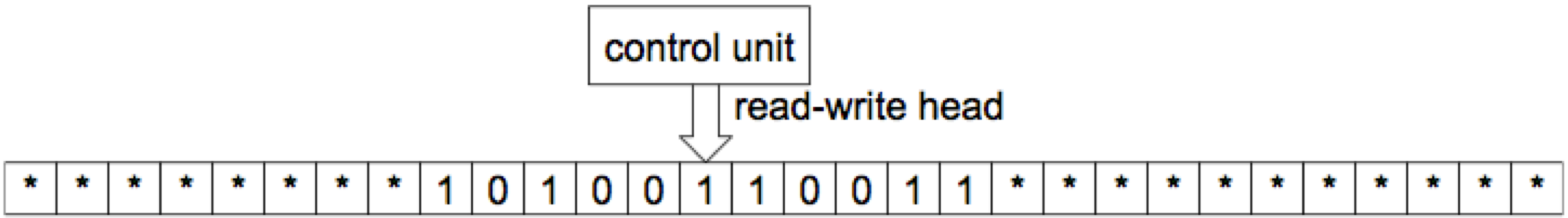
CLASSICAL AND QUANTUM COMPUTATION

COMPUTATIONAL COMPLEXITY CLASSES

Deterministic computation and deterministic Turing machine

Turing machine consists of

1. a finite *alphabet* Σ containing the blank symbol $*$;
2. a 2-way infinite tape divided into cells, one of which is a special *starting cell*. Each cell contains a symbol from the alphabet Σ. All but a finite number of cells contain the special blank symbol $*$, denoting an empty cell;
3. *read-write head* that examines a single cell at a time and can move left ($\leftarrow$) or right ($\rightarrow$);
4. a *control unit* along with a finite set of states Γ including a distinguished starting state, γ_0, and a set of *halting states*.



The computation of a Turing machine is controlled by a *transition function*:

$$\delta : \quad \Gamma \times \Sigma \quad \rightarrow \quad \Gamma \times \Sigma \times \{\leftarrow, \rightarrow\}$$

Example: Unary addition Turing machine

States: $\Gamma = \{\gamma_0, \gamma_1, \gamma_2, \gamma_3\}$ with the starting state γ_0 and the halting state γ_3;

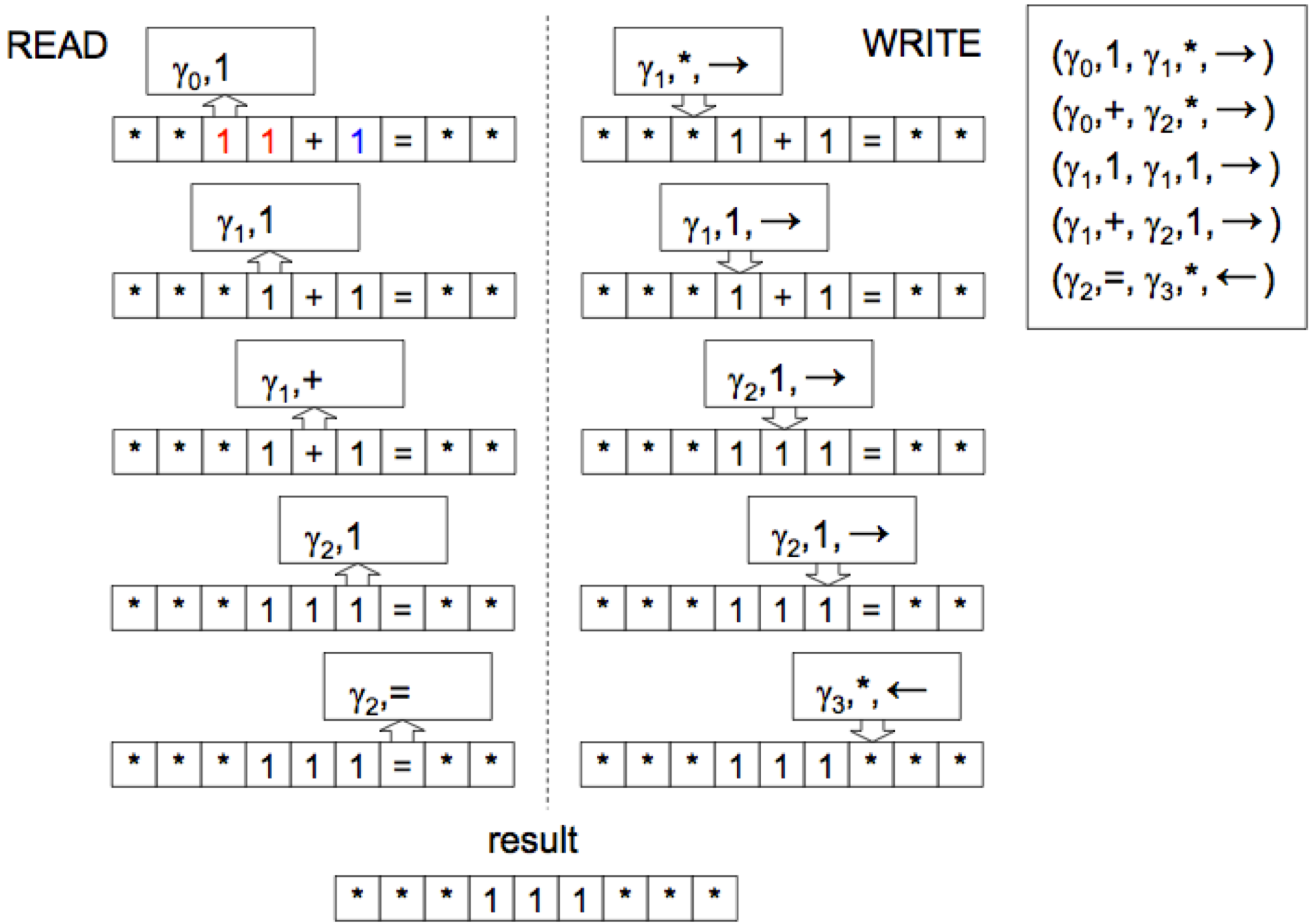
Alphabet: $\Sigma = \{*, 1, +, =\} = \Sigma_0 \cup \{*\}$ where Σ_0 is called external alphabet;

Input: integers $a, b \geq 0$ with the symbol $+$ and $=$
(e.g. $2 + 1$ is written as '$11 + 1 =$' on the tape with the leftmost input symbol in the starting square);

Output: $a + b$ unary

Transition function:

$(\gamma_0, 1, \gamma_1, *, \rightarrow)$	$a \neq 0,$ reading a
$(\gamma_0, +, \gamma_2, *, \rightarrow)$	$a = 0,$ erase +, read b
$(\gamma_1, 1, \gamma_1, 1, \rightarrow)$	reading a
$(\gamma_1, +, \gamma_2, 1, \rightarrow)$	replace + by $1,$ read b
$(\gamma_2, =, \gamma_3, *, \leftarrow)$	finish reading $b,$ erase =, halt


READ
WRITE
result
(γ0,1, γ1,*, →)
(γ0,+, γ2,*, →)
(γ1,1, γ1,1, →)
(γ1,+, γ2,1, →)
(γ2,=, γ3,*, ←)
γ0,1
γ1,1
γ1,+
γ2,1
γ2,=
γ1,*, →
γ1,1, →
γ2,1, →
γ2,1, →
γ3,*, ←

Church-Turing thesis

Any algorithm can be realized by a Turing machine.

A Turing machine is a finite object, so it can be encoded by a string. Then for any fixed alphabet Σ_0^*, we can consider a *universal Turing machine* U which computes the function

$$u([M], x) = \phi_M(x)$$

where $[M]$ is the encoding of a Turing machine M.

Computable functions and decidable predicates

Every Turing machine M computes a function

$$\phi_M : \Sigma_0^* \to \Sigma_0^*$$

where Σ_0^* is the set of all strings over Σ_0 (external alphabet). $\phi_M(x)$ is the output string for input x. The value of $\phi_M(x)$ is undefined if the computation never terminates.

A function $f : \Sigma_0^* \to \Sigma_0^*$ is *computable* if there exists a Turing machine M such that $\phi_M = f$. In this case we say f is computed by M.

A predicate is a function $L : \Sigma_0^* \to \{0, 1\}$, a function with a Boolean value. A predicate is called *decidable* if this function is computable.

Decision problems

An input $x \in \Sigma_0^*$ is *accepted* by an (acceptor) deterministic TM, if the machine terminates in the state γ_T on input x and is *rejected* if it halts in state γ_F.

Any set of string $L \subseteq \Sigma_0^*$ is called a language. If M is an acceptor deterministic TM, then we define the *language accepted by M* to be

$$L(M) = \left\{x \in \Sigma_0^* \mid M \text{ accepts } x\right\}$$

If M halts on all inputs $x \in \Sigma_0^*$ then we say that M *decides* L.

For a general decision problem Π we have the *associated language*

$$L_\Pi = \left\{x \in \Sigma_0^* \mid x \text{ is a natural encoding of a true instance of } \Pi\right\}.$$

Example

PRIME

Input: an integer $n \geq 2$.

Question: is n prime?

$$L_{\text{PRIME}} = \{x \mid x \text{ is the binary encoding of a prime number.}\}$$

Complexity

A TM works in time $T(n)$ if it performs *at most* $T(n)$ steps for any input of size n.

A function/predicate F on $\mathbb{B}^*$, that is on binary strings, is *computable/decidable in polynomial time* if there exists a TM that computes it in time $T(n) = poly(n)$, where n is the input length.

A class of all functions (predicates) computable (decidable) in polynomial time is called P.

We say that these functions are efficiently solvable or tractable on deterministic Turing machine.

P = $\{L \subseteq \Sigma_0^*$ | there is a deterministic TM M which decides L and a polynomial $p(n)$ such that $T(n) \leq p(n)$ for all $n \geq 1\}$

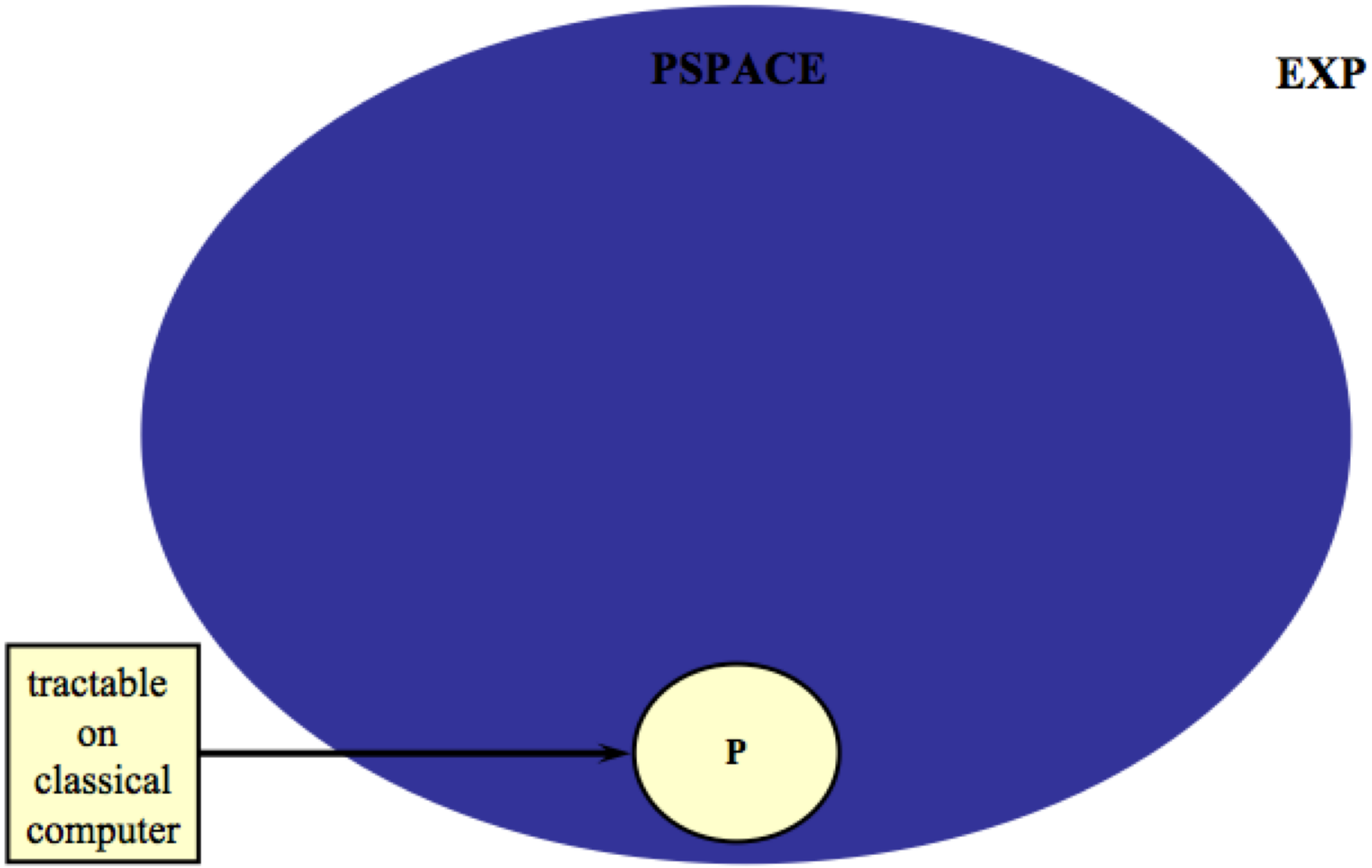
A TM works in space $s(n)$ if it visits at most $s(n)$ cells for any computation on inputs of size n.

A function (predicate) F on $\mathbb{B}^*$ is *computable (decidable) in polynomial space* if there exists a TM that computes F and runs in space $s(n) = poly(n)$ where n is the input length.

A class of all functions (predicates) computable (decidable) in polynomial space is called PSPACE.

P $\subseteq$ PSPACE

It is generally believed that this inclusion is strict though this is an open question.



Non-deterministic computation

A non-deterministic Turing machine is a hypothetical machine that resembles a deterministic Turing machine but can non-deterministically choose one of several actions possible in a given configuration. **Its transition function is multivalued.**

A predicate L belongs to the **class NP, Non-deterministic Polynomial**, if there exist a non-deterministic Turing machine M and a polynomial $p(n)$ such that

$L(x) = 1 \quad \Rightarrow$ there exists a computational path that gives the answer 'yes' in time $p(|x|)$, where $|x|$ is the size of the input;

$L(x) = 0 \quad \Rightarrow$ there is no path with this property.

Alternative definition of the complexity class NP (Kitaev)

Imagine two persons: King Arthur (with polynomially bounded mental capabilities) and a wizard Merlin (intellectually omnipotent). Arthur is interested in $L(x)$. Merlin wants to convince Arthur that $L(x)$ is true, but Arthur does not trust Merlin (he is too smart to be loyal) and wants to make sure that $L(x)$ is true.

So Arthur arranges that, after both he and Merlin see input string x, Merlin writes a note to Arthur where he proves that $L(x)$ is true. Then Arthur verifies this proof by some polynomial proof-checking predicate (procedure)

$$R(x, y) = \text{"}y \text{ is a proof of } L(x)\text{"}$$

where $L(x) = 1$ implies that Merlin can convince Arthur that $L(x)$ is true by presenting some proof y such that $R(x, y)$; and $L(x) = 0$ implies that whatever Merlin says, Arthur is not convinced: $R(x, y)$ is false for any y.

NP, NP hardness and NP completeness

A predicate L_1 is *reducible* to a predicate L_2 if there exists a function $f \in$ P such that $L_1(x) = L_2(x)$ for any input string x. We say $L_1 \propto L_2$.

Lemma: Let $L_1 \propto L_2$, then

(a) $L_2 \in$ P $\quad\Rightarrow\quad L_1 \in$ P

(a) $L_2 \notin$ P $\quad\Rightarrow\quad L_1 \notin$ P

(a) $L_2 \in$ NP $\quad\Rightarrow\quad L_1 \in$ NP

Predicate L is NP-*hard* if any predicate in NP is reducible to it.

Predicate L is NP-*complete* if it is NP-hard and $L \in$ NP.

Example: SAT (satisfiability)

SAT(x) means that x is a propositional formula, containing Boolean variables and operations (negation, disjunction, conjunction) that is satisfiable, that is ”true” for some values of the variables.

Cook-Levin Theorem:

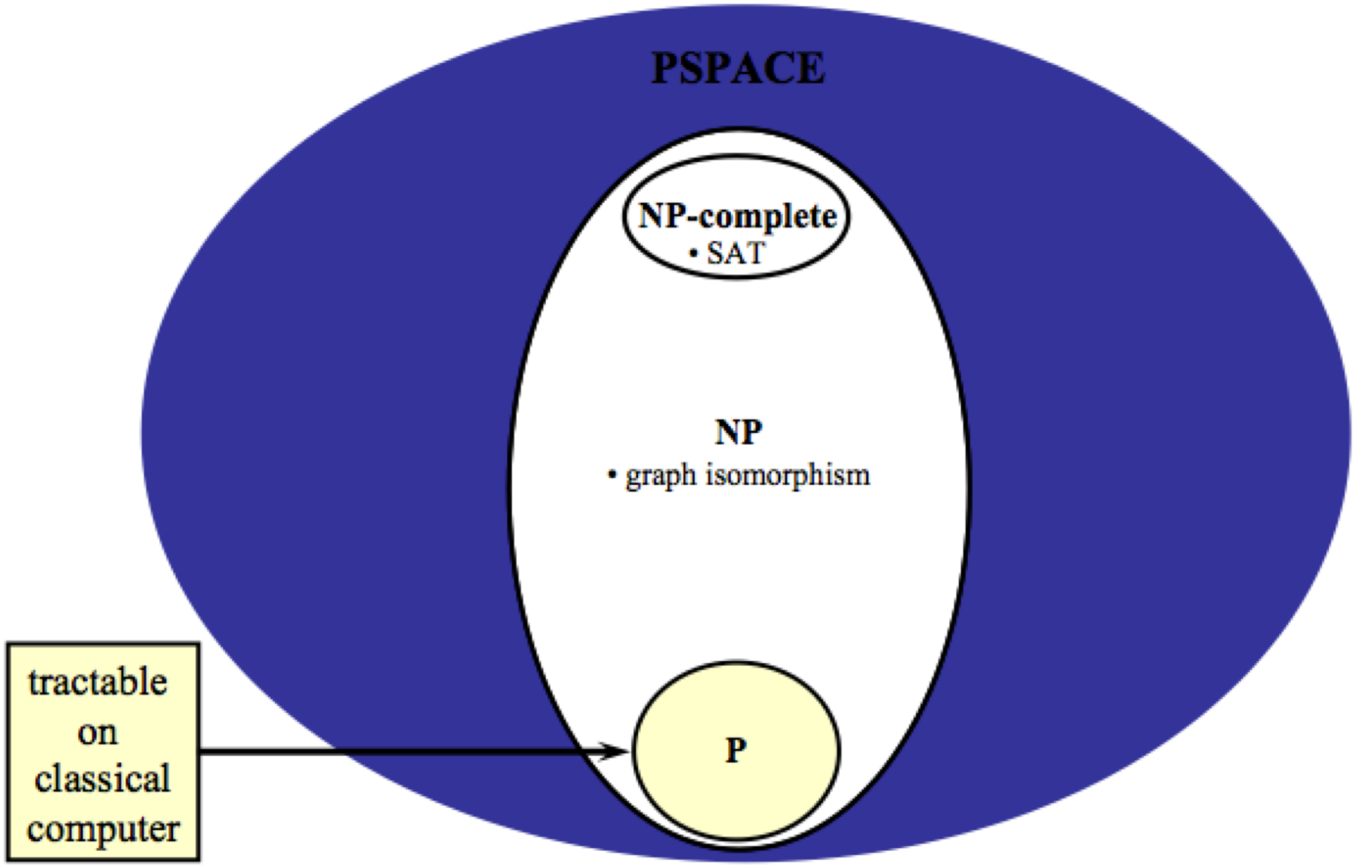
$$\text{SAT} \in \text{NP}$$
$$\text{SAT} \in \text{NP-complete}$$

Other examples: 3-COLORING, CLIQUE, ...

$P \subseteq NP \subseteq PSPACE$

Again it is believed that the inclusions are strict though this is an open question.

If you could prove that SAT $\in$ P, then you would resolve the problem P vs. NP which is one of the Millenium problems of the Clay Mathematics Institute with a prize of $ 1,000,000.



Probabilistic computation

A probabilistic Turing machine can probabilistically choose one of several actions possible in a given configuration. This is similar to a non-deterministic TM but the choice is made by coin tossing rather than guessing. PTM is in principle physical.

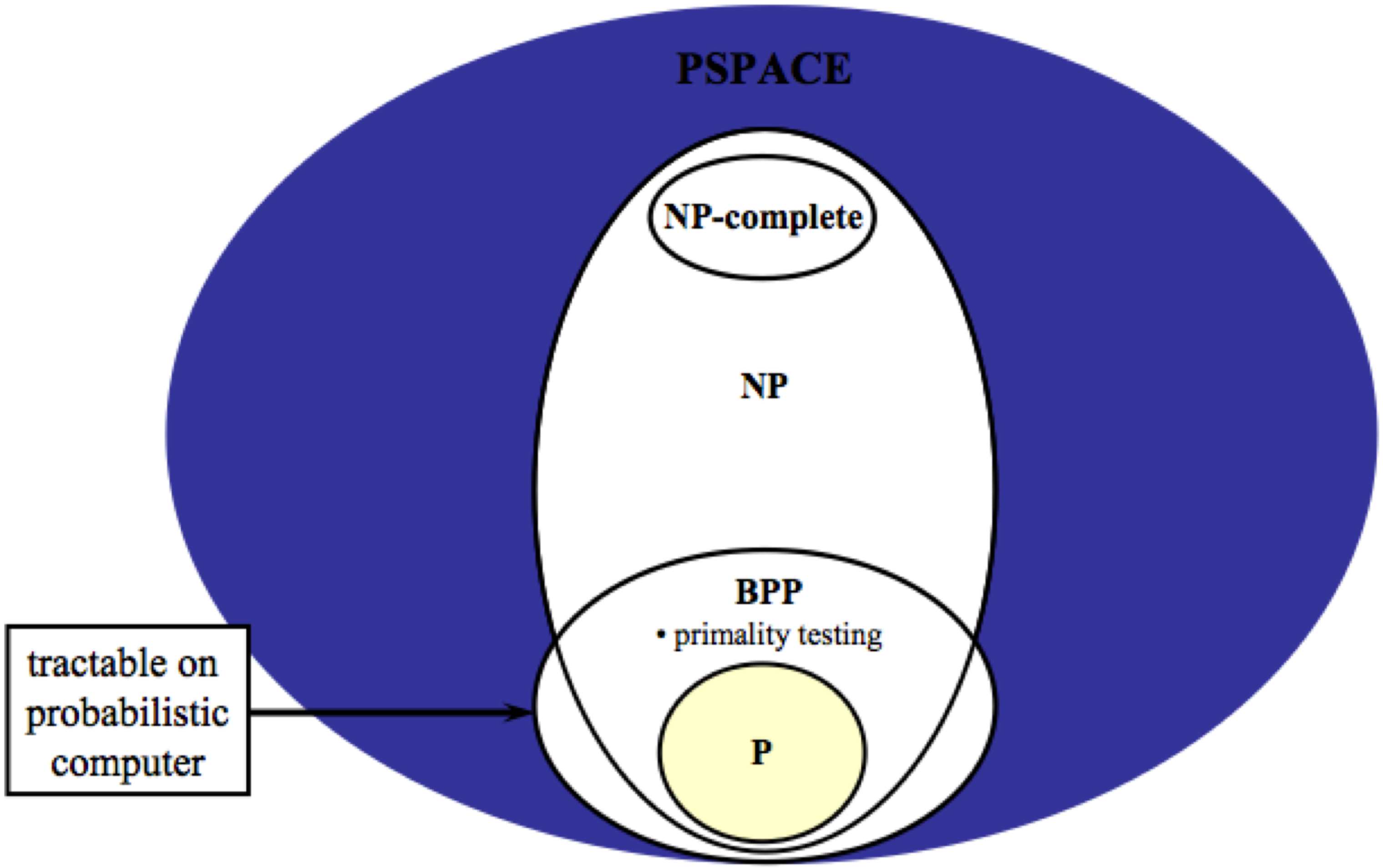
Let ϵ be a constant such that $0 < \epsilon < \frac{1}{2}$. A predicate L belongs to the **class BPP, Bounded-error Probabilistic Polynomial,** if there exists a probabilistic Turing machine M and a polynomial $p(n)$ such that the machine M running on input string x always terminates at most $p(|x|)$ steps, and

$L(x) = 1 \quad \Rightarrow \quad$ M gives the answer 'yes' with probability $\geq 1 - \epsilon$;

$L(x) = 0 \quad \Rightarrow \quad$ M gives the answer 'no' with probability $\leq \epsilon$.

Example: PRIMALITY, i.e. checking whether a given integer is a prime number.

$P \subseteq BPP \subseteq PSPACE$



Quantum computation

A quantum Turing machine can choose a superposition of several actions in a given configuration. This is somewhat similar to a probabilistic TM.

Let ϵ be a constant such that $0 < \epsilon < \frac{1}{2}$. A predicate L belongs to the **class BQP, Bounded-error Quantum Polynomial,** if there exists a quantum Turing machine M and a polynomial $p(n)$ such that the machine M running on input string x always terminates at most $p(|x|)$ steps, and

$L(x) = 1 \quad \Rightarrow \quad$ M gives the answer 'yes' with probability $\geq 1 - \epsilon$;

$L(x) = 0 \quad \Rightarrow \quad$ M gives the answer 'no' with probability $\leq \epsilon$.

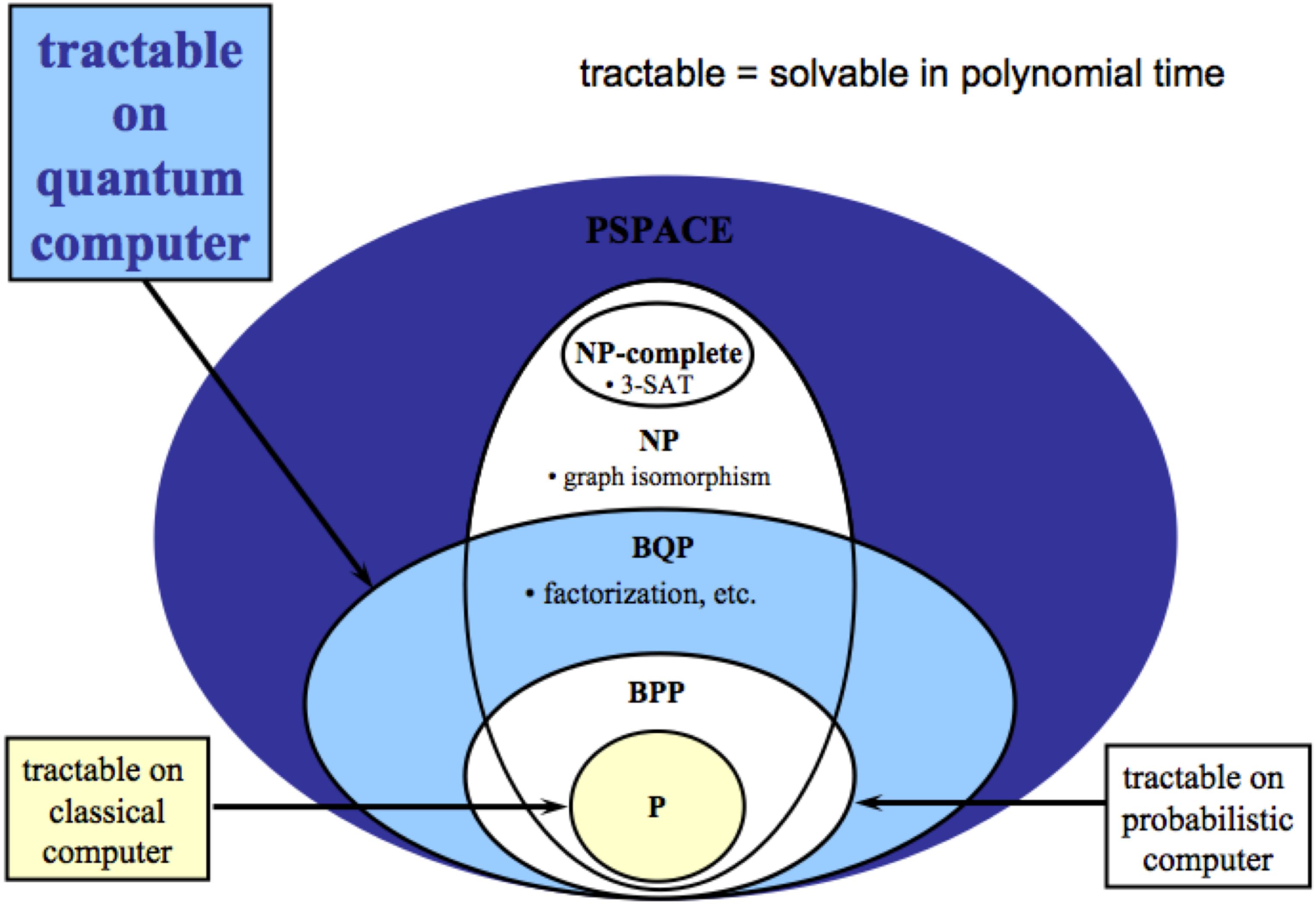
Alternatively using quantum circuit:

A *quantum algorithm* for the computation of a function $F : \mathbb{B}^* \to \mathbb{B}^*$ is a classical algorithm, that is, a deterministic Turing machine, that computes a function of the form $x \mapsto Z(x)$ where $Z(x)$ is a description of a *quantum circuit* which computes $F(x)$ on empty input.

The function F is said to belong to the class BQP if there is a quantum algorithm that computes F in time $poly(n)$.

P ⊆ BPP ⊆ BQP ⊆ PSPACE

tractable = solvable in polynomial time



CLASSICAL AND QUANTUM COMPUTATION

QUANTUM ALGORITHMS

Deutsch-Jozsa algorithm

It computes whether a Boolean function F over n variables is constant or balanced.
A Boolean function F over n variables is said to be

- *constant* if it gives the same output to all possible inputs;
- *balanced* if it outputs 0 for half of all possible inputs and 1 to the other half.

Examples:

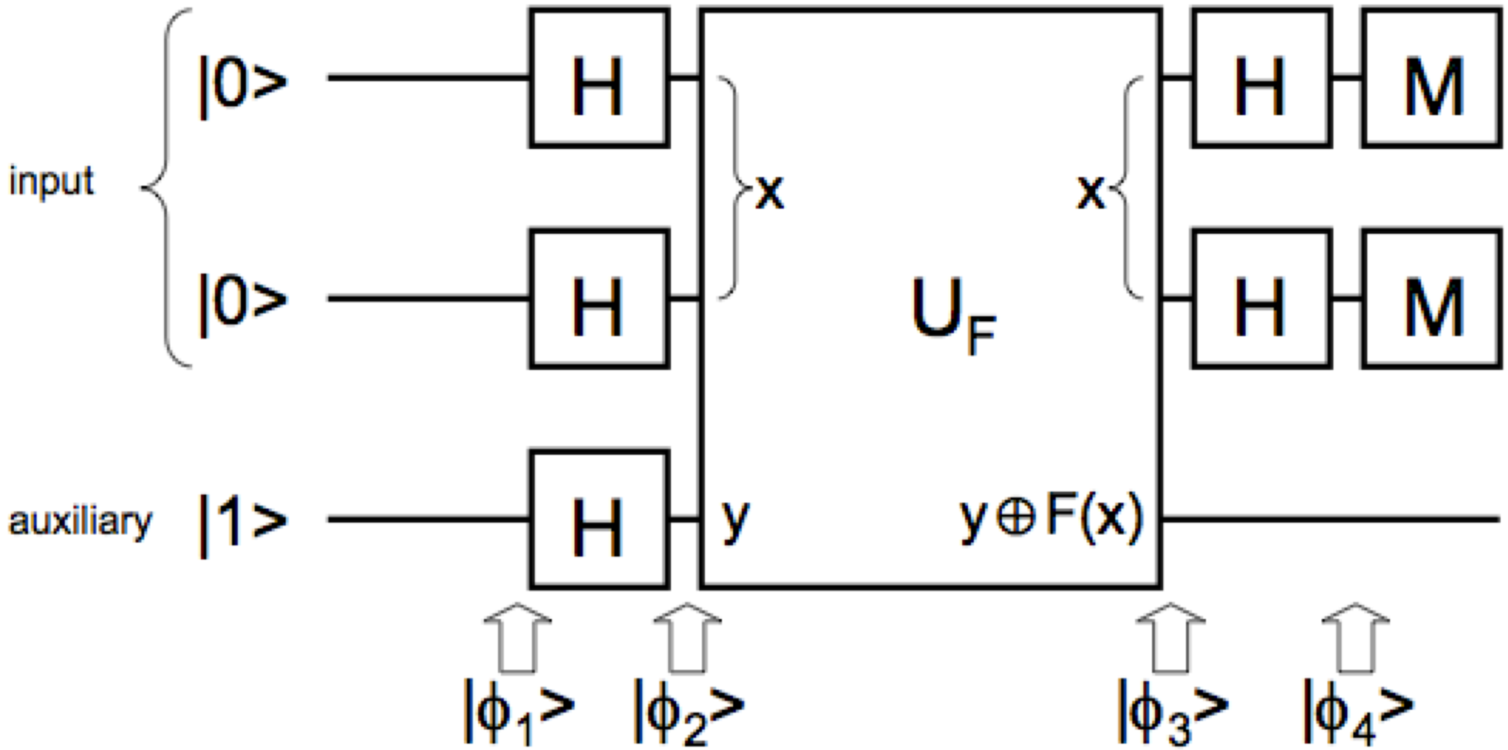
	Constant:		Balanced:					
x_1 x_2	$F(x_1,x_2)$		$F(x_1,x_2)$					
0 0	1	0	1	0	1	0	0	1
0 1	1	0	1	0	0	1	1	0
1 0	1	0	0	1	1	0	1	0
1 1	1	0	0	1	0	1	0	1

Classical complexity is exponential: in the worst case, the function needs to be applied $2^{n-1}+1$ times to check its output for more than a half of all inputs.

Deutsch-Jozsa algorithm for a two-qubit function

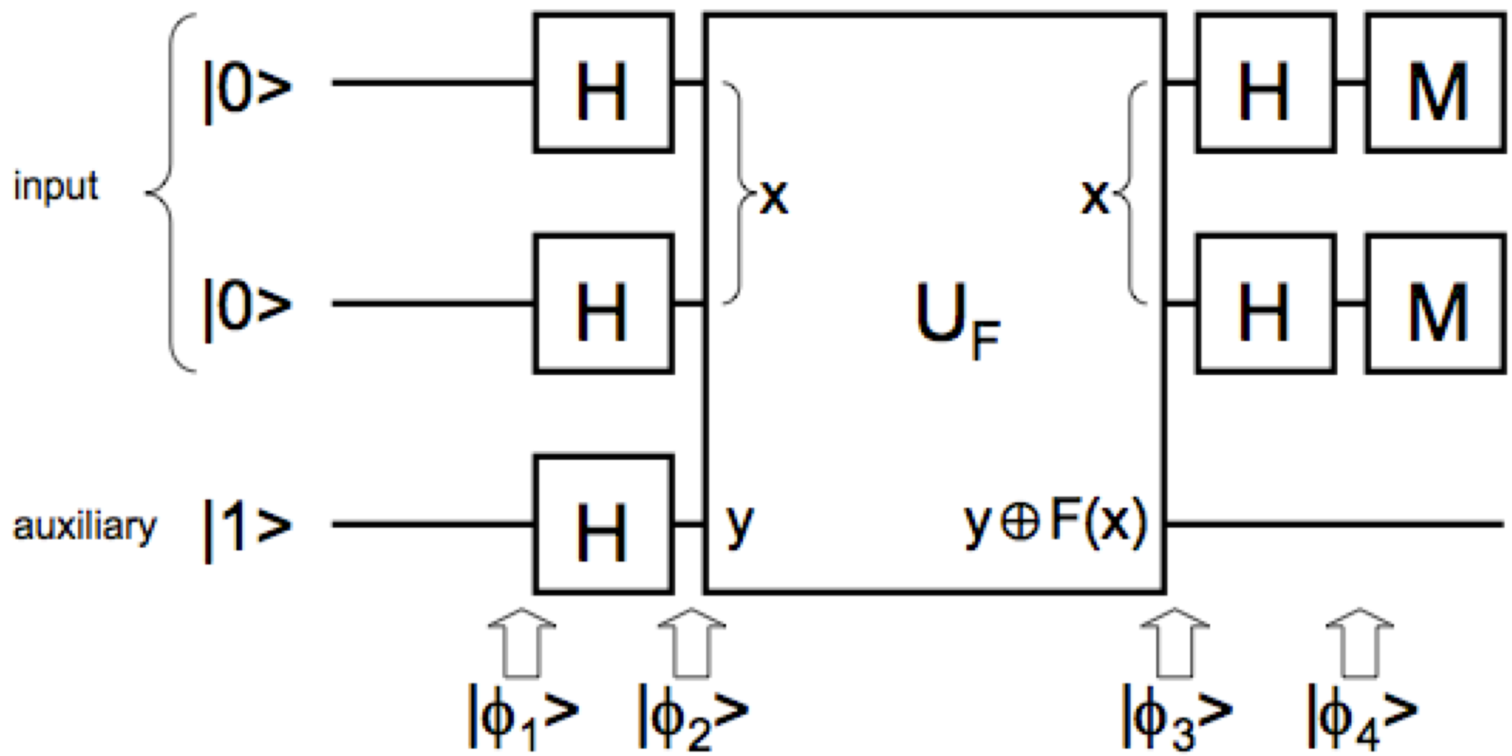
The initial state:

$$|\phi_1\rangle = |0\rangle \otimes |0\rangle \otimes |1\rangle = |001\rangle$$



Deutsch-Jozsa algorithm for a two-qubit function: Hadamard gates

$$
\begin{aligned}
|\phi_2\rangle &= \left(\hat{H} \otimes \hat{H} \otimes \hat{H}\right)(|0\rangle \otimes |0\rangle \otimes |1\rangle) = \hat{H}|0\rangle \otimes \hat{H}|0\rangle \otimes \hat{H}|1\rangle \\
&= \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] \otimes \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] \otimes \left[\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right] \\
&= \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes \left[\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right]
\end{aligned}
$$



x_1	x_2	$F(x_1,x_2)$
0	0	0
0	1	1
1	0	0
1	1	1

Deutsch-Jozsa algorithm for a two-qubit function: $\hat{U}_F$

$$
\begin{aligned}
|\phi_3\rangle &= \hat{U}_F\,|\phi_2\rangle = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right] \\
&= \hat{U}_F\left\{\frac{1}{2}|00\rangle \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right]\right\} + \hat{U}_F\left\{\frac{1}{2}|01\rangle \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right]\right\} \\
&\quad + \hat{U}_F\left\{\frac{1}{2}|10\rangle \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right]\right\} + \hat{U}_F\left\{\frac{1}{2}|11\rangle \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right]\right\} \\
&= \frac{1}{2}\Bigg(|00\rangle \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right] + |01\rangle \otimes \left[\frac{1}{\sqrt{2}}\left(|1\rangle - |0\rangle\right)\right] \\
&\quad + |10\rangle \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right] + |11\rangle \otimes \left[\frac{1}{\sqrt{2}}\left(|1\rangle - |0\rangle\right)\right]\Bigg) \\
&= \frac{1}{2}\left(|00\rangle - |01\rangle + |10\rangle - |11\rangle\right) \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right]
\end{aligned}
$$

Deutsch-Jozsa algorithm for a two-qubit function: readout

Now, we can disregard the auxiliary qubit and focus on the first factor of $|\phi_3\rangle$ above. We first rewrite it as

$$\frac{1}{2}\left(|00\rangle - |01\rangle + |10\rangle - |11\rangle\right) = \left[\frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)\right] \otimes \left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right]$$

and then perform the Hadamard rotations

$$\begin{aligned} |\phi_4\rangle &= \hat{H}\left[\frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)\right] \otimes \hat{H}\left[\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right] \\ &= |0\rangle \otimes |1\rangle = |01\rangle \end{aligned}$$

The measurement of each qubit reveals that the function is balanced.

The function is constant if the measurement of each input qubit at the end of the computation yields 0. Otherwise the function is balanced.

Deutsch-Jozsa algorithm

Inputs:

A black box $\hat{U}_F$ which performs the transformation $|\mathbf{x}\rangle|\mathbf{y}\rangle \rightarrow |\mathbf{x}\rangle|\mathbf{y} \oplus F(\mathbf{x})\rangle$ for $\mathbf{x} \in \{0, 1, \ldots, 2^{n-1}\}$ and $F(\mathbf{x}) \in \{0, 1\}$. It is promised that the function $F(\mathbf{x})$ is either constant or balanced.

Outputs:

0 iff F is constant.

Complexity/Runtime:

One evaluation of $\hat{U}_F$. Always succeeds.

Exponential speed-up compared to classical algorithm