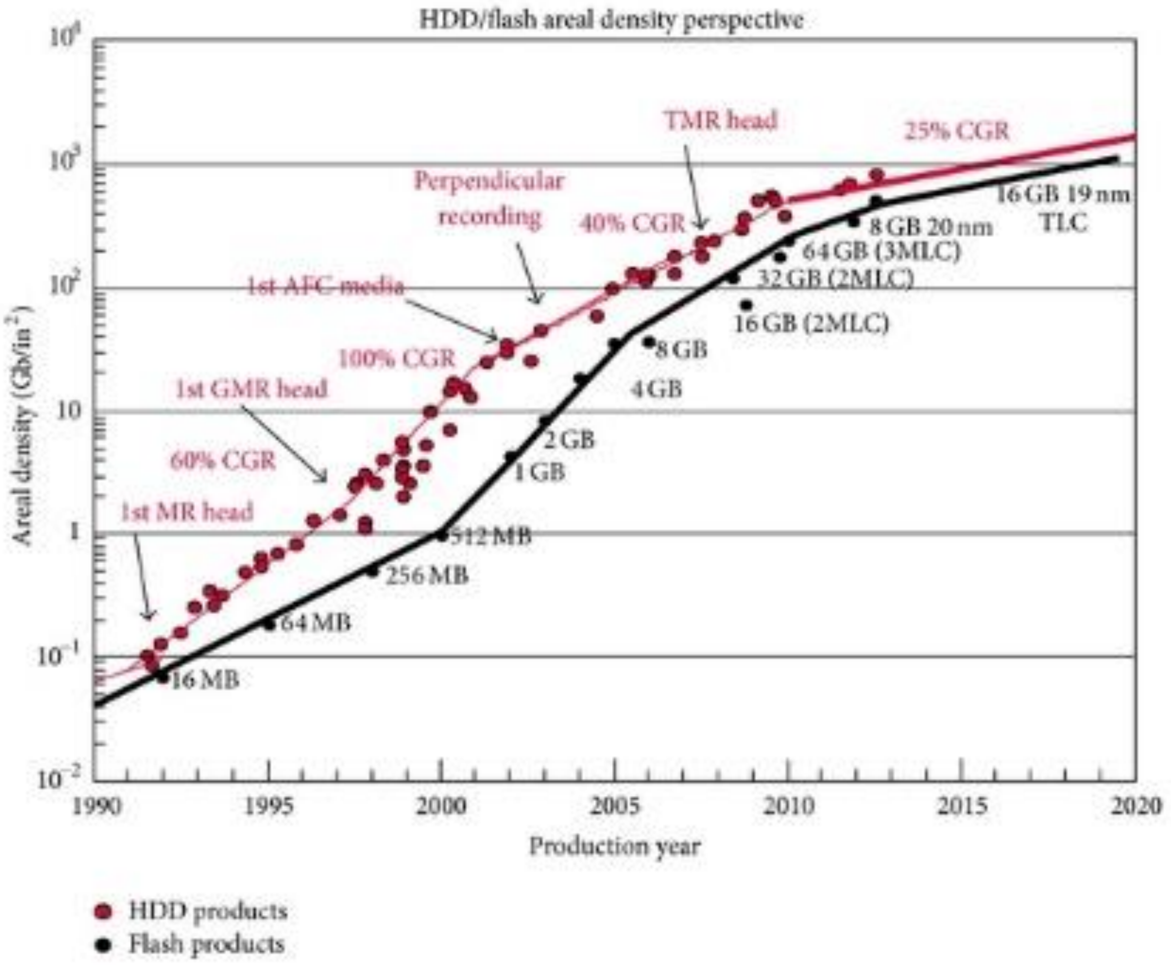
The theoretical energy cost of information erasure on a hard disk of capacity 1 gigabyte is equal to $2.4 \cdot 10^{-11}$ Joules, which corresponds to the energy spent in moving the disk head by a fraction of the size of an atom. This is many orders of magnitude smaller than the actual displacement of the head through formatting.

Remark – for big disks

On the other hand, if the capacity of disks were to continue growing as fast as now, then at the end of the twenty-third century formatting of a hard disk would require as much energy as the Sun generates in a year.

The garbage removal lemma shows that it is possible to avoid such losses of energy connected with irreversible operations.

Remark – for big disks



Reversible computation and memory

Memory usage

It is likewise possible to show that arbitrary computation performed with memory s, can be realized in a reversible manner through the use of memory not exceeding $s^{O(1)}$. We will give a sketch of the proof. However, we should keep in mind that computation with bounded space is not easily defined in terms of circuits.

Memory usage

Indeed, if a circuit is allowed to be exponentially large (though of polynomial "width"), it can contain the value table of the desired function, which makes its computation trivial. Therefore, a rigorous proof should either deal with circuits of some regular structure, or involve a definition of a reversible Turing machine.

Reduction to TQBF

An arbitrary computation with a given memory s can be reduced to solving a $poly(s)$-size instance of TQBF, since TQBF is PSPACE-complete. We will show how to compute reversibly, with a small amount of memory, the value of the formula

$$\exists x_1 \forall y_1 \cdots \exists x_M \forall y_M f(x_1, y_1, \ldots, x_M, y_M, z),$$

where $f(\cdot)$ is computed by a Boolean circuit of size L.

Reduction to TQBF

Actually, in this case the value of the formula can be represented by a reversible circuit with $O(L+M)$ bits. The computation will be organized recursively, beginning with the innermost quantifiers.

$$\exists x_1 \forall y_1 \cdots \exists x_M \forall y_M f(x_1, y_1, \ldots, x_M, y_M, z)$$

Reduction to TQBF

In order to compute $\forall x F(x,z)$, we compute $F(0,z)$ and put the result into a supplementary bit. Then we compute $F(1,z)$ and put the result into another bit. Next we compute $\forall x F(x,z) = F(0,z) \wedge F(1,z)$ and save the result in a third bit. In order to remove the garbage, we undo all calculations, except for the final step.

$$\exists x_1 \forall y_1 \cdots \exists x_M \forall y_M f(x_1, y_1, \ldots, x_M, y_M, z)$$

Reduction to TQBF

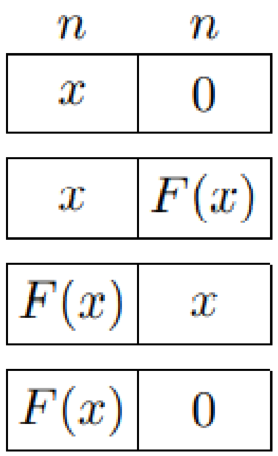
Dealing with the formula $\exists x F(x, y)$ in a similar manner, we arrive at the following result: adding a quantifier in one Boolean variable increases the required memory by at most a constant number of bits.

$$\exists x F(x, z) = F(0, z) \vee F(1, z)$$

$$\exists x_1 \forall y_1 \cdots \exists x_M \forall y_M f(x_1, y_1, \dots, x_M, y_M, z)$$

Theorem

Let F and F^{-1} be computed by Boolean circuits of size $\leq L$ and depth $\leq d$. Then F can be realized by a reversible circuit of size $O(L+n)$ and depth $O(d)$ using ancillas.



— computation of $F_\oplus$ by the circuit from the proof of Lemma 7.2

— permutation of bits

— applying $(F^{-1})_\oplus$ (by the circuit from the proof of Lemma 7.2) yields $x \oplus F^{-1}(F(x)) = 0$ in the right register

Complete basis

Negation and the Toffoli gate form a complete basis for reversible circuits.

¬ and $\Lambda_{\oplus}: (x, y, z) \to (x, y, z \oplus xy)$

Complete basis

Since the conjunction $\wedge$ and the negation $\neg$ form a complete basis for Boolean circuits, and it is sufficient to realize the functions $\wedge_{\oplus}$ (i.e., the Toffoli gate), $\neg_{\oplus}$: $(x, y) \to (x, x \oplus y \oplus 1)$ and $\circledcirc$. But the Toffoli gate is already in the basis, $\neg_{\oplus}[1, 2] = \neg[2] \circledcirc [1, 2]$, so it suffices to realize $\circledcirc$. Let us introduce an auxiliary bit u initialized by 0. Then the action of $\circledcirc [1, 2]$ can be represented as $\neg[u] \wedge_{\oplus} [u, 1, 2] \neg[u]$.

Ideas for circuit comparisons

Ensuring reversibility

This is the most general approach to make quantum computation reversible, instead of computing function F: $B^n \to B^m$, we compute the permutation $F_{\bigoplus}$: $B^{n+m} \to B^{n+m}$.

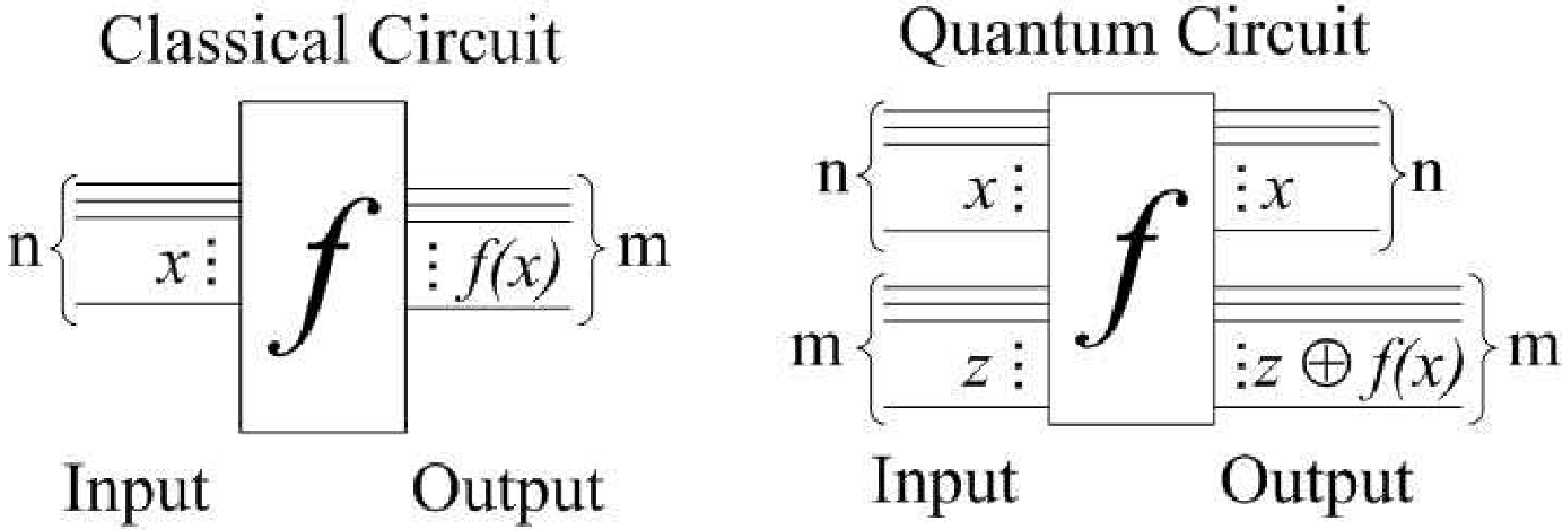
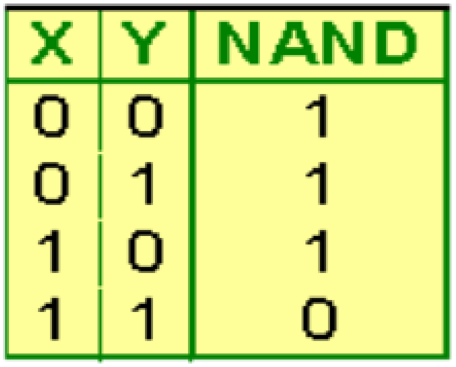


Figure 2.1: Differences between classical circuits and quantum circuits

NAND – universal classical gate

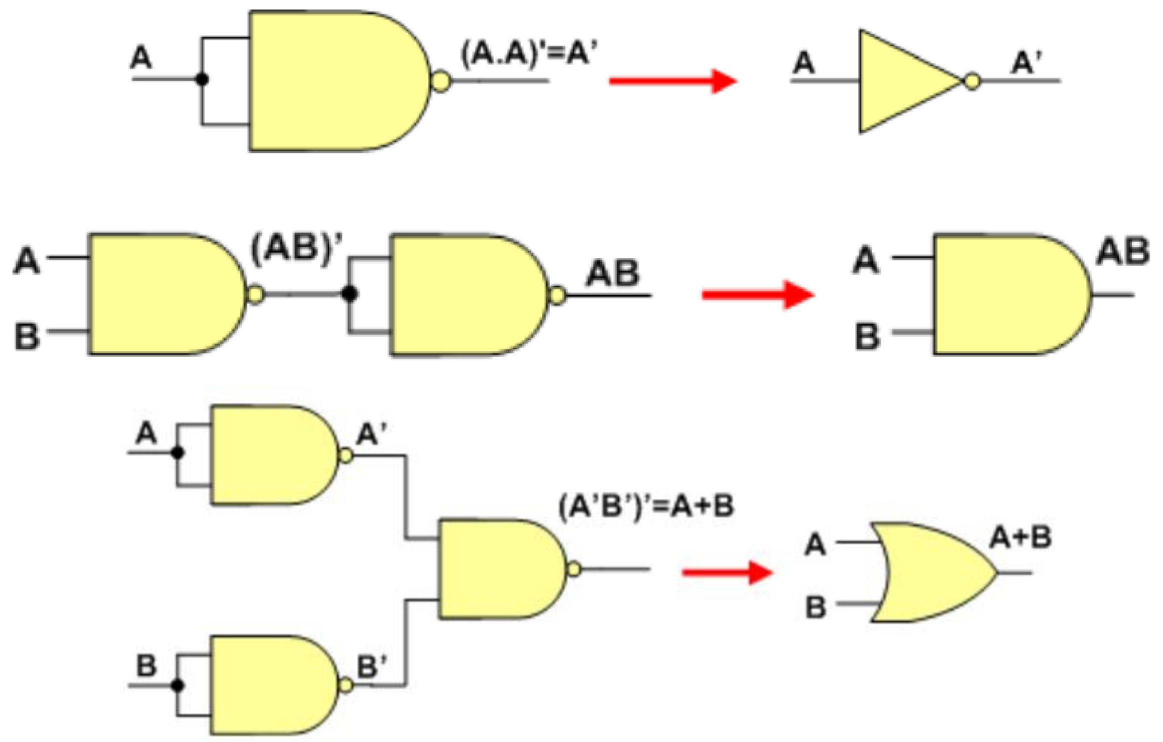
The NAND gate represents the complement of the AND operation. Its name is an abbreviation of NOT AND. The truth table and the graphic symbol of NAND gate is shown in the figure:



X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

NAND – universal classical gate

To prove that any Boolean function can be implemented using only NAND gates, we can show that the AND, OR, and NOT operations can be performed using only these gates.



NAND – to quantum operation

Toffoli gate allows us to calculate AND value of first bits by storing it in the third bit. Therefore, to make it work like NAND gate, we just need to reverse the resulting bit – either by applying NOT operator, or by initializing it in state 1 initially.

$$\text{NAND} = \neg\Lambda_{\oplus} : (x, y, 1) \rightarrow (x, y, 1 \oplus xy)$$

Thank you for your attention!