

Quantum Computing

Francesco Testa

February 2025

Indice

1 Bit	2
1.1 NOT in bit come vettori	2
1.2 Swapping	2
1.3 Controlled NOT	3
2 Qubit	3
2.1 Assiomi della Quantum Theory	3
2.2 Misurazione	5
2.3 Preparazione stati	5
3 Parallelismo Quantico	7
3.1 Assicurare reversibilitá	7
3.2 Superposizioni	7
3.3 No cloning Theorem	8
4 II modulo	9
4.1 Intro	9
4.2 Complessità dei circuiti	9
5 Algoritmi Quantici	10
5.1 Deutsch problem	10
5.2 Bernstein-Vazirani problem	11
5.3 Simon problem	12
5.4 Shor's Algorithm	13
5.5 Grover's Algorithm	18
6 Protocolli quantistici	20
6.1 Quantum Teleportation	21
6.2 Quantum Pseudotelepathy	22
6.3 Quantum Key Distribution	23
6.4 Quantum Commitment	24
7 Error correction	25
7.1 QECCs	25
8 Quantum programming Languages	28
9 Quantum Program and System Verification	29
A Applicazione Matrici Unitarie	30
B Misurazione qubit	30

1 Bit

Il bit classico ha soltanto due stati: 0 o 1, e possiamo fare le classiche operazioni di Not, Or ecc.

Dati n bit possiamo rappresentare 2^n numeri.

Possiamo rappresentare le stringhe bit **come vettori**:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Questa rappresentazione ci permette di definire delle operazioni comuni tra bit e qubit, come quelle in seguito. Parliamo di **prodotto tensoriale** (\otimes) l'operazione che ci restituisce tutte le possibili combinazioni, ovvero tutti i possibili prodotti:

$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \otimes \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 y_0 \\ x_0 y_1 \\ x_1 y_0 \\ x_1 y_1 \end{bmatrix}$$

esempio: $|001\rangle = |00\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Rappresentiamo tutto come matrice, sia il bit sia le operazioni base sui bit. Nota: il prodotto tensoriale non è commutativo

1.1 NOT in bit come vettori

x può essere rappresentato come vettore colonna (ad esempio $\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$) Poiché rappresentiamo anche la NOT come una matrice, nel caso della due per due avremo:

$$NOT = X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Richiamando il prodotto tra matrici, vediamo come le proprietà della NOT sono rispettate, infatti se applichiamo la not a se stessa otteniamo l'identità:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

Notazione: $|\bar{x}\rangle = X|x\rangle$

$$\xrightarrow[X]{}$$

Andando a manipolare un singolo bit le uniche operazioni possibili sono l'identità (tenerlo com'è) oppure il NOT (fliparlo).

Le operazioni in quantum computing devono essere reversibili! Ovvero possiamo tornare indietro a partire dal risultato (ad esempio la AND tra due bit mi restituisce un bit solo e non sappiamo da quali bit siamo partiti, quindi è necessario portarsi indietro altra informazione)

1.2 Swapping

Un'operazione **reversibile** che possiamo fare in Quantum Computing, dati due bit, è lo **swapping**

$$S_{01}|x\rangle|y\rangle = |y\rangle|x\rangle$$

Se prendiamo questa operazione come matrice:

$$S_{01} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Nota: quando mettiamo due vettori vicini si intende un'operazione di prodotto tensoriale tra di loro.

1.3 Controlled NOT

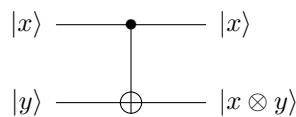
É un'operazione classica che esegue la stessa operazione della XOR, ma la rende **biettiva**(e self-invertible) utilizzando un nuovo filo che controlla l'operazione. Nell'output infatti compare uno dei due input (x) senza modificarlo. Infatti posso riottenere y a partire da x e dal risultato.¹

$$CNOT|x\rangle|y\rangle = |x\rangle|x \otimes y\rangle$$

Se vediamo l'operazione come matrice, abbiamo:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Intuizione: abbiamo una not in alto a sinistra e invertiamo in basso a destra in base al risultato della not. Il circuito del CNOT è il seguente:



2 Qubit

Le operazioni viste fino ad ora (NOT, swap, CNOT) sono delle operazioni che possono essere svolte anche sui circuiti classici (quindi con i bit) e danno parte di un particolare ramo, ovvero i **circuiti reversibili**². Queste porte sono anche **self-inverse**, ovvero applicandole a se stesse si ottiene l'identità. In realtà, per la reversibilità basta che sia invertibile, ovvero è importante che l'inverso esista.

Questi operatori, in quanto reversibili, possono essere utilizzate anche nei circuiti quantici. In effetti, il concetto di *Qubit* è un'estensione del concetto di bit.

Definizione. Un *qubit* è un sistema quantico il cui stato è un vettore **bidimensionale, unitario, complesso**.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \text{ dove } |\alpha|^2 + |\beta|^2 = 1$$

- Bidimensionale: è una combinazione lineare della base (composta da due elementi indipendenti: $|0\rangle$ e $|1\rangle$)³.
- Complesso: α e β sono numeri complessi ($\alpha = a + ib$). Quindi in realtà siamo in un campo bidimensionale complesso, che corrisponde ad un campo reale a quattro dimensioni.
- Unitario: la lunghezza del vettore è sempre 1. Per misurarlo, possiamo applicare il teorema di Pitagora applicato ai numeri complessi, ovvero: $|\alpha| = \sqrt{a^2 + b^2}$, che, poiché deve essere sempre 1, equivale a $|\alpha| = a^2 + b^2$

In realtà, il bit classico è un qubit che soddisfa $\alpha = 1$ o $\beta = 1$, ma i processi fisici non ci permettono di vedere stati intermedi. Questo infatti già è il primo scoglio del quantum computing, ovvero è difficile creare delle condizioni favorevoli per il quantum (ad esempio temperature molto basse). I dettagli a basso livello non ci interessano, dunque partiamo da un livello di astrazione che ci permette di usare i qubit, senza sapere come essi siano creati.

2.1 Assiomi della Quantum Theory

Axioma 2.1 (stati). Lo stato di un sistema quantico è un vettore complesso unitario:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \text{ dove } |\alpha|^2 + |\beta|^2 = 1$$

¹Non posso fare lo stesso ragionamento con la AND perché se volessi capire da dove proviene lo 0 del risultato ho 3 possibilità, è necessario avere più bit per coprire tutte le combinazioni, a differenza della XOR

²Ogni porta ha n input e n output, e dall'output possiamo risalire all'input.

³La base di uno spazio vettoriale è un insieme di vettori linearmente indipendenti che generano lo spazio.

α e β sono chiamati **ampixezza** (amplitudes).

Ecco alcuni esempi di **stati semplici**:

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \quad |-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

Entrambi gli stati denotano un momento in cui è "50% 0, 50% 1", tuttavia, effettuando delle operazioni, questi due stati si comportano in modo diverso. Non possiamo quindi sostituire lo stato con la probabilità per effettuare delle operazioni.

Axioma 2.2 (dinamica). *L'evoluzione di un sistema chiuso è descritto da una matrice unitaria U .*

$$|\psi\rangle \xrightarrow{\boxed{U}} U|\psi\rangle$$

Si parla di matrice unitaria in quanto, nella meccanica quantistica, si lavora con vettori unitari. Dunque le matrici che utilizziamo per effettuare delle operazioni sui vettori devono dare in output un vettore unitario, per questo chiamate matrice unitarie.

Teorema 2.1 (Matrice Unitaria). *Una matrice U si dice **unitaria** $\iff U^\dagger U = I$, dove U^\dagger è la matrice coniugata trasposta di U*

Data una matrice U , andiamo a trasporla (swap di righe e colonne) e coniugarla (per ogni elemento, andiamo ad invertire il segno della parte complessa, ovvero $3 + 2i$ diventa $3 - 2i$) per ottenere U^\dagger .

Esistono 4 matrici unitarie molto importanti che sono: Identità e le 3 matrici di Pauli (che ci permettono di ruotare gli assi). Queste sono:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

(esercizi: Appendice A).

Un'altra matrice unitaria importante è quella di **Hadamard**:

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

È utile perché $H|0\rangle = |+\rangle$ e $H|1\rangle = |-\rangle$, ovvero ci permette di creare **superposizioni**.

Axioma 2.3 (Misurazione - Regola di Born). *Possiamo misurare un sistema in una qualsiasi base dello spazio di stati, e otteniamo un risultato probabilistico. Tuttavia, il sistema poi collassa sul risultato ottenuto*

Ad esempio, preso il qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, possiamo misurarlo nella base $\{|0\rangle, |1\rangle\}$ (chiamata anche **base computazionale**). Otterremo un risultato $x \in [0, 1]$ con probabilità $|\alpha_x|^2$, e il sistema collasserà a $|x\rangle$ (esercizi: Appendice B).⁴

Il disegno del circuito è il seguente:

$$|\psi\rangle \xrightarrow{\boxed{\text{ }}|x\rangle}$$

Axioma 2.4 (Composizione di sistemi). se

- *A ha uno stato in $\text{span}(V)$ ⁵ per un insieme di vettori V ;*
- *B ha uno stato in $\text{span}(W)$ per un insieme di vettori W ;*

allora AB ha uno stato nello $\text{span}(\{v \otimes w | v \in V \wedge w \in W\})$

La definizione può sembrare ostica ma in realtà è molto semplice. Considerando due qubit che hanno base $\{|0\rangle, |1\rangle\}$, allora la base della composizione dei due qubit sarà $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$

⁴In seguito utilizzeremo anche la misurazione basata sul coniugato. Ovvero, per misurare $|A_y|^2$ la formula sarà $A_y^\dagger A_y$

⁵Span è il sottospazio generato da un insieme di vettori, ovvero l'insieme finito di tutte le possibili combinazioni lineari degli elementi di V

2.2 Misurazione

La regola di Born, ovvero l'assioma 2.3 esprime la misurazione di un singolo Qubit. Supponiamo ad esempio di avere un sistema composto da piú qubit, come possiamo misurarne ad esempio solo uno del sistema? Innanzitutto riscriviamo la formula di uno sistema ad n qubit:

$$|\psi\rangle = \alpha_0|0\rangle \otimes |\psi_0\rangle + \alpha_1|1\rangle \otimes |\psi_1\rangle$$

$|\psi_i\rangle$ sono gli $n-1$ qubit rimanenti del sistema.

Axioma 2.5 (Regola di Born generalizzata). *Possiamo misurare il **primo** qubit in un sistema ad n qubit. $|\psi\rangle = \alpha_0|0\rangle \otimes |\psi_0\rangle + \alpha_1|1\rangle \otimes |\psi_1\rangle$ nella base $\{|0\rangle, |1\rangle\}$. Il risultato sarà x con probabilità $|\alpha_x|^2$. Lo stato del sistema collassa a $|x\rangle \otimes |\psi_x\rangle$.*⁶

Gli esercizi in appendice B (il 2 e il 3) valgono da esempi.

2.3 Preparazione stati

2.3.1 Stato formato da un qubit

L'obiettivo é preparare un qualsiasi stato $|\psi\rangle$ a partire dallo stato $|0\rangle$ (o $|1\rangle$). Ovvero, vogliamo trovare quella matrice unitaria U t.c. $|\psi\rangle = U|0\rangle$. Per preparare uno stato $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ a partire da voglio che:

$$U|0\rangle = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \implies U = \begin{bmatrix} \alpha & x \\ \beta & y \end{bmatrix} \text{ per qualche } x \text{ e } y$$

In realtà x e y non possono essere prese in modo casuale in quanto **U deve essere unitaria**. Si puó dimostrare che la scelta di U é la seguente:

$$U = \begin{bmatrix} \alpha & -\beta^* \\ \beta & \alpha^* \end{bmatrix} \text{ infatti, } UU^\dagger = \begin{bmatrix} \alpha & -\beta^* \\ \beta & \alpha^* \end{bmatrix} \begin{bmatrix} \alpha & -\beta^* \\ \beta & \alpha^* \end{bmatrix}^\dagger = \begin{bmatrix} \alpha\alpha^* + \beta^*\beta & \alpha\beta^* - \beta^*\alpha \\ \beta\alpha^* - \alpha^*\beta & \beta^*\beta + \alpha^*\alpha \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$\alpha\beta^* - \beta^*\alpha = 0$ perché il prodotto complesso é commutativo; $\beta^*\beta + \alpha^*\alpha = 1$ poiché é il modulo del numero complesso.

2.3.2 Stato $|0\rangle$

Siamo riusciti a preparare uno stato a partire dal qubit $|0\rangle$. Come possiamo ottenere $|0\rangle$? Ci sono due metodi:

- Processi fisici
- Partiamo da uno stato qualsiasi e lo misuriamo per ottenere $|0\rangle$ o $|1\rangle$. In questo caso andiamo a negarlo.

2.3.3 Stato di due qubit arbitrari

Ora vogliamo preparare qualsiasi stato $|\psi\rangle$ fatto da due qubit, a partire da degli zeri ($|0\rangle$). Lo stato che vogliamo raggiungere ha forma: $|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$.

Possiamo **comporre stati** grazie al tensor product dei vettori.

Qubit indipendenti Per ottenere dei qubit indipendenti possiamo utilizzare:

$$U|0\rangle \otimes V|0\rangle = (\alpha_1|0\rangle + \alpha_1|1\rangle) \otimes (\beta_1|0\rangle + \beta_1|1\rangle) = \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle$$

Quindi

$$\alpha_{00} = \alpha_0\beta_0 \quad \alpha_{01} = \alpha_0\beta_1 \quad \alpha_{10} = \alpha_1\beta_0 \quad \alpha_{11} = \alpha_1\beta_1 \implies \frac{\alpha_{00}}{\alpha_{01}} = \frac{\beta_0}{\beta_1} = \frac{\alpha_{10}}{\alpha_{11}}$$

Quest'ultima uguaglianza **non é sempre vera**, o almeno nel caso generale non lo é.

⁶Nota: considera bene che x puó essere soltanto 0 o 1. Quindi in base alla misurazione del primo bit cambia totalmente lo stato del sistema!

Qubit entangled Vogliamo ottenere $|\sigma\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$. Raccogliendo ($|0\rangle$ e $|1\rangle$) abbiamo:

$$|\sigma\rangle = |0\rangle \otimes |\psi\rangle + |1\rangle \otimes |\phi\rangle \quad \text{con} \quad |\psi\rangle = \alpha_{00}|0\rangle + \alpha_{01}|1\rangle \quad \text{e} \quad |\phi\rangle = \alpha_{10}|0\rangle + \alpha_{11}|1\rangle$$

Vogliamo ora trovare quella U (da applicare soltanto sul primo qubit) t.c.

$$U \otimes I |\sigma\rangle = |0\rangle \otimes |\psi'\rangle + |1\rangle \otimes |\phi'\rangle$$

con $|\psi'\rangle$ e $|\phi'\rangle$ ortogonali⁷.

Proviamo con la matrice U di prima:

$$U \otimes I |\sigma\rangle = \begin{bmatrix} a & -b^* \\ b & a^* \end{bmatrix} \otimes I |\sigma\rangle = (a|0\rangle + b|1\rangle) \otimes |\psi\rangle + (-b^*|0\rangle + a^*|1\rangle) \otimes |\phi\rangle = |0\rangle \otimes |\psi'\rangle + |1\rangle \otimes |\phi'\rangle$$

$$\text{dove} \quad |\psi'\rangle = a|\psi\rangle - b^*|\phi\rangle \quad \text{e} \quad |\phi'\rangle = b|\psi\rangle a^*|\phi\rangle$$

Sostanzialmente stiamo applicando la matrice U soltanto ad un qubit, lasciando inalterato il secondo. Poiché vogliamo che i due vettori siano ortogonali, deve valere:

$$0 = \langle \phi' | \psi' \rangle = b^*a\langle \psi | \psi \rangle + aa\langle \phi | \psi \rangle - b^*b^*\langle \psi | \phi \rangle - ab^*\langle \phi | \phi \rangle = a^2\langle \phi | \psi \rangle - b^{*2}\langle \psi | \phi \rangle + ab^*(\langle \psi | \psi \rangle - \langle \phi | \phi \rangle) = a^2\langle \phi | \psi \rangle - b^{*2}\langle \psi | \phi \rangle$$

Possiamo dunque risolvere l'equazione per derivare i valori di a e b , che sono i componenti della matrice U . In questo modo abbiamo dunque trovato la matrice U .

Nota: NORMALIZZAZIONE I vettori possono non essere normalizzati, ovvero possono non essere unitari. Quindi per renderli unitari possiamo normalizzare, dividendo i vettori per il loro modulo:

$$|\psi''\rangle = \frac{|\psi'\rangle}{\lambda} \quad |\phi''\rangle = \frac{|\phi'\rangle}{\mu}$$

Non perdono la proprietá dell'ortogonalitá.

2.3.4 Composizione circuiti

Se per comporre degli stati usiamo il tensor product, possiamo usarlo anche tra matrici per **comporre i circuiti**.

Tensor Product di matrici Definiamo il Tensor Product tra matrici. Quello che vogliamo é:

$$(U \otimes V)(|\phi\rangle \otimes |\psi\rangle) = U|\phi\rangle \otimes V|\psi\rangle$$

Definizione (Tensor product tra matrici). Date due matrici:

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \end{bmatrix} \quad V = \begin{bmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{bmatrix}$$

$$U \otimes V = \begin{bmatrix} u_{1,1} \begin{bmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{bmatrix} & u_{1,2} \begin{bmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{bmatrix} \\ u_{2,1} \begin{bmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{bmatrix} & u_{2,2} \begin{bmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} u_{1,1}v_{1,1} & u_{1,1}v_{1,2} & u_{1,2}v_{1,1} & u_{1,2}v_{1,2} \\ u_{1,1}v_{2,1} & u_{1,1}v_{2,2} & u_{1,2}v_{2,1} & u_{1,2}v_{2,2} \\ u_{2,1}v_{1,1} & u_{2,1}v_{1,2} & u_{2,2}v_{1,1} & u_{2,2}v_{1,2} \\ u_{2,1}v_{2,1} & u_{2,1}v_{2,2} & u_{2,2}v_{2,1} & u_{2,2}v_{2,2} \end{bmatrix}$$

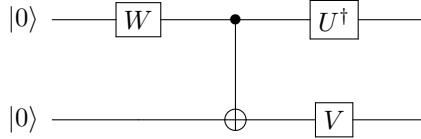
Costruzione circuito Preso V unitario t.c. $|\psi''\rangle = V|0\rangle, |\phi''\rangle = V|1\rangle$ abbiamo:

$$\begin{aligned} U \otimes I |\sigma\rangle &= |0\rangle \otimes |\psi'\rangle + |1\rangle \otimes |\phi'\rangle = |0\rangle \otimes \lambda|\psi''\rangle + |1\rangle \otimes \mu|\phi''\rangle = \\ &= |0\rangle \otimes \lambda V|0\rangle + |1\rangle \otimes \mu V|1\rangle = (I \otimes V)(\lambda|0\rangle \otimes |0\rangle + \mu|1\rangle \otimes |1\rangle) \\ \implies |\sigma\rangle &= (U^\dagger \otimes V)(\lambda|0\rangle \otimes |0\rangle + \mu|1\rangle \otimes |1\rangle) \\ &= (U^\dagger \otimes V)CNOT((\lambda|0\rangle + \mu|1\rangle) \otimes |0\rangle) \\ &= (U^\dagger \otimes V)CNOT((W|0\rangle) \otimes |0\rangle) \end{aligned}$$

Per qualche W unitaria.

Quindi il circuito che mi prepara lo stato $|\sigma\rangle$ a partire da due qubit $|0\rangle$ é:

⁷Due vettori $|\psi\rangle, |\phi\rangle$ si dicono **ortogonali** $\iff \langle \psi | \phi \rangle = \sum_{i=1}^n \psi_i^* \phi_i = 0$. $\langle \psi | \phi \rangle = \langle \phi | \psi \rangle^*$. L'operazione $\langle \cdot | \cdot \rangle$ é chiamata **inner product**



Dunque porte come la **CNOT** sono **fondamentali per creare entanglement**, soltanto attraverso queste porte riusciamo.

Nell'esempio specifico di prima, sceglieremo di usare la cnot raccogliere (entangle) gli $|0\rangle$ quando abbiamo $(|0\rangle|0\rangle|1\rangle|1\rangle)$.

3 Parallelismo Quantico

Nel computing classico siamo abituati ovviamente a scrivere funzioni e circuiti che dato un x producono un certo $f(x)$. Per trattare numeri,abbiamo bisogno di codificarli. Una codifica ad esempio potrebbe essere quella per gli *unsigned int*, in cui semplicemente si codifica il numero in base 2^8 . Ogni intero è quindi rappresentato da una stringa di k bit.

Se passiamo al quantum computing, e volessimo scrivere funzioni classiche, sembra non cambiare troppo. Infatti partiamo da un x per produrre un $f(x)$. Possiamo prendere un intero e codificarlo nello stesso modo. (Potremmo poi applicare quelle che sono le operazioni standard a $|1\rangle$ e $|0\rangle$ per ottenere dei risultati simili a quelli del computing classico). Ogni intero è quindi rappresentato dal corrispondente stato della base computazione di k qubits. La limitazione ovviamente arriva dalla necessità ei **reversibilità** del quantum.

3.1 Assicurare reversibilità

Per ovviare al problema, é possibile rendere reversibili le porte. Questo si puó fare banalmente andando a computare $f(x)$ su fili nuovi che partono da 0, e lasciando invariati i cavi di x , come é possibile vedere in figura 1. Per definizione dunque $U_f|x\rangle_n \otimes |0\rangle_m = |x\rangle_n|f(x)\rangle_m$.

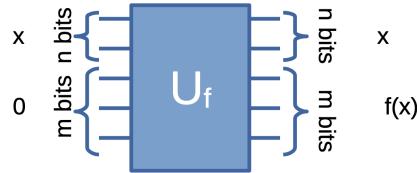


Figura 1: Assicurare la reversibilitá. In questo modo noi definiamo la trasformazione U_f come reversibile e unitaria.

Esiste un caso generalizzato. Se avessimo un circuito in cui abbiamo dei fili che non ci servono piú, oppure non abbiamo 0 disponibili, possiamo usare lo XOR(bitwise) come si vede in figura 2. Utilizzando le proprietà dello xor, ovviamente possiamo tornare y riapplicando $f(x)$.

Nota: con $y = 0$ ci riduciamo a figura 1;

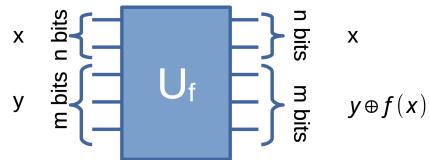


Figura 2: Generalizzazione di figura 1.

Nota: la f che stiamo considerando per ora é una f arbitraria classica, ovvero che agisce soltanto su $|1\rangle$ e $|0\rangle$.

3.2 Superposizioni

Quello che vogliamo fare ora é computare la funzione f nello stesso momento su valori diversi. Le **superposizioni** (superposition) sono un elemento chiave per cercare di raggiungere questo obiettivo.

⁸ovviamente con k bit abbiamo 2^k numeri.

Per costruire una superposizione⁹ sfruttiamo l’Hadamard.

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

Otteniamo in questo modo una **superposizione uniforme** su tutti i possibili valori.

Possiamo ovviamente **generalizzare** ad n qubit, utilizzando n Hadamards:

$$H^{\otimes n} |0\rangle^n = \frac{1}{2^{n/2}} \sum_{0 \leq x < 2^n} |x\rangle_n$$

dove x sono gli interi rappresentati in unsigned int, la n sotto il ket indica ”stringhe di lunghezza n ”.

La domanda é: **perché abbiamo bisogno di questo?**

L’idea é la seguente: a partire da una funzione classica f e cerchiamo di eseguirla **nello stesso tempo su tutti i possibili input**. In questo modo, con una sola esecuzione posso sapere tutto sulla mia funzione¹⁰.

Ovviamente non é cosí semplice ma questo é il primo passo per raggiungere questo risultato. Cerchiamo quindi di costruire il circuito in cui siamo ora:

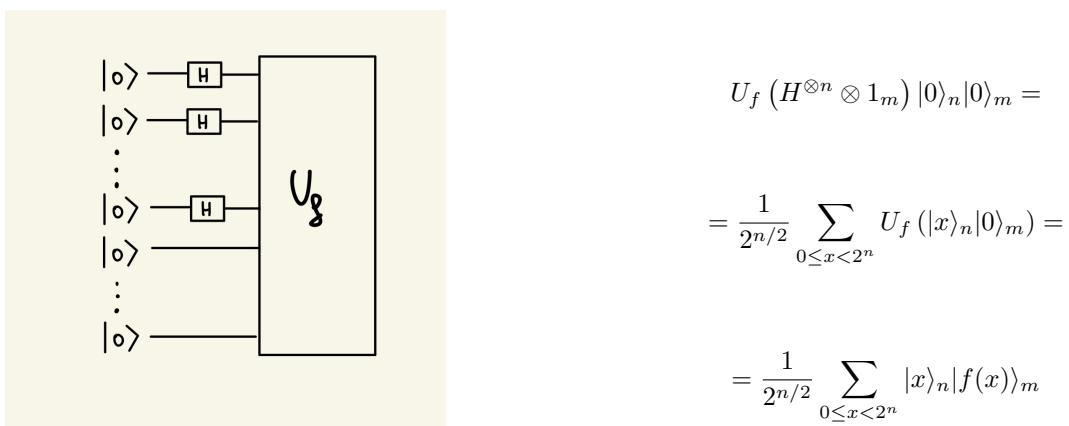


Figura 3: Primo tentativo di costruzione del circuito per realizzare parallelismo.

dove 1_m é l’identitá. Serve per far restare 0 gli altri cavi che non partono in superposizione.

Questo è anche detto **Parallelismo quantico**, ovvero possiamo computare la funzione f su tutti i possibili input contemporaneamente. Tuttavia ci sono dei problemi: il risultato è nello stato finale, ovvero in una **superposizione**. Ovviamente questa superposizione deve essere misurata affinché possiamo capire cosa ci sia, ed in questo momento avviene il collasso ad un singolo valore (dunque otterremo il valore di $f(x)$ per un solo input randomico). Una possibile soluzione potrebbe essere quella di clonare il risultato e misurarla diverse volte. Tuttavia ...

3.3 No cloning Theorem

Non é possibile copiare un qubit **arbitrario**. Non esiste nemmeno una clonazione **approssimativa**, semplicemente non si può fare.

Teorema 3.1 (No cloning Theorem). *Non esiste alcuna trasformazione unitaria U che prende in input lo stato $|y\rangle_n |0\rangle_n$ e come output $|y\rangle_n |y\rangle_n$ per ogni y .*

Dimostrazione. É una conseguenza della linearitá. Dimostriamo per assurdo. Ipotizziamo che esista, ovvero $\exists U$ t.c. $U|y\rangle|0\rangle = |y\rangle|y\rangle$. Abbiamo dunque (per linearitá¹¹) che $U((a|y\rangle + b|x\rangle)|0\rangle) = aU(|y\rangle|0\rangle) + bU(|x\rangle|0\rangle) = a|y\rangle|y\rangle + b|x\rangle|x\rangle$. Tuttavia, poiché posso clonare qualsiasi stato, posso utilizzare direttamente U : $U((a|y\rangle + b|x\rangle)|0\rangle) = (a|y\rangle + b|x\rangle)(a|y\rangle + b|x\rangle) = a^2|y\rangle|y\rangle + b^2|x\rangle|x\rangle + ab|y\rangle|x\rangle + ab|x\rangle|y\rangle$. Gli unici casi in cui i risultati sono uguali sono o $a = 0$ o $b = 0$. Questo significa che non posso copiare qubit arbitrari. \square

Teorema 3.2 (No approximate cloning theorem). *Non esiste alcuna trasformazione unitaria U che prende in input lo stato $|y\rangle_n |0\rangle_n$ e approssima l’output $|y\rangle_n |y\rangle_n$ per ogni y .*

⁹Superposizione é diverso da entangled. La prima indica che contemporaneamente un qubit é 0 o 1.

¹⁰Nel computing classico devo per forza usare x per trovare $f(x)$. In quantum noi puntiamo a comprendere l’andamento di tutta la funzione f in una singola esecuzione.

¹¹posso distribuire U

Dimostrazione. Dimostriamo per assurdo. Supponiamo che esista U t.c.: $U|y\rangle|0\rangle \sim |y\rangle|y\rangle$ e $U|x\rangle|0\rangle \sim |x\rangle|x\rangle$. Ricordando che: 1. $\langle\psi_1 \otimes \psi_2|\phi_1 \otimes \phi_2\rangle = \langle\psi_1|\phi_1\rangle\langle\psi_2|\phi_2\rangle$, 2. $\langle\psi|\phi\rangle = \langle U\psi|U\phi\rangle$, 3. $\langle\psi|\psi\rangle = 1$:

$$\begin{aligned}\langle y0|x0\rangle &\sim \langle yy|xx\rangle \stackrel{\text{per } 1}{\implies} \\ \langle y|x\rangle\langle 0|0\rangle &\sim \langle y|x\rangle\langle y|x\rangle \stackrel{\text{per } 3}{\implies} \\ \langle y|x\rangle &\sim \langle y|x\rangle^2\end{aligned}$$

Questo vale solo se l'inner product è molto vicino allo 0 (ortogonale) oppure a 1 (uguale). Questo significa che se riesci a copiare qualcosa (ad esempio y), allora puoi copiare soltanto altri stati molto vicini a y o ortogonali a y (sempre con un piccolo errore). Altrimenti l'errore è alto. \square

Cosa possiamo fare quindi? Se misuro uno superstato non mi allontano troppo da ciò che riesco a fare con la computazione classica.

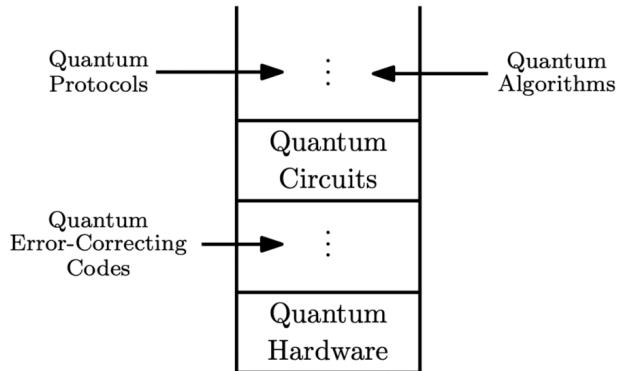
Se provassi a copiarlo (ad esempio per ripetere la misurazione più volte) fallirei.

Dobbiamo quindi non misurare direttamente ma lavorare ancora di più sui qubit per misurare NON $f(0)$ o $f(1)$ ma per misurare una combinazione di f su valori diversi. In realtà non scopriamo tutti i valori ma possiamo estrarre delle informazioni riguardo le relazioni tra i valori di f (esempio: periodicità di una funzione).

Arriviamo quindi alle applicazioni.

4 II modulo

4.1 Intro



Nel secondo modulo vedremo:

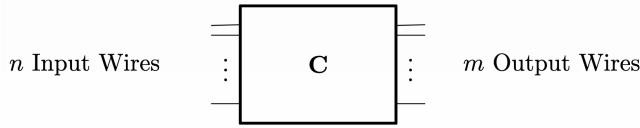
- Algoritmi Quantici: focus principale sull'efficienza, unico motivo di interesse nel quantum;
- Protocolli quantici: comunicazione e protocolli di sicurezza (esempio: Quantum Telepaty);
- Codici per la correzione di errori quantici (**decoherence**): per ora soltanto astrazione. Attualmente riusciamo implementare fisicamente, tenendo basso l'errore, soltanto algoritmi che impiegano meno di 20 qubits.

4.2 Complessità dei circuiti

Parliamo di:

- **Task:** problema, formulato spesso con simboli matematici. Siamo interessati in task formulati con funzioni matematiche:
 - **Booleane:** $0, 1^n \rightarrow 0, 1^m$ che indichiamo con \mathcal{F}_m^n
 - **Parametriche Booleane:** $\mathcal{F}_q^p \rightarrow \mathcal{F}_m^n$ **funzionali** (funzioni di funzioni). Ad esempio, preso $p = q = 2, n = 0$ $m = 1$, il task di controllare se la funzione in input ritorna 00 su tutti i suoi possibili inputs.
- **Process:** artefatto, in questo caso il **circuito**, l'algoritmo che deve essere eseguito

La relazione è che Process $\xrightarrow{\text{solves}}$ Task.



- Un **processo classico** sarà un **circuito booleano** \mathcal{C} . Computa una funzione booleana $f_C \in \mathcal{F}_m^n$ t.c. $f_C(x) = y$ precisamente quando \mathcal{C} eseguendo x sugli n fili input produce y sugli m fili in output.

Questo può essere generalizzato per le booleane parametriche: necessitiamo di un gate che permette di fare la "chiamata di funzione" su tot input.

La grandezza (*size*) di \mathcal{C} è una **misura** di quante risorse consumerà il circuito \mathcal{C} quando sarà eseguito. La grandezza di \mathcal{C} (indicata con $|\mathcal{C}|$) può essere misurata in diversi modi: **width**, ovvero il massimo numero di bit necessari nella computazione (nota: se abbiamo n bit in input, per la computazione potrebbero servire anche più di n bit); **depth**, ovvero la lunghezza massima tra i path da output a input (si può usare come una sorta di riferimento temporale); **numero di gate**, che rappresenta un upperbound di depth e width.

Se parliamo di funzioni parametriche booleane, un interessante parametro è il **numero di chiamate** del gate speciale che computa la funzione parametrica (che indichiamo con $|\mathcal{C}|_P$).

- Un **processo quantico** è semplicemente un **circuito quantico**: Possiamo generalizzare quanto detto per



i circuiti classici, considerando che:

- i circuiti quantici hanno un comportamento probabilistico. Questo implica che un circuito *computa* una funzione $f_Q \in \mathcal{F}_m^n$ se mantiene bassa la probabilità di errore
- tralasciando la misurazione, il circuito Q deve essere **reversibile**. Quindi dobbiamo supporre che un circuito che computare $f \in \mathcal{F}_m^n$ deve avere $k = n+m+r$ input (input + output + altro) e output t.c., con alta probabilità, su input $|x\rangle \otimes |y\rangle \otimes |0^{\otimes r}\rangle$ produrrà $|x\rangle \otimes |f(x) \oplus y\rangle \otimes |\psi\rangle$, dove $|0^{\otimes r}\rangle$ rappresenta qubit ausiliari necessari nella computazione e ψ rappresenta garbage

Ora ci chiediamo: esiste qualche funziona parametrica f t.c. ci sia un circuito quantico Q che computa f t.c. tutti i circuiti booleani classici \mathcal{C} che calcolano f lo fanno con $|\mathcal{Q}| < |\mathcal{C}|$?¹² Spoiler: sì. Ora ne analizzeremo alcuni (a partire da cose inutili a cose utili).

Ciò che dobbiamo fare ora però è **uniformare** la teoria della complessità per poter confrontare i circuiti classici con quelli quantici (ovvero dobbiamo mappare le turing machines).

Nella computazione classica, una *circuit family* è una famiglia $\{\mathcal{C}_n\}_{n \in N}$ t.c. $\forall n \in N$, il circuito \mathcal{C}_n computa una funzione in \mathcal{F}_1^n . Questa famiglia di circuiti, può essere vista anche come una funzione: $f_{\{\mathcal{C}_n\}} : \{0,1\}^* \rightarrow \{0,1\}$ (problemi decisionali). La classe dei linguaggi ($\subseteq \{0,1\}^*$) che può essere computata dalla famiglia dei circuiti classici che hanno un bound polinomiale e sono generati in tempo polinomiale attraverso algoritmi è **precisamente \mathcal{P}** .

Possiamo fare un discorso simile per i circuiti quantici: se consideriamo la famiglia dei circuiti *quantici* di grandezza polinomiale generati in tempo polinomiale *classico* otteniamo una classe chiamata BQP , ovvero Bounded Quantum P (bounded è per gli errori).

Se consideriamo la macchina di Turing, la funzione di transizione può essere simulata da un circuito perché È un circuito. Esistono anche le QUantum Turing Machine ma in questo ambito si lavora con i circuiti.

5 Algoritmi Quantici

5.1 Deutsch problem

Nel Deutsch Problem, siamo interessati nel determinare se una funzione parametrica $f \in \mathcal{F}_1^1$ è **costante** (i.e. $f(0) = f(1)$).

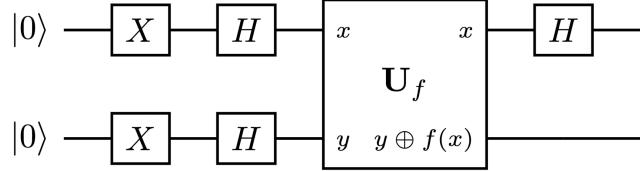
Nella computazione classica, dobbiamo per forza invocare f **due volte**, altrimenti non potremmo mai saperlo. In quantum riusciamo a farlo invocando la funzione soltanto una volta.

¹²Sostanzialmente ci chiediamo se esiste una funzione che riusciamo a calcolare con meno risorse usando un circuito quantico rispetto ad uno classico.

Partiamo assumendo che esista un gate \mathcal{U}_f che computa f senza utilizzare altri qubit ausiliari:

$$\mathcal{U}_f(|x\rangle|y\rangle) = |x\rangle|y \oplus f(x)\rangle$$

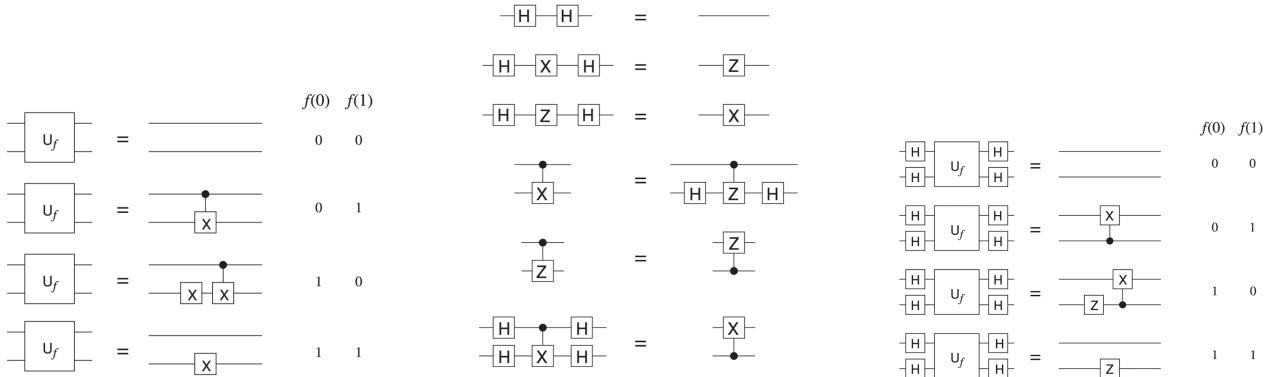
Il circuito che risolve il problema di Deutsch è:



Possiamo verificarne la correttezza:

$$\begin{aligned} & (\mathbf{H} \otimes \mathbf{1}) \mathbf{U}_f (\mathbf{H} \otimes \mathbf{H}) (\mathbf{X} \otimes \mathbf{X}) (|0\rangle|0\rangle) \\ &= \begin{cases} |1\rangle \frac{1}{\sqrt{2}} (|f(0)\rangle - |\tilde{f}(0)\rangle), & f(0) = f(1), \\ |0\rangle \frac{1}{\sqrt{2}} (|f(0)\rangle - |\tilde{f}(0)\rangle), & f(0) \neq f(1). \end{cases} \end{aligned}$$

Se analizzassimo l'interpretazione equazionale (abbiamo 4 possibili funzioni da 2 qubit a 2 qubit):

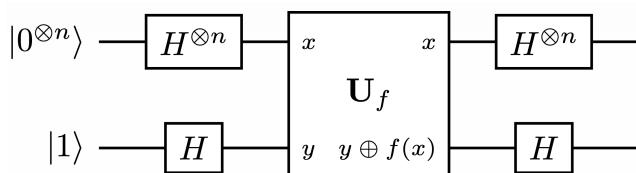


5.2 Bernstein-Vazirani problem

Dati $a, x \in \{0,1\}^n$, scriviamo $a \cdot x$ per indicare l'inner product bit a bit di a e x : $a_1x_1 \oplus a_2x_2 \oplus \dots \oplus a_nx_n$. Il problema di Bernstein Vazirani ha a che fare con quelle funzioni $f_a \in \mathcal{F}_1^n$ ¹³ t.c. $f_a(x) = a \cdot x$.

Vogliamo **determinare** a invocando f_a , cercando di minimizzare il numero di invocazioni.

Nella computazione classica, non possiamo svolgere il task senza invocare f_a n volte¹⁴. Con la computazione Quantica, basta invocare f_a **solo una volta**.



Vediamo come funziona:

$$|000\dots 0\rangle \xrightarrow{\mathcal{H}^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \xrightarrow{f_a} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{a \cdot x} |x\rangle \xrightarrow{\mathcal{H}^{\otimes n}} |a\rangle$$

Un esempio:

Interpretazione: (usando $|a| = 5$)

¹³1 perché abbiamo un solo bit in output a causa dell'inner product bit a bit

¹⁴in input abbiamo stringhe di lunghezza n . Possiamo, per ogni chiamata, trovare l'i-esimo bit di a . Quindi abbiamo bisogno di n chiamate $s \in \{0,1\}^n$

Let's go through a specific example for $n = 2$ qubits and a secret string $s = 11$. Note that we are following the formulation in Reference [2] that generates a circuit for the Bernstein-Vazirani quantum oracle using only one register.

1. The register of two qubits is initialized to zero:

$$|\psi_0\rangle = |00\rangle$$

2. Apply a Hadamard gate to both qubits:

$$|\psi_1\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

3. For the string $s = 11$, the quantum oracle performs the operation:

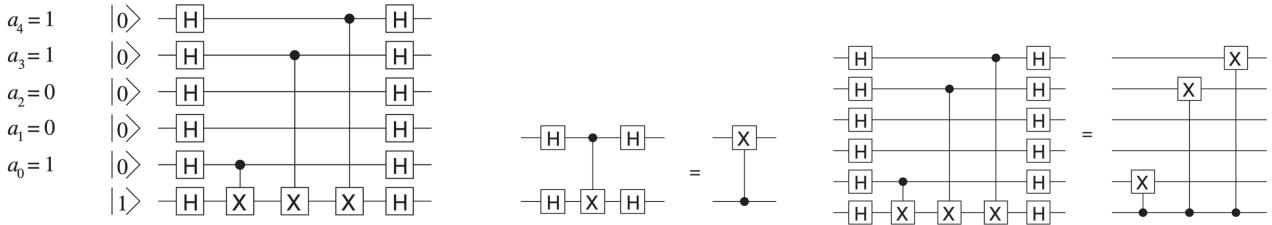
$$|x\rangle \xrightarrow{f_s} (-1)^{x \cdot 11}|x\rangle.$$

$$|\psi_2\rangle = \frac{1}{2}((-1)^{00 \cdot 11}|00\rangle + (-1)^{01 \cdot 11}|01\rangle + (-1)^{10 \cdot 11}|10\rangle + (-1)^{11 \cdot 11}|11\rangle)$$

$$|\psi_2\rangle = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle)$$

4. Apply a Hadamard gate to both qubits:

$$|\psi_3\rangle = |11\rangle$$



5.3 Simon problem

Come in Bernstein-Vazirani, anche nel problema di Simon abbiamo a che fare con una f che dipende da un $a \in \{0, 1\}^n$. Tuttavia, questo é una funzione definita in \mathcal{F}_n^n . Sappiamo che f é **periodica modulo a** , ovvero

$$f(x) = f(y) \iff y = x \oplus a \iff x \oplus y = a \quad \forall x, y \in \{0, 1\}^n$$

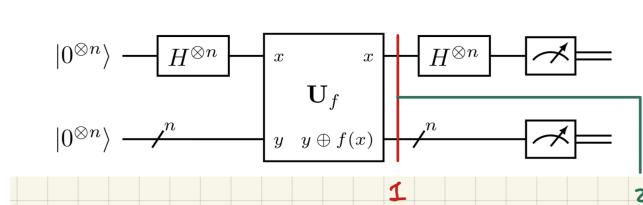
Siamo interessati a computare funzionali parametrici, che hanno come input una funzione $f \in \mathcal{F}_n^n$ é periodica di modulo a (come scritto su), e vogliamo determinare quale sia il **numero minimo** di invocazioni di f per trovare a .

Classicamente, possiamo risolvere (con alta probabilitá) soltanto invocando f circa $2^{n^{15}}$ volte, ovvero **esponenziale**. Approccio classico: Faccio k query e ottengo $y_1 = f(x_1, \dots, y_k = f(x_k)$. Ora:

- se $\forall i, j \in \{1, \dots, k\} |y_i \neq y_k \implies$ sappiamo che $a \neq x_i \oplus x_j$
- se $y_i = y_k \implies a = x_i \oplus x_j$

\implies caso pessimo è esponenziale, in quanto ci sono $\frac{k(k-1)}{2}$ paia di indici distinti nell'insieme $\{1, \dots, k\}$, quindi il numero di query nel caso pessimo é esponenziale.

Il circuito quantico impiega **tempo polinomiale**. Analizziamo cosa fa il circuito:



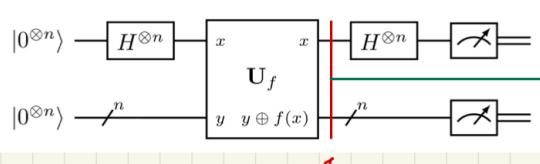
Il circuito da solo non risolve il problema. Quello che facciamo é usare il circuito piú volte in questo modo:

1. $i \leftarrow 1$

¹⁵Dobbiamo provare fino a quando 2 input diversi hanno stesso output. Otteniamo tempo esponenziale perché é un approccio **brute force**.

Spiegazione C_{SIMON}

1. Dopo $H^{\otimes n}$ e U_f lo stato è: $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$



2. Dopo la misurazione del registro in output, abbiamo lo stato:

$$\frac{1}{\sqrt{2}} \left[\frac{1}{\sqrt{2}} |x_0\rangle |f(x_0)\rangle + \frac{1}{\sqrt{2}} |x_0 \oplus 2\rangle |f(x_0 \oplus 2)\rangle \right]$$

c'è ancora superposizione (50% 1, 50% 2)
non possiamo doverne. Altrimenti: potremmo determinare x_0 e $x_0 \oplus 2$ con alte probabilità.

3. Vediamo cosa succede dopo l'applicazione di $H^{\otimes n}$ sui registri dell'input:

$$H^{\otimes n} \left(\frac{1}{\sqrt{2}} |x_0\rangle |f(x_0)\rangle + \frac{1}{\sqrt{2}} |x_0 \oplus 2\rangle |f(x_0 \oplus 2)\rangle \right) = \frac{1}{\sqrt{2^{n/2}}} \sum_{y \in \{0,1\}^n} \left[(-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus 2) \cdot y} \right] |y\rangle$$

Consideriamo i coefficienti: $(-1)^{(x_0 \oplus 2) \cdot y} = (-1)^{x_0 \cdot y} (-1)^{2 \cdot y}$

Distinguiamo due casi:

$$\begin{aligned} \bullet & 2 \cdot y = 1 \Rightarrow (-1)^{(x_0 \oplus 2) \cdot y} = (-1)^{x_0 \cdot y} (-1) = -(-1)^{x_0 \cdot y} \Rightarrow \text{Negative interference: somma dei coefficienti è } 0 \\ \bullet & 2 \cdot y = 0 \Rightarrow (-1)^{(x_0 \oplus 2) \cdot y} = (-1)^{x_0 \cdot y} \end{aligned}$$

In altre parole, le y che sopravvivono alla negative interference sono precisamente quelle per cui $2 \cdot y = 0$.

Quindi, il comportamento generale di C_{SIMON} è che il circuito fa sampling uniforme delle stringhe t.c. $2 \cdot y = 0$

2. Applicazione di C_{SIMON} per ottenere, nei primi n qubit, il valore $|x_i\rangle$

3. Controlla se il sottospazio generato dai vettori x_1, \dots, x_n abbia dimensione minore o uguale a $n - 1$. Se si, vai allo step 4, altrimenti incrementa i di 1 e vai allo step 2

4. Risovi il sistema lineare di equazioni:

$$\begin{cases} a \cdot x_1 = 0 \\ \vdots \\ a \cdot x_i = 0 \end{cases}$$

5. Se esiste una soluzione unica non nulla di a , allora ritornala, altrimenti fail.

Nota: la probabilità di fallire dipende dal numero di iterazioni che facciamo. Infatti, vogliamo che il rank della matrice (del sistema lineare) sia massimo, ma nessuno ce lo garantisce. Quindi, potenzialmente, più volte invochiamo il circuito e più migliora la probabilità.

5.4 Shor's Algorithm

L'algoritmo di Shor è considerato uno dei **risultati** più importanti nella teoria della computazione. È il **primo** problema che risolve un problema vero, e non una sorta di gioco.

Il problema risolto include:

- rompere **RSA**, uno degli schemi di criptazione molto usato (basato su chiave pubblica)
- **Fattorizzazione** di interi
- Logaritmo discreto.

A partire da questo algoritmo, è nata la **post-quantum cryptography**.

In ogni caso, anche questo si tratta di un algoritmo basato sul **period finding**. Stiamo quindi sempre cercando il periodi di una funzione, che in questo caso è più aritmetica (rispetto alla stringhe prese in considerazione da Simon). Usiamo proprio la somma tra numeri naturali nella definizione (unica differenza con Simon)

5.4.1 Preliminari

Sia \mathbb{G}_N l'insieme di tutti gli interi positivi (1 incluso) che sono strettamente minori di N e che non hanno fattori primi in comune con N .

Definiamo la **moltiplicazione modulo N** in questo modo:

$$(x, y) \mapsto x \cdot y \bmod N$$

La moltiplicazione modulo N é ben definita, associativa, ha identitá (1) e ha un inverso. É dunque un **gruppo** (finito).

La cardinalità $\Phi(N) \in \mathbb{N}$ di \mathbb{G}_N è al massimo $N - 1$

Quando N é **primo**, allora il *teorema di Fermat* ci dice che

$$\forall a \in \mathbb{G}_N \quad a^{N-1} \equiv 1 \bmod N$$

Una variazione molto interessante del teorema di Fermat è quella in cui $N = pq$ dove p, q sono **primi**:

$$\forall a \in \mathbb{G}_N \quad a^{(p-1)(q-1)} \equiv 1 \bmod N$$

In questo caso $\Phi(N) = (p-1)(q-1)$ ($< N - 1$)

Nota (alla base di RSA): il numero $N - 1$ è ovviamente facilmente computabile a partire da N , mentre $(p-1)(q-1)$ assolutamente no. Il modo più efficiente è fattorizzando N

In generale,

$$\forall a \in \mathbb{G}_N \quad a^{\Phi(N)} \equiv 1 \bmod N$$

L'**ordine** di un elemento $n \in \mathbb{N}$ é il piú piccolo r t.c.

$$n^r \equiv 1$$

L'ordine di un elemento divide sempre l'ordine del gruppo ($\Phi(N)$). Se l'ordine di un elemento é uguale all'ordine del gruppo, allora quell'elemento é un *generatore* del gruppo. Altrimenti n genera un **sottogruppo** contenente tutti quegli elementi che possono essere scritti

$$n^k \bmod N \text{ dove } k \in \mathbb{Z}$$

$\langle n \rangle^{16} = \langle m \rangle \implies n$ e m hanno lo stesso ordine.

5.4.2 Rompere RSA

RSA é uno degli schemi di criptazione basati a chiave pubblica piú comuni. Ci sono due parti, Alice e Bob che vogliono comunicare su un canale non sicuro. Quello che succede è questo:



L'idea che c'è dietro é:

dato $N = pq$ (p e q sono primi abbastanza grandi, esprimibili ad esempio con 256 bit) e dati $c, d \mid cd \equiv 1 \bmod \Phi(N)$, la chiave pubblica sarà (N, e) , la chiave privata (N, d) .

La sicurezza si basa sul fatto che conosciamo soltanto N , non p e q , quindi computare $\Phi(N)$ non é assolutamente semplice. Se conoscessimo $\Phi(N)$ potremmo calcolare semplicemente l'inverso.

- un messaggio é semplicemente un elemento all'interno di \mathbb{G}_N
- dato un messaggio $m \in \mathbb{G}_N$ e una chiave pubblica (N, e) , il messaggio criptato sarà: $Enc(m, (N, e)) = m^e \bmod N$
- dato un testo cifrato $c \in \mathbb{G}_N$ e una chiave privata (N, d) , il messaggio decrittato sarà: $Dec(c, (N, d)) = c^d \bmod N$

$$Dec(Enc(m, (N, e)), (N, d)) = m$$

Il primo metodo per **rompere** RSA consiste in: dato $N = pq$ e $e \in \mathbb{G}_{(p-1)(q-1)}$, determina d t.c.

$$ed \equiv 1 \bmod (p-1)(q-1)$$

¹⁶Spazio generato

CORRECTNESS OF RSA

$$\begin{aligned}
 \text{Dec}(\text{Enc}(m, (N, e)), (N, d)) &= (m^e \bmod N)^d \bmod N \\
 &\stackrel{\downarrow}{=} (m^{ed} \bmod N) \\
 &\stackrel{\downarrow}{=} (m^{1+k\phi(N)} \bmod N) \\
 &\stackrel{\downarrow}{=} (m \cdot m^{k\phi(N)}) \bmod N \\
 &\stackrel{\downarrow}{=} m
 \end{aligned}$$

Questo è il caso in cui cerchiamo di determinare la chiave privata dalla chiave pubblica. L'operazione di inversione di e è aritmeticamente fattibile (usando le definizioni date prima), tuttavia in questo caso il problema è calcolare $(p-1)(q-1)$, anche se abbiamo N .

Il problema è in questo caso la **fattorizzazione** di N , in quanto se riuscissimo a fattorizzarlo potremmo poi calcolare $p-1$ e $q-1$ facilmente (a partire da q e p)¹⁷

Il secondo metodo invece si basa sul **period finding**. Partiamo da questo presupposto: se possiamo trovare, dato c e N , il più piccolo $r \geq 1$ t.c.

$$c^r \equiv m \pmod{N}$$

¹⁸, possiamo trovare il messaggio in modo abbastanza naturale.

Vediamo quindi ora perché il period finding riesce a rompere RSA.

- Innanzitutto, osserviamo come l'ordine di c e l'ordine di m deve essere lo stesso.
- in realtà, c è una potenza di m e m è una potenza di c . ¹⁹ Questo significa che m è un **elemento** del sottogruppo generato da c e viceversa:

$$c \in \langle m \rangle \quad m \in \langle c \rangle$$

Questo significa che i due sottogruppi devono essere uguali, i.e. $\langle m \rangle = \langle c \rangle$, ovvero l'ordine di m e l'ordine di c deve essere lo stesso (primo punto).

- Da adesso in poi, consideriamo r come l'**ordine** di c (ovvero il periodo).
- Poiché $e \in \mathbb{G}_{(p-1)(q-1)}$ e r **divide** $(p-1)(q-1)$, quindi $\gcd(e, r) = 1$ ²⁰.
- prendiamo il resto della divisione di $e \bmod r$:

$$e = k \cdot r + e'$$

- e' , ovviamente, non può avere fattori comuni con r , altrimenti anche e li avrebbe, ovvero $\gcd(e', r) = 1$
- per come è definito \mathbb{G}_N , esiste un inverso d' di $e' \bmod r$, i.e.

$$e' \cdot d' \equiv 1 \pmod{r}$$

- dunque possiamo concludere che:

$$ed' \equiv (kr + e')d' \equiv krd'^{21} + e'd' \equiv e'd' \equiv 1 \pmod{r}$$

in altre parole, otteniamo un inverso di $e \bmod r$, ovvero d' .

- Possiamo ora determinare il messaggio:

$$c^{d'} \equiv m^{ed'} \equiv m^{1+kr} \equiv m \cdot (m^k r) \equiv m \pmod{N}$$

¹⁷L'algoritmo di Shor è spesso presentato come un algoritmo di fattorizzazione, ma in realtà, come vedremo, si tratterà di period finding. E sfruttando il period finding si riesce a fattorizzare.

¹⁸ovvero riusciamo a calcolare l'order del ciphertext

¹⁹banalmente quello che fa RSA è calcolare delle potenze, di cui non conosciamo l'esponente.

²⁰non ci sono fattori comuni tra r e e

²¹Si cancella in $\bmod r$ in quanto è multiplo di r

L'algoritmo di Shor quindi è un algoritmo che computa il **periodo di una funzione** della forma:

$$c \mapsto b^x \bmod N$$

dove b è parte stessa della funzione (ovvero diverse b corrispondono a diverse funzioni). Nel caso di RSA, $b = c$. Possiamo ricondurci quindi a quello che è l'algoritmo di Simon, tuttavia abbiamo un dominio differente (quello di Simon è più semplice, in quanto si utilizzano insiemi di stringhe, qui invece abbiamo a che fare con numeri naturali, XOR è più semplice sulle stringhe rispetto all'addizione di numeri naturali)²².

Proviamo però ad applicare lo stesso ragionamento (naive) utilizzato con Simon, ovvero computiamo quindi parallelamente delle superposizioni massime usando le Hadamard delle stringhe di bit e poi le usiamo come input per un oracolo che computa la funzione descritta su e poi misuriamo l'output. Lo stato risultante sarà:

$$|\Psi\rangle_n = \frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + kr\rangle$$

dove l'input che misuriamo è $|x_0 + kr\rangle$. Compare la r , che è il periodo che stiamo cercando, dove m è il più piccolo intero t.c. $mr + x_0 \geq 2^n$

Siamo ancora lontani però dalla soluzione, in quanto x_0 ci tiene ancora lontano da kr (poiché non si parla di stringhe di bit come in Simon, qui non abbiamo solo due possibilità, ovvero 0 o 1).

Un altro problema è la presenza dell'esponente x nella funzione. Questo rende la funzione non semplice da computare. Non possiamo servirci di algoritmi naive come quelli visti fino ad ora per tradurlo in un circuito (basalmente non possiamo fare un for poiché avremmo un numero di moltiplicazioni troppo grande). Con algoritmi naive è esponenziale in x , mentre dobbiamo renderlo logaritmico in x (utilizzeremo la Fast exponentiation). Non possiamo quindi procedere come in Simon, dobbiamo applicare qualcosa di leggermente più complesso.

Quantum Fourier Transform Attraverso la QFT (Quantum Fourier Transform) andremo a eliminare quell' x_0 per ottenere soltanto kr .

Ovviamente si basa sulla trasformata di Fourier (che obv non abbiamo studiato)²³. Vediamo quindi cosa succede quando utilizziamo la QFT come un operatore unitario su un elemento della base computazionale.

$$\mathbf{U}_{FT}^n |x\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{y=0}^{2^n-1} e^{2\pi i \frac{xy}{2^n}} |y\rangle$$

Otteniamo dunque un vettore in superposizione massima. Questo prima lo avevamo ottenuto solamente con le Hadamard, che quindi giocheranno un ruolo fondamentale anche qui. Possiamo considerare la QFT una "variazione sul tema" dell'operatore Hadamard.

Tuttavia, cambia molto il coefficiente, che si tratta di un numero complesso.

\mathbf{U}_{FT}^n può essere implementato con un circuito che utilizza soltanto gate unitari e di grandezza quadratica. Vediamo come applicarlo a quello visto prima e perché può essere utile.

$$\begin{aligned} \mathbf{U}_{FT} \left(\frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + kr\rangle \right) &= \frac{1}{2^{\frac{n}{2}}} \sum_{y=0}^{2^n-1} \frac{1}{\sqrt{n}} \sum_{k=0}^{m-1} e^{2\pi i \frac{(x_0+kr)y}{2^n}} |y\rangle = ^{24} \\ &= \sum_{y=0}^{2^n-1} e^{2\pi i \frac{x_0 y}{2^n}} \frac{1}{\sqrt{2^n m}} \left(\sum_{k=0}^{m-1} e^{2\pi i \frac{kry}{2^n}} \right) |y\rangle \end{aligned}$$

Quindi, se ora andassimo a *tutti* i bit, la probabilità di osservare y misurando i registri di input è:

$$\begin{aligned} p(y) &= |\alpha_y|^2 = ^{25} \\ &= \left| e^{2\pi i \frac{x_0 y}{2^n}} \frac{1}{\sqrt{2^n m}} \left(\sum_{k=0}^{m-1} e^{2\pi i \frac{kry}{2^n}} \right) \right|^2 = \\ &= \left| e^{2\pi i \frac{x_0 y}{2^n}} \right|^2 \cdot \left| \frac{1}{\sqrt{2^n m}} \right|^2 \cdot \left| \sum_{k=0}^{m-1} e^{2\pi i \frac{kry}{2^n}} \right|^2 = ^{26} \\ &= 1 \cdot \frac{1}{2^n m} \cdot \left| \sum_{k=0}^{m-1} e^{2\pi i \frac{kry}{2^n}} \right|^2 \end{aligned}$$

²²Periodicità dello XOR è molto più semplice del periodi aritmetici

²³tutta la parte dello studio del perché la trasformata di Fourier è fondamentale nello studio di funzioni periodiche non è argomento di questo corso.

Questo risultato è importante in quanto otteniamo uno stato in cui se misuriamo i registri di input otterremo y con una probabilità che **dipende da y e da kr** , ma **non da x_0** .

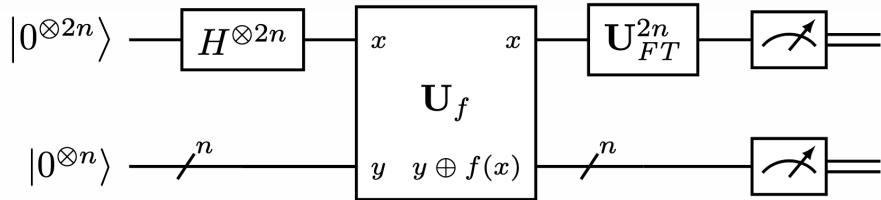


Figura 4: Circuito di Shor

Le differenze tra il circuito di Shor e quello di Simon sono 2:

1. Invece di applicare le Hadamard nei registri di input applichiamo le U_{FT} , e questo perché dobbiamo togliere la x_0 .
2. ci sono 2^n qubits nei registri di input. Sono qubit che serviranno nel **post-processing**, per la riduzione degli errori. Altrimenti all'oracolo basterebbero n bit per calcolare la funzione.

Post-processing misurando y , tuttavia, non otteniamo quello che ci interessa, ovvero r . Abbiamo bisogno di un po' di post-processing.

Teorema 5.1. *La probabilità $p(y)$ ha valore massimo quando y è vicino ai multipli interi di $\frac{2^{2n}}{r}$, dove 2^{2n} è il numero di elementi in superposizione (perché abbiamo usato U_{FT}^{2n} , ovvero parametrizzata con $2n$)*

Dimostrazione. stacce □

Se y si trova ad una distanza di massimo $\frac{1}{2}$ da $\frac{j2^{2n}}{r}$ per qualche intero j , allora

$$\left| \frac{y}{2^{2n}} - \frac{j}{r} \right| \leq \frac{1}{2^{2n+1}}$$

Una volta computato $\frac{y}{2^{2n}}$ e $\left| \frac{y}{2^{2n}} - \frac{j}{r} \right| \leq \frac{1}{2^{2n+1}}$ si possono ottenere j_0, r_0 t.c. $\frac{j_0}{r_0} = \frac{j}{r}$ e j_0, r_0 sono **irriducibili**.

²⁷ In particolare, r_0 è una divisore di r .

Successivamente, si controllare se r_0 è a sua volta un **periodo**. La probabilità che $r_0 = r$ è abbastanza alta ($> 40\%$). Se r_0 non è un periodo, bisogna ricominciare da 0. Ci sono delle tecniche di ottimizzazione (ad esempio cercando delle r vicine a r_0).

Ripetendo più volte tutto comunque si riesce a trovare r .

La cosa positiva di questi problemi è che sono facilmente verificabili (ovvero sono in NP).

5.4.3 Fast Exponentiation

La computazione della funzione $x \mapsto b^x \bmod N$ è necessaria per la computazione dell'algoritmo di Shor (ovvero dobbiamo implementare l'oracolo U_f). In particolare, abbiamo bisogno che questo sottocircuito non sia eccessivamente grande. Ovviamente iterando sul numero delle moltiplicazioni **non funziona**, in quanto, nel caso pessimo, il numero di iterazioni è uguale a x (richiede tempo esponenziale).

Abbiamo quindi bisogno di un algoritmo chiamato **fast exponentiation**, che usa 3 registri: l'input register x , l'output register y , work register w .

- All'inizio, $y = 1$ e $w = b$.
- Iterativamente, facciamo l'update di y e w , facendo w^2 e moltiplicando y per w sse il bit corrispondente di x è 1.
- il risultato è in y

In questo modo, il processo iterativo è eseguito **non** un numero pari a x di volte, ma pari ai bit di x (ovvero la lunghezza della stringa x).

Questo algoritmo funziona molto bene perché siamo in aritmetica con modulo. Ovvero, non dobbiamo preoccuparci della lunghezza dei risultati intermedi (ad esempio $b^6 4$ potrebbe essere difficile da computare) perché tagliamo ad N .

²⁷Continuous Fraction Development, fa parte dell'Analisi Numerica (classica).

FAST EXPONENTIATION - EXAMPLE

w	y	x	
b	1	<u>101101</u>	
b^2	b	<u>45</u>	1 (LEAST SIGNIFICANT BIT)
b^4	b		2
b^8	b^{1+4}		3
b^{16}	b^{2+4+8}		4
b^{32}	b^{2+4+8}		5
b^{64}	$b^{1+4+8+32}$		6
	<u>b^{45}</u>		

Figura 5: Esempio della fast exponentiation. In questo caso usiamo 6 iterazioni invece di 45.

5.4.4 Factoring e Logaritmo Discreto

Lo stesso algoritmo di Shor possiamo risolvere altri problemi interessanti:

- **Integer Factorization:** dopo aver controllato se N è primo (si fa in tempo polinomiale). Se non lo è, genera $1 < a < N$ randomicamente. Se a è coprimo con N (ovvero gcd, polinomiale), calcola il periodo r di $x \mapsto a^x \pmod{N}$. Se r è pari, allora qualsiasi fattore di N è un fattore di $a^{r/2} - 1$ o di $a^{r/2} + 1$ (dividiamo in due sottoproblemi - Divide et Impera).
- **Discrete Logarithm:** abbiamo bisogno anche qui di preprocessing e postprocessing.

5.5 Grover's Algorithm

Non abbiamo un guadagno esponenziale, tuttavia questo è comunque interessante.

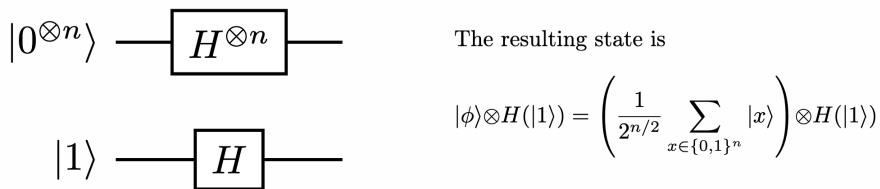
Il problema è la ricerca all'interno di un database non strutturato:

- *input:* un circuito booleano (non accessibile) che computa una funzione sconosciuta: $f : \{0,1\}^n \rightarrow \{0,1\}$ t.c. $\exists a \in \{0,1\}^n$ con $f(x) = 1 \iff x = a$
- *output:* l'unica stringa $a \in \{0,1\}^n | f(a) = 1$

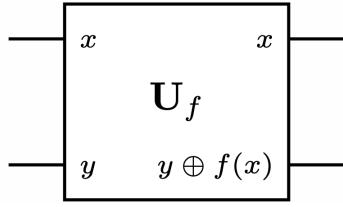
Vogliamo appunto cercare la s invocando f . In modo classico, per eseguire questa verifica, servono 2^n iterazioni (ovvero lo dobbiamo calcolare per ogni coordinata).

L'algoritmo di Grover, usando una tecnica che si chiama **amplitude amplification** riesce a risolvere il problema in tempo $O(\sqrt{2^n})$. Non cambiamo di classe di complessità, ma comunque rappresenta un buon guadagno. Vediamo la soluzione andando ad analizzare le varie componenti utilizzate in modo separato:

- **Primo componente:** classico per mettere in superposizione (massima) senza entanglement (tra $H(|1\rangle)$ e il resto non c'è entanglement).



- **Secondo componente:**

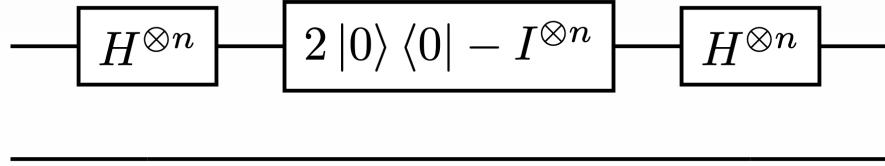


On an input $|x\rangle \otimes H(|1\rangle)$, the output is
 $(-1)^{f(x)}|x\rangle \otimes H(|1\rangle)$
and so there must be a unitary transformation \mathbf{V} on n qubits capturing its behaviour.

$$\begin{aligned}
U_f(|x\rangle \otimes H(|1\rangle)) &= U_f\left(\frac{1}{\sqrt{2}}|x\rangle \otimes |0\rangle - \frac{1}{\sqrt{2}}|x\rangle \otimes |1\rangle\right) = \\
&= \frac{1}{\sqrt{2}}U_f\left(\frac{1}{\sqrt{2}}|x\rangle \otimes |0\rangle\right) - \frac{1}{\sqrt{2}}U_f\left(\frac{1}{\sqrt{2}}|x\rangle \otimes |1\rangle\right) =^{28} \\
&= \frac{1}{\sqrt{2}}|x\rangle \otimes |f(x)\rangle - \frac{1}{\sqrt{2}}|x\rangle \otimes \neg f(x)\rangle = \\
&= (-1)^{f(x)}|x\rangle \otimes \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = \\
&= (-1)^{f(x)}|x\rangle \otimes H(|1\rangle)
\end{aligned}$$

Nel caso in cui $x = a, f(x) = 1$ e quindi avviene la negazione di tutta l'amplitude. Sfrutteremo questo nel terzo componente (altrimenti è inutile)

- **Terzo componente:** cerchiamo di amplificare l'amplitude del componente a



The action of the component on the first n qubits can be seen as that of a unitary transformation, call it \mathbf{W}

Analizziamo il componente interno. Partiamo da quel simbolo nuovo. A partire da uno stato (VETTORE) μ , si può **formare un OPERATORE** $\langle \mu |$. Avremo quindi che

$$\langle \mu | (\cdot) \rangle = \langle \mu | \cdot \rangle$$

è semplicemente una notazione che supporta l'associatività, in modo da poter formare espressioni combinando ket e bra, in particolare possiamo comporre questi operatori.

Quello che succede ora utilizzando le due Hadamard, è che la prima fluisce nel $|0\rangle$, la seconda nel $\langle 0|$

$$H^{\otimes n} \circ 2|0\rangle\langle 0| - I^{\otimes n} \circ H^{\otimes n} = 2|\phi\rangle\langle\phi| - I^{\otimes n}$$

In questo modo dimostriamo come tutto sia fatto da gate elementary.

Analisi Geometrica Vediamo come operano questi operatori sugli stati $|a\rangle$ e $|\phi\rangle$:

$$\begin{aligned}
\mathbf{V}|a\rangle &= -|a\rangle; & \mathbf{V}|\phi\rangle &= |\phi\rangle - \frac{2}{2^{n/2}}|a\rangle; \\
\mathbf{W}|a\rangle &= \frac{2}{2^{n/2}}|\phi\rangle - |a\rangle & \mathbf{W}|\phi\rangle &= |\phi\rangle
\end{aligned}$$

In tutti i 4 casi otteniamo una combinazione lineare dei 2 vettori. Possiamo vedere quindi come una combinazione lineare del sottospazio generato dai due vettori. Vediamo perché queste 4 espressioni valgono:

$$\begin{aligned}
V|a\rangle &= (-1)^{f(a)}|a\rangle = -1|a\rangle = -|a\rangle \\
V|\phi\rangle &= V\left(\frac{1}{2^{\frac{n}{2}}}\sum_{x \in \{0,1\}^n}|x\rangle\right) = \frac{1}{2^{\frac{n}{2}}}\sum_{x \in \{0,1\}^n}V(|x\rangle) = \frac{1}{2^{\frac{n}{2}}}\sum_{x \in \{0,1\}^n}(-1)^{f(x)}|x\rangle = \\
&= \frac{1}{2^{\frac{n}{2}}}\left(\sum_{x \in \{0,1\}^n \setminus \{a\}}(-1)^{f(x)}|x\rangle - |a\rangle\right) = \frac{1}{2^{\frac{n}{2}}}\left(\sum_{x \in \{0,1\}^n}|x\rangle - 2|a\rangle\right) = \\
&= |\phi\rangle - \frac{2}{2^{\frac{n}{2}}}|a\rangle
\end{aligned}$$

$$\begin{aligned}
W|a\rangle &= 2|\phi\rangle\langle\phi| - 1|a\rangle = 2|\phi\rangle\langle\phi|a\rangle - |a\rangle = 2|\phi\rangle\langle\frac{1}{2^{\frac{n}{2}}}\sum|x\rangle|a\rangle - |a\rangle = \\
&= 2|\phi\rangle\frac{1}{2^{\frac{n}{2}}}\sum\langle x|a\rangle - |a\rangle = {}^{29}2|\phi\rangle\frac{1}{2^{\frac{n}{2}}}[0 + \dots + 0 + 1 + 0 \dots + 0] - |a\rangle = \\
&= 2 \cdot \frac{1}{2^{\frac{n}{2}}}|\phi\rangle - |a\rangle = \frac{2}{2^{\frac{n}{2}}}|\phi\rangle - |a\rangle \\
W|\phi\rangle &= (2|\phi\rangle\langle\phi| - 1)\phi = 2|\phi\rangle\langle\phi|\phi\rangle - |\phi\rangle = 2|\phi\rangle - |\phi\rangle = |\phi\rangle
\end{aligned}$$

Quindi quello che succede è che applicando \mathbf{V} e \mathbf{W} , lo stato **rimane** nella forma $|\psi\rangle \otimes H(|1\rangle)$, dove $|\psi\rangle$ vive in uno spazio bidimensionale di tutte le compinazioni lineari $\alpha|\phi\rangle + \beta|a\rangle$, con $\alpha, \beta \in \mathbb{R}$ e $\alpha^2 + \beta^2 = 1$.

Gli stati $|a\rangle$ e $|\phi\rangle$ sono quasi ortogonali, in quanto l'inner product è molto piccolo: $\theta = \langle a|\phi\rangle = 2^{-\frac{n}{2}}$. C'è quindi un vettore che forma un piccolo angolo con $|\phi\rangle$ che è ortogonale con $|a\rangle$. Chiamiamo questo vettore $|a^\perp\rangle$. Quindi \mathbf{V} e \mathbf{W} sono reflections, e il loro prodotto \mathbf{WV} è una **rotazione**.

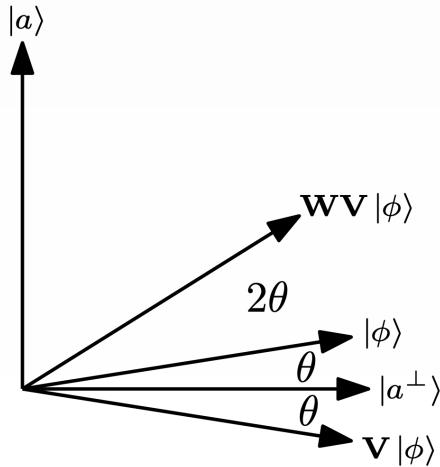
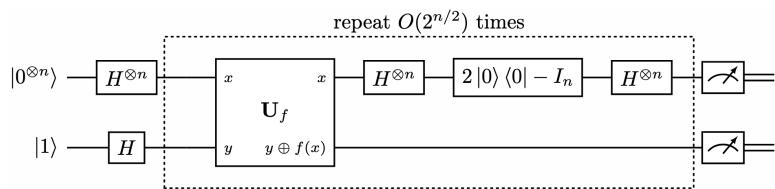


Figura 6: Analisi Geometrica. Leggere a partire da $|\phi\rangle$

Partendo da $|\phi\rangle$, applicando \mathbf{V} otteniamo la reflection su $|a^\perp\rangle$. A questo punto, se applichiamo $\mathbf{W}(WV(|\phi\rangle))$ otteniamo una reflection su $|\phi\rangle$

Continuando questo procedimento di rotazione, se lo applichiamo diverse volte arriveremo quindi vicini ad $|a\rangle$. A questo punto possiamo misurare.

Circuit La parte centrale praticamente è un loop, è eseguita quel numero di volte.



6 Protocolli quantistici

Fino ad adesso abbiamo lavorato con i circuiti quantistici:

- sequenze di operazioni unitarie applicate a qubits in una maniera ben precisa.
- eseguiti da hardware quantico o simulati da hardware classico.
- modo per computare funzioni da $\{0,1\}^n$ a $\{0,1\}^m$

Vediamo ora cosa sono i **Quantum Protocols**.

- protocolli di comunicazioni, ovvero una sequenza di computazioni e steps di comunicazioni eseguita da un insieme di agenti (possibilmente distribuiti).
- gli agenti possono non solo scambiarsi dati classici, ma anche qubits (possono anche non scambiarsi nulla, ma abbiamo la potenza dell'**entanglement**).

Definiamo ora formalmente cos'è un protocollo quantico. Non esiste un modello formale unico, quindi probabilmente per ora non ha senso introdurne uno. Ci sono attualmente *multi* linguaggi e modelli di protocolli quantici, quindi rimaniamo informali.

Descriveremo il circuito e chi possiede i qubit all'interno del circuito (e come sono interscambiati), oppure descriveremo cosa deve fare ciascun agente con i propri dati.

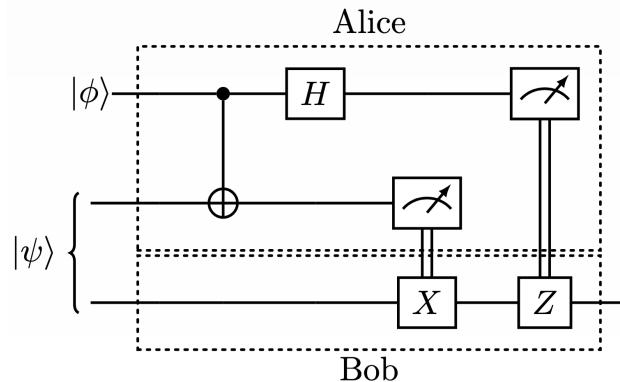
6.1 Quantum Teleportation

Supponiamo che Alice ha un qubit che non può misurare, e vuole "mandarlo" a Bob.

Vogliamo farlo senza mandare *fisicamente* il qubit, ovvero può comunicare in modo classico con Bob ma non può mandare dati quantistici a Bob.

Questo risultato può essere raggiunto, con l'unica condizione che Alice e Bob condividono un paio di qubit entangled $|\psi\rangle$. Questo paio di qubit *dove* esserci da prima dell'inizio del protocollo, ovvero prima che Alice decida di mandare il qubit ad Bob. Tuttavia, avviene la **distruzione** dei qubit entangled e di quello da comunicare (perdiamo risorse).

Il circuito che rappresenta la Quantum Teleportation è:



Il qubit $|\phi\rangle$ è quello che Alice vuole inviare a Bob, mentre $|\psi\rangle$ è la coppia di qubit entangled. La storia di $|\phi\rangle$ è completamente asincrona rispetto a $|\psi\rangle$.

Il risultato delle due misurazioni distrugge l'entanglement e produce 2 bit classici (inviai a Bob in modo classico). Bob in questo modo ottiene $|\phi\rangle$

Vediamo la correttezza:

stato di Bell

- ① $|\phi\rangle_a |\psi\rangle_{ab} = (\alpha|0\rangle_a + \beta|1\rangle_a) \left(\frac{1}{\sqrt{2}} |0\rangle_b + \frac{1}{\sqrt{2}} |1\rangle_b \right)$
- ② $\alpha|0\rangle_a \left(\frac{1}{\sqrt{2}} |0\rangle_b + \frac{1}{\sqrt{2}} |1\rangle_b \right) + \beta|1\rangle_a \left(\frac{1}{\sqrt{2}} |1\rangle_b + \frac{1}{\sqrt{2}} |0\rangle_b \right)$
- ③ $\alpha \frac{1}{\sqrt{2}} (|0\rangle_a + |1\rangle_a) \left(\frac{1}{\sqrt{2}} |0\rangle_b + \frac{1}{\sqrt{2}} |1\rangle_b \right) + \beta \frac{1}{\sqrt{2}} (|0\rangle_a - |1\rangle_a) \left(\frac{1}{\sqrt{2}} |1\rangle_b + \frac{1}{\sqrt{2}} |0\rangle_b \right)$
- ④ Analizziamo cosa succede dopo le due misurazioni (separatamente):

$$\alpha \frac{1}{2} |0\rangle_a |0\rangle_b |0\rangle_b + \alpha \frac{1}{2} |0\rangle_a |1\rangle_b |1\rangle_b + \beta \frac{1}{2} |1\rangle_a |0\rangle_b |0\rangle_b + \beta \frac{1}{2} |1\rangle_a |1\rangle_b |1\rangle_b + \beta \frac{1}{2} |0\rangle_a |1\rangle_b |0\rangle_b + \beta \frac{1}{2} |0\rangle_a |0\rangle_b |1\rangle_b - \beta \frac{1}{2} |1\rangle_a |1\rangle_b |0\rangle_b - \beta \frac{1}{2} |1\rangle_a |0\rangle_b |1\rangle_b$$

$\left. \begin{array}{l} \text{distribuzione flavor} \\ \text{entanglement, il} \\ \text{funzionamento} \\ \text{del CNOT} \\ \text{è elementare: in} \\ \text{superposto, con H} \\ \text{mettiamo ancora più superposto} \end{array} \right\}$

$\left. \begin{array}{l} \text{il elemento: in} \\ \text{superposto, con H} \\ \text{mettiamo ancora più superposto} \end{array} \right\}$

} stato a ③, riscritto

fisicamente indistinguibile da $|0\rangle$

6.2 Quantum Pseudotelepathy

La **Quantum Pseudotelepathy** fa riferimento ai protocolli che sfruttano l'entanglement per dimostrare che (almeno una parte del) la comunicazione tra le diverse parti può essere **evitata**. Ovviamente questo è impossibile nella computazione classica.

Vediamo quindi un piccolo esempio attraverso un gioco (**GHZ**, dagli studiosi).

GHZ game Supponiamo di avere tre giocatori: Alice, Bob e Charlie. C'è un arbitro che effettua a ciascun giocatore una domanda (rispettivamente x, y, z , dove ognuno è $\in \{\blacksquare, \square\}^{30}$), alla quale il singolo giocatore può rispondere solo con 0 o 1. Ovviamente i 3 giocatori, una volta ricevuta la domanda, non possono comunicare tra loro. Chiamiamo le 3 risposte rispettivamente a, b, c .

I giocatori **vincono** solo se lo XOR ($a \oplus b \oplus c$) delle loro **risposte** segue questa tabella:

x, y, z	Requisito per le Risposte $a \oplus b \oplus c$
($\blacksquare, \blacksquare, \blacksquare$)	1
($\blacksquare, \square, \square$)	0
($\square, \blacksquare, \square$)	0
($\square, \square, \blacksquare$)	0

Utilizzando una strategia classica (pre accordata), è impossibile vincere sempre. Si può dimostrare che utilizzando una strategia classica non si potranno mai soddisfare tutte le condizioni di vittoria, ottenendo al massimo una percentuale di vittorie dell'**75%**.

Vediamo ora come, attraverso una strategia quantistica, si riesca a vincere al 100%. I 3 giocatori devono condividere uno stato comune **entangled**, chiamato **GHZ State**:

$$|GHZ\rangle = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle$$

Ogni giocatore utilizza questa strategia (andando ad applicare queste regole al suo bit: Alice il primo, Bob il secondo, Charlie il terzo). Vediamo quindi il caso di Alice:

- Se Alice riceve un \blacksquare , ovvero un 1, quello che fa è:

$$|1yz\rangle \rightarrow \frac{1}{\sqrt{2}}|1yz\rangle + \frac{1}{\sqrt{2}}|0yz\rangle$$

$$|0yz\rangle \rightarrow \frac{1}{\sqrt{2}}|1yz\rangle - \frac{1}{\sqrt{2}}|0yz\rangle$$

³⁰questa domanda vedilo ad esempio come un assegnamento, quindi può essere 1 o 0. Per renderlo più comprensibile, possiamo considerare delle stanze in cui sono i tre giocatori, e ad un giocatore viene assegnato "colore" (rispettivamente 1), mentre agli altri due ad esempio viene assegnato "forma" (rispettivamente 0). È un assegnamento booleano. A loro volta, i giocatori possono rispondere solamente con 1 o 0 (ad esempio, per chi avrà il colore, può rispondere solo blu o rosso).

- Se Alice riceve un \square , ovvero un 0, allora **non fa nulla**.

Vediamo ora un esempio. Supponiamo che Alice riceva un \blacksquare . Quello che succede è quindi:

$$\begin{aligned} \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle &\rightarrow \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2}}|100\rangle - \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle + \frac{1}{\sqrt{2}}|011\rangle \right) = \\ &\rightarrow \frac{1}{2} (|100\rangle - |000\rangle + |111\rangle + |011\rangle) \end{aligned}$$

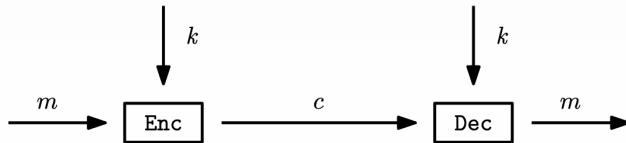
Supponiamo che anche a Bob e Charlie arriva un \blacksquare :

$$\begin{aligned} \frac{1}{2} (|100\rangle - |000\rangle + |111\rangle + |011\rangle) &\xrightarrow{Bob} \\ \frac{1}{2} \left(\left(\frac{1}{\sqrt{2}}|110\rangle - \frac{1}{\sqrt{2}}|100\rangle \right) - \left(\frac{1}{\sqrt{2}}|010\rangle - \frac{1}{\sqrt{2}}|000\rangle \right) + \left(\frac{1}{\sqrt{2}}|111\rangle + \frac{1}{\sqrt{2}}|101\rangle \right) + \left(\frac{1}{\sqrt{2}}|011\rangle - \frac{1}{\sqrt{2}}|001\rangle \right) \right) &= \\ = \frac{1}{2\sqrt{2}} (|000\rangle - |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle + |111\rangle) &\xrightarrow{Charlie} \\ \dots \\ = \frac{1}{2} (|001\rangle + |010\rangle + |100\rangle + |111\rangle) \end{aligned}$$

Ora, a partire da questo stato, quando i giocatori misureranno otterranno sempre un risultato dispari, ovvero 1. Quindi si trova.

6.3 Quantum Key Distribution

Possiamo sfruttare le proprietà del Quantum Computing per effettuare **Criptazione simmetrica**.



Come vediamo nello schema, la chiave utilizzata nel Enc e nel Dec è la stessa: k .

Si possono costruire schemi di encryption simmetrica che sono sicuri ed efficienti. Tuttavia c'è un problema: come possono il sender e il receiver **condivideri** la chiave simmetrica k prima della comunicazione?

Alcune soluzioni sono:

- utilizzo di un canale già **sicuro**.
- utilizzo degli schemi a chiave pubblica-privata per condividere la chiave, e poi passare a encryption simmetrico.
- sfruttando il **quantum computing**.

Focalizziamoci su quest'ultimo caso.

Tradizionalmente, i canali di comunicazioni si assumono classici, ovvero i dati che fluiscono sul canale possono essere **osservati** senza essere alterati, e il cui valore può essere **duplicato**.

In pratica, se parliamo di quantum, osservare dei dati diventa un **quantum measurement** (che ricordiamo fa collassare il sistema), mentre la duplicazione non è possibile a causa del **No-cloning Theorem**. Possiamo sfruttare in qualche modo queste due caratteristiche?

Lavoriamo con due basi ortonormali³¹:

$$\begin{aligned} |\rightarrow\rangle &= |0\rangle & |\uparrow\rangle &= |1\rangle \\ |\nwarrow\rangle &= -\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle & |\nearrow\rangle &= \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \end{aligned}$$

Le due basi sono quindi:

$$\mathbf{P} = \{|\rightarrow\rangle, |\uparrow\rangle\} \quad \mathbf{T} = \{|\nwarrow\rangle, |\nearrow\rangle\}$$

Queste basi ci sono utili perché:

³¹Ortonormali = Ortogonali (vettori della base perpendicolari tra loro - inner product o prodotto scalare = 0) + Normali (norma = 1)

- Se si prepara uno stato sulla base **P** e lo misuriamo rispettivamente a **T** otteniamo un risultato casuale (tra $|\nwarrow\rangle$ e $|\nearrow\rangle$ con prob = $\frac{1}{2}$)
- Se si prepara uno stato sulla base **T** e lo misuriamo rispettivamente a **P** otteniamo un risultato casuale (tra $|\uparrow\rangle$ e $|\rightarrow\rangle$ con prob = $\frac{1}{2}$)

Il **protocollo BB84** quindi sfrutta queste due basi per capire se qualcuno ha ascoltato durante la comunicazione. Quello che succede é:

1. Alice vuole inviare una serie di bit casuali (ovvero la **chiave**) a Bob. La prima cosa che fa è scegliere a caso una delle due basi (**P o T**) per codificarlo in un qubit. Invia quindi sul canale i Bit codificati
2. Bob si ritrova a non sapere che scelta ha fatto Alice riguardo la sequenza di **P e T**. Effettua quindi una scelta casuale per ogni qubit che gli arriva. A questo punto abbiamo due possibili risultati (50%): Bob ha scelto la stessa base di Alice e quindi ottiene il bit corretto; oppure Bob ha scelto la base sbagliata. In questo caso misura 1 o 0 casualmente.
3. A questo punto la stringa di bit di Bob é differente da quella di Alice, dunque non puó usarla come chiave segreta. Sono corretti all'incirca il 50% dei bit ricevuti. Avviene dunque il **sifting**, ovvero Bob e Alice si comunicano in un canale **non sicuro** la lista delle basi utilizzate (T o P). Partono dunque dalla sequenza di bit che entrambi hanno e **tengono** soltanto i bit che hanno misurato con la stessa base. In questo modo ritrovano la stessa sequenza di bit e hanno dunque la loro chiave.

Perché é sicuro? La condivisione della lista delle basi utilizzate é pubblica e non sicura, quindi Eve potrebbe ascoltarla. Tuttavia, non ci puó fare niente perché questa é una fase **successiva** alla comunicazione dei qubit, e dunque non ha piú accesso ai qubit originali per ricostruirsi la chiave.

Eve potrebbe ascoltare la prima comunicazione, quella dei qubit ma anche lei si trova nello stesso dilemma di Bob, ovvero deve scegliere casualmente una base. Facendo questo, va a modificare lo stato dei qubit nel caso di base scelta sbagliata. Questo stato errato poi verrá comunicato a Bob che potrà accorgersi di questo cambiamento una volta condivisa la lista di scelte con Alice.

Al massimo, Eve potrà avere **porzione** della chiave, e Alice e Bob possono accorgersi della sua presenza.

6.4 Quantum Commitment

Un **commitment protocol** permette a due parti (Alice e Bob) di interagire in modo che Alice possa impegnarsi(commit) su un valore scelto, garantendo che:

- Il valore scelto da Alice rimane *nascosto* fino a quando Alice decide di rivelarlo.
- Né Alice né Bob possono *cambiare* il valore scelto da Alice, ovvero il protocollo é vincolante (**binding**).

Nella crittografia classica, esistono varie forme di commitment protocols che si basano su assunzioni crittografiche, come l'esistenza delle one-way functions.

Nel quantum computing, a causa del fatto che l'osservazione puó potenzialmente alterare il valore del dato, sembra un approccio promettente.

Quantum Bit Commitment Supponiamo che Alice e Bob vogliano su un singolo bit, scelto da Alice.

Alice quindi sceglie il valore b . Genera poi n bit casuali, ottenendo dei valori $v_1, \dots, v_n \in \{0, 1\}$ (la password per codificare il bit) e prepara n qubits $|x_1\rangle, \dots, |x_n\rangle$ in questo modo:

$$\begin{aligned} b = 0 &\implies |x_i\rangle \text{ settato a } |v_i\rangle \\ b = 1 &\implies |x_i\rangle \text{ settato a } H(|v_i\rangle) \end{aligned}$$

A questo punto Alice invia a Bob $|x_1\rangle, \dots, |x_n\rangle$, che li mantiene in un posto segreto.

Quando arriverà il momento in cui Alice vuole **rivelare** il valore di b , dirá non solo b , ma anche i valori v_1, \dots, v_n . In questo modo Bob può verificare se Alice lo sta ingannando semplicemente misurando $|x_1\rangle, \dots, |x_n\rangle$ (basandosi sulla base P o T in base al bit b inviatogli).

Analizziamo ora la sicurezza di questo metodo.

Si puó dimostrare che Bob non puó estrarre **nessuna informazione** riguardo b a partire dall'insieme di qubits inviati da Alice. (v_1, \dots, v_n sono completamente randomici e Bob non li conosce nella prima fase).

Si puó dimostrare tuttavia che il protocollo cosí **non é sicuro**:

- Invece di preparare $|x_1\rangle, \dots, |x_n\rangle$, Alice puó preparare n coppie di qubits entangled (Bell state) e mandare *il primo* qubit di ogni coppia a Bob.

- Bob ovviamente non sa che i qubits ricevuti sono entangled
- Al momento della rivelazione, Alice può **cambiare il valore** di b a suo piacimento applicando un H sui propri qubit. Ovviamente, la misurazione va ad influenzare i qubit posseduti da Bob per l'effetto dell'entanglement, quindi non si accorgerà di nulla.

Nota: come provato da Mayers, il quantum bit commitment sicuro senza condizioni é *impossibile*.

Vediamo un piccolo esempio. Per semplicità, supponiamo $n = 1$.

Se Bob misura un qubit che é o $|\phi\rangle$ o $|\psi\rangle$ dove $|\phi\rangle$ e $|\psi\rangle$ sono stati **ortogonali** $\langle\psi|\phi\rangle = 0$, allora la probabilità di osservare 0 é:

$$p(0) = \frac{1}{2} |\langle 0|\phi\rangle|^2 + \frac{1}{2} |\langle 0|\psi\rangle|^2$$

Data una base ortonormale, possiamo definire qualsiasi qubit in quella base. Ovvero, possiamo esprimere $|0\rangle$ nella base ortogonale $\{|\phi\rangle, |\psi\rangle\}$:

$$|0\rangle = |\phi\rangle\langle|\phi|0\rangle + |\psi\rangle\langle|\psi|0\rangle$$

Possiamo quindi concludere che:

$$|\langle 0|\phi\rangle|^2 + |\langle 0|\psi\rangle|^2 = 1 \implies p(0) = \frac{1}{2}$$

7 Error correction

Ci possono essere errori anche nella computazione classica. La materia che si occupa di questo é l'**Information Theory**. Tuttavia, è una materia ora poco studiata dagli informatici, che cercano di astrarre sempre di più, e lasciata più agli ingegneri o chi si occupa della parte "fisica". Inoltre, i **gates** logici alla base della computazione classica effettuano degli errori con probabilità così remota che può essere considerata 0. Diverso è il caso della **comunicazione** a distanza, in cui gli errori sono ancora presenti (infatti é ancora studiato il livello Fisico in un esame di reti).

L'information theory quindi cerca anche delle strategie per **correggere** gli errori. Ad esempio, l'uso **repetition code**, in cui prendiamo un bit che deve essere inviato e lo replichiamo n volte. Usando $n = 3$ ($0 \rightarrow 000, 1 \rightarrow 111$), l'errore può essere riconosciuto e corretto.³²

In **Quantum Computation** abbiamo numerosi altri errori. Non ci sono errori soltanto nella comunicazione, ma qui sono presenti molti anche nella **computazione**.

- qubits sono implementati a livello atomico! Il disturbo causato dall'ambiente è fondamentale e può variare lo stato del sistema
- **Rilevare** un errore é impossibile, in quanto dovremmo misurare il qubit e questo porta al collasso del sistema. Dobbiamo quindi usare una strategia differente (senza guardare il valore direttamente).
- Se dal lato classico un errore può essere modellato come un **random bit flip** a causa di un problema sistema fisico, nel *quantum* ovviamente non abbiamo solo 0 e 1, quindi abbiamo un **errore di fase**. Questi ultimi sono invisibili, in quanto hanno anche la stessa distribuzione dell'output nel momento della computazione.

$$\begin{aligned} \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle &\xrightarrow{\text{Phase error}} \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \\ \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle &\xrightarrow{\text{Bit flipping}} \frac{1}{\sqrt{2}}|1\rangle - \frac{1}{\sqrt{2}}|0\rangle \end{aligned}$$

Nonostante tutti questi problemi, la quantum error connection esiste.

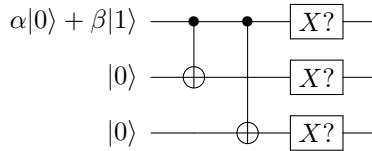
7.1 QECCs

Vediamo ora i **Quantum Error Correcting Codes** (QECCs). Per vedere come funziona, usiamo un setting semplificato.

³²Vista la probabilità comunque bassa di errori, di solito su 3 bit al massimo 1 sarà sbagliato, e dunque possiamo riconoscerlo e correggerlo.

Quantum Repetition Code Partiamo dal provare a ricreare quello che é il repetition code, quindi cerchiamo di fare un Quantum Repetition Code. In questo caso, siamo interessati a risolvere i **singoli qubit flipping errors**.

In altre parole, siamo in questo stato:



Abbiamo quindi uno stato arbitrario $(\alpha|0\rangle + \beta|1\rangle)$ e vogliamo cercare di "replicare" quello stato, ovviamente abbiamo la limitazione del no cloning theorem. Prima dei random gate X (che rappresentano il bit flipping, quindi la possibilità di errore), ci troviamo in questo stato:

$$\alpha|000\rangle + \beta|111\rangle$$

Questo infatti **non é clonare**. Il cloning sarebbe questo (impossibile):

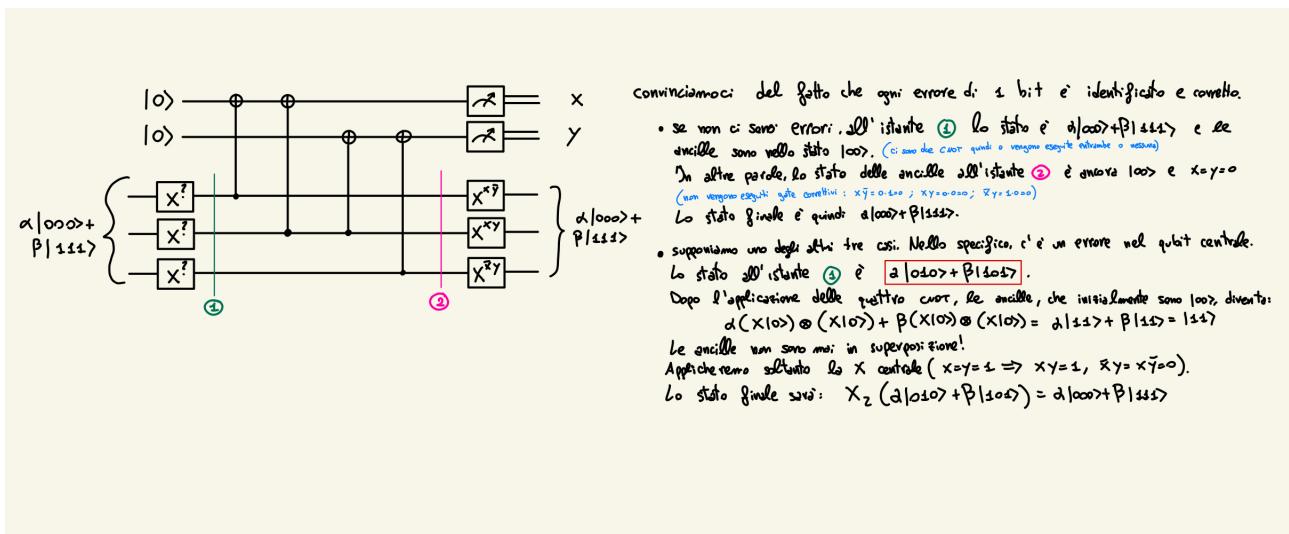
$$(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle \otimes |0\rangle \mapsto (\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle)$$

Quello che stiamo facendo é soltanto replicare la base senza i coefficienti.

Come possiamo quindi, a partire da questo, fare la **detection** e la **correction**?

Vediamo il circuito piú semplice di QECC:

Quello che facciamo é cercare di capire cosa sta succedendo attraverso le cnot. I valori dei due qubit ancillari, una volta misurati, sono poi usati per effettuare una X che vada a correggere quindi l'errore.



Vediamo ora il caso generale degli errori (anche considerando quelli che sono gli errori di fase). Come l'ambiente interagisce sul qubit?



General Error Model Non conosciamo molto dell'ambiente. Sappiamo che si tratta comunque di un sistema quantistico.

Gli **errori** e il **rumore** é dunque un'interazione tra sistema e ambiente. Possiamo quindi vederlo come una forma di trasformazione lineare (non per forza unitaria):

$$|\Psi\rangle|E\rangle \mapsto |\Theta\rangle$$

Possiamo quindi scrivere questa trasformazione usando la forma generale:

$$\begin{aligned} |0\rangle|E\rangle &\mapsto \beta_1|0\rangle|E_1\rangle + \beta_2|1\rangle|E_2\rangle \\ |1\rangle|E\rangle &\mapsto \beta_3|1\rangle|E_3\rangle + \beta_4|1\rangle|E_4\rangle \end{aligned}$$

Non sappiamo cosa siano β_i e E_i . Dovremmo quindi, a partire da questa situazione, trovare e correggere gli errori, che sembra alquanto difficile.

Ovviamente contano anche le amplitudini del sistema, quindi altri due fattori (α_0, α_1) che influenzano il tutto (considera le superposizioni ad esempio):

$$(\alpha_0|0\rangle + \alpha_1|1\rangle)|E\rangle \mapsto \alpha_0\beta_1|0\rangle|E_1\rangle + \alpha_0\beta_2|1\rangle|E_2\rangle + \alpha_1\beta_3|1\rangle|E_3\rangle + \alpha_1\beta_4|1\rangle|E_4\rangle$$

Attraverso l'uso dell'algebra, possiamo semplificare il tutto:

$$\begin{aligned} \alpha_0\beta_1|0\rangle|E_1\rangle + \alpha_0\beta_2|1\rangle|E_2\rangle + \alpha_1\beta_3|1\rangle|E_3\rangle + \alpha_1\beta_4|1\rangle|E_4\rangle = \\ \frac{1}{2}(\alpha_0|0\rangle + \alpha_1|1\rangle)(\beta_1|E_1\rangle + \beta_3|E_3\rangle) + \\ \frac{1}{2}(\alpha_0|0\rangle - \alpha_1|1\rangle)(\beta_1|E_1\rangle - \beta_3|E_3\rangle) + \\ \frac{1}{2}(\alpha_0|1\rangle + \alpha_1|0\rangle)(\beta_2|E_2\rangle + \beta_4|E_4\rangle) + \\ \frac{1}{2}(\alpha_0|1\rangle - \alpha_1|0\rangle)(\beta_2|E_2\rangle - \beta_4|E_4\rangle) \end{aligned}$$

Dividiamo la parte del sistema da quella ambientale. In questo modo, vediamo che sul lato a sinistra (del sistema) non compare nessun β , quindi l'errore non dipende dalla parte di β .

Supponiamo di avere $|\Psi\rangle = \alpha|0\rangle + \alpha_1|1\rangle$, allora possiamo scrivere le quattro forme a sinistra (quelle che riguardano il sistema) in questo modo:

$$\begin{aligned} \alpha_0|0\rangle + \alpha_1|1\rangle &= I|\Psi\rangle \\ \alpha_0|0\rangle - \alpha_1|1\rangle &= Z|\Psi\rangle \\ \alpha_0|1\rangle + \alpha_1|0\rangle &= X|\Psi\rangle \\ \alpha_0|1\rangle - \alpha_1|0\rangle &= XZ|\Psi\rangle \end{aligned}$$

Nota: Tutte le trasformazioni lineari possono essere viste come una combinazione lineare sulla base $\{X, Z, I\}$. Come abbiamo visto, l'esempio di prima del bit flipping é il terzo caso, ovvero la X .

Possiamo quindi ora scrivere l'**effetto** degli errori (noise, bit flipping, qualsiasi) usando i gate appena scritti:

$$\begin{aligned} |\Psi\rangle|E\rangle \mapsto &\frac{1}{2}|\Psi\rangle(\beta_1|E_1\rangle + \beta_3|E_3\rangle) + \\ &\frac{1}{2}(Z|\Psi\rangle)(\beta_1|E_1\rangle - \beta_3|E_3\rangle) + \\ &\frac{1}{2}(X|\Psi\rangle)(\beta_2|E_2\rangle + \beta_4|E_4\rangle) + \\ &\frac{1}{2}(XZ|\Psi\rangle)(\beta_2|E_2\rangle - \beta_4|E_4\rangle) \end{aligned}$$

Il caso degli errori di **bit flip** puó essere visto come un **caso specifico** del general error model, ovvero quello in cui:

$$\beta_1|E_1\rangle = \beta_3|E_3\rangle \quad \beta_2|E_2\rangle = \beta_4|E_4\rangle$$

In questo modo cancelliamo i termini dei casi di Z e XZ . Ovviamente la probabilitá che un errore di bit flip (X -error) non é per forza 50%, ma dipende dagli altri coefficienti (β_i) e dagli stati $|E_i\rangle$.

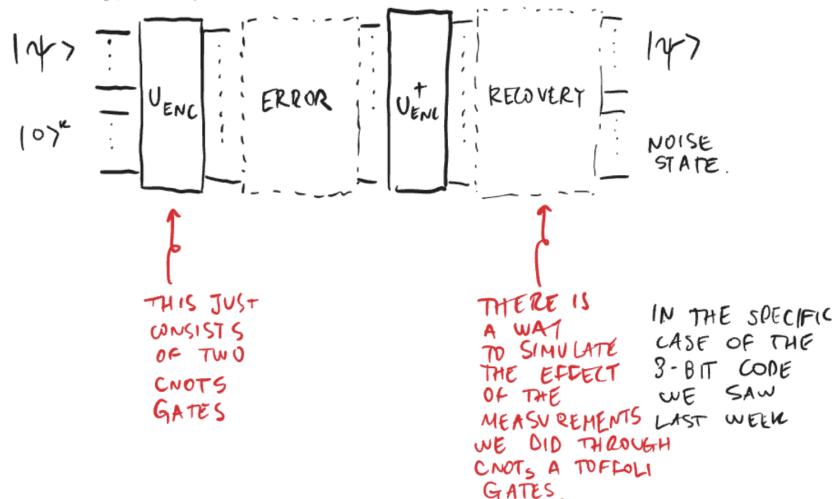
Abbiamo quindi visto il caso di un solo qubit.

Tutto questo puó essere generalizzato a sistemi con più di un qubit. Tuttavia, richiede matrici di densità, superoperatori e altro. Dunque non lo vediamo.

Però, la definizione di quello che un QECC dovrebbe fare è semplice:

Per quanto riguarda invece gli errori di **phase-flip** errors (SOLO phase flip, non combinazione), possono essere scritti in questo modo:

$$|\Psi\rangle|E\rangle \mapsto \frac{1}{2}|\Psi\rangle(\beta_1|E_1\rangle + \beta_3|E_3\rangle) + \frac{1}{2}(Z|\Psi\rangle)(\beta_1|E_1\rangle - \beta_3|E_3\rangle)$$



Possiamo trasformare l'errore di fase in un errore di bit flip **facilmente**. Un errore di fase può essere visto semplicemente come un bit flip error sulla base Hadamard.

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

L'effetto di Z è trasformare $|+\rangle$ in $|-\rangle$ e viceversa.

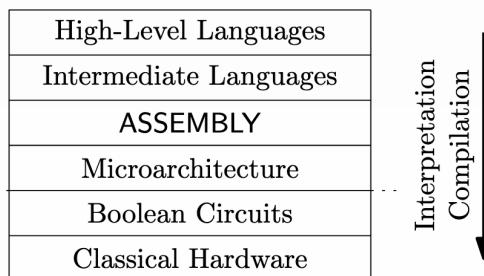
Quindi tutto si può ridurre ad un problema di encoding:

$$|0\rangle \mapsto |+\rangle|+\rangle|+\rangle$$

$$|1\rangle \mapsto |-\rangle|-\rangle|-\rangle$$

8 Quantum programming Languages

Nei sistemi classici, la struttura è la seguente:



I primi due livelli, linguaggi intermedi (JVM,...) e alto livello (C, JAVA, Python, ...), sono ovviamente indipendenti dalla macchina, c'è astrazione. Questo ovviamente ha portato alla creazione di diversi paradigmi di programmazione: logico, imperativo, funzionale, ...

Dal livello assembly, invece tutto è **architecture-dependent**.

L'approccio main stream ovviamente è la creazione di layer che nascondono le implementazioni.

Vediamo ora il **quantum**. Non esiste uno stack come quello che usiamo nella computazione classica. In ogni caso, ci servono entrambe le parti della computazione, sia la parte classica sia quella quantica. Ovvero, servirebbe un paradigma che permetta di usare i diversi circuiti, classici e quantici. Questo ovviamente con un certo livello di astrazione (ad esempio dobbiamo essere noi programmati a decidere quale circuito o un middleware che lo gestisce?)

Una delle idee è stato il **modello QRAM**. Cercava di replicare una variazione dell'architettura di Von-neumann. L'idea principale è che ci sono due parti:

High-Level Languages	
:	
Boolean Circuits	Quantum Circuits
Classical Hardware	Quantum Hardware

- Classical Control: le operazioni che eseguiamo sui qubit non sono in superposizione (ad esempio i vari gate).
- Quantum Store: memoria che può essere usata per creare nuovi qubit su cui applicare poi le trasformazioni.

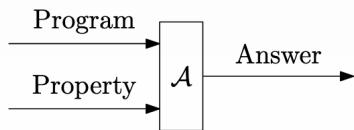
C'è stata anche un'idea di usare un Quantum Control ma ha avuto poco successo (nessun linguaggio implementato effettivamente).

Tuttavia, i "linguaggi" più utilizzati sono i **Quantum Circuit Description Languages (QCDL)**.

Il linguaggio è **puramente classico** e l'idea è integrare delle librerie nei linguaggi classici con cui manipolare circuiti come qualsiasi altra struttura dati, che poi può essere eseguito da hardware quantistico o simulato. Un esempio è **Qiskit**.

9 Quantum Program and System Verification

Classic Program Verification Siamo interessati a vedere se un programma soddisfa una proprietà:



La proprietà di solito specifica un comportamento del programma, in forma *funzionale* (*cosa* il programma dovrebbe fare)³³ o *non-funzionale* (*come* il programma garantisce un certo servizio o meno).

La risposta rientra in SI, NO o UNKNOWN. La verifica di sistemi può essere fatta ovviamente **solo** se la proprietà si preserva durante la compilazione o l'interpretazione (spesso le proprietà funzionali sono preservate, quelle non funzionali no).

Alcuni esempi di verifica di sistemi classica:

- **Program Logics.** Usata per programmi imperativi. Si dimostra, in modo induttivo, la validità di triple della forma $\vdash \{\Phi\}P\{\Psi\}$. Questa tripla è valida se usando la precondizione (input) Φ eseguendo P riusciamo ad arrivare in uno stato finale che rispetta la postcondizione (output) Ψ . Spesso è usato per proprietà funzionali, ma può essere adattato per quelle non funzionali.
- **Model Checking.** Si vede se in un sistema (visto come una struttura logica) una determinata formula vale ($S \models A$). Visto in Logica.
- **Type Systems.** Supportato da molti linguaggi funzionali. Si effettuano dimostrazioni del tipo $\Gamma \vdash P : A$, dove A può essere non solo un tipo base, ma anche ad esempio funzione. È usato per garantire **safety**, ma può essere generalizzato ad altre tipi di proprietà.

Quantum Verification Verificare sistemi Quantici è molto difficile e spesso non si possono nemmeno testare. Può essere molto dispendioso in quanto serve innanzitutto l'hardware, ed eseguirlo costa. Si possono usare le simulazioni, ma in molti casi richiederebbero tempo di esecuzione troppo elevato.

Molti cercano di usare lo stesso approccio di verifica di sistemi per i sistemi quantici sia per studiare proprietà funzionali sia non funzionali. Tuttavia:

- Il modello computazionale è diverso. Basta pensare al fatto che è probabilistico.
- Oltre al blowup esponenziale che è già un problema nella verifica di sistemi, c'è anche il problema delle superposizioni! Un solo stato può essere già esponenziale.
- Algoritmi quantici sono corretti solo **approssimativamente**. Ovvero c'è sempre una piccola percentuale di errore.

³³Teorema di Rice: problemi indecibili. Sono quelli funzionali.

A Applicazione Matrici Unitarie

Quale output hanno i seguenti circuiti?

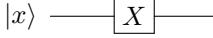
1.



soluzione:

$$Z|+\rangle = Z \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \right) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} = |- \rangle$$

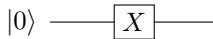
2.



soluzione:

$$X|+\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = |+\rangle$$

3.



soluzione:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

4.



soluzione:

$$Y|1\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -i \\ 0 \end{bmatrix}$$

Nota: essendo tutto lineare, posso anche applicare direttamente l'operatore sui singoli componenti e poi applicare le costanti. Ad esempio $X|+\rangle$ può essere risolto in questo modo:

$$X|+\rangle = X \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \right) = \frac{1}{\sqrt{2}}X|0\rangle + \frac{1}{\sqrt{2}}X|1\rangle = \dots$$

B Misurazione qubit

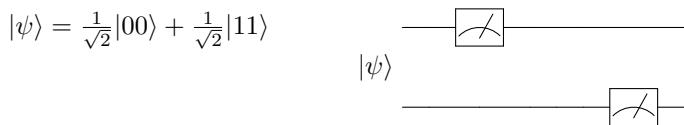
1. Misuriamo $|1\rangle$ nella base $\{|+\rangle, |-\rangle\}$:

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \alpha_0 \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} + \alpha_1 \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\begin{cases} \alpha_0 \frac{1}{\sqrt{2}} + \alpha_1 \frac{1}{\sqrt{2}} = 0 \\ \alpha_0 \frac{1}{\sqrt{2}} - \alpha_1 \frac{1}{\sqrt{2}} = 1 \end{cases} \implies \begin{cases} \alpha_0 = \frac{1}{\sqrt{2}} \\ \alpha_1 = -\frac{1}{\sqrt{2}} \end{cases} = |+\rangle$$

Questo significa che il risultato $\frac{1}{\sqrt{2}}$ uscirà con probabilità $\left| \frac{1}{\sqrt{2}} \right|^2$, ovvero il 50% .

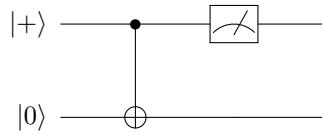
2. Misuriamo il sistema:



$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Quindi il primo bit sarà 0 al 50% e 1 al 50%. Se sarà 0, allora il sistema collassa a $|0\rangle \otimes |0\rangle = |00\rangle$; altrimenti il sistema collassa a $|1\rangle \otimes |1\rangle = |11\rangle$

3. Misuriamo il sistema:



$$\begin{aligned}CNOT|\psi\rangle &= CNOT|+\rangle|0\rangle = CNOT\left(\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes |0\rangle\right) = CNOT\left(\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle\right) = \\&= \frac{1}{\sqrt{2}}CNOT|00\rangle + \frac{1}{\sqrt{2}}|10\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle\end{aligned}$$

Misurando successivamente come nell'esercizio due avrò $|00\rangle$ al 50% e $|11\rangle$ con probabilitá al 50%