Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea in Fisica

MACCHINA DI TURING: MODELLO CLASSICO E QUANTISTICO

Relatore:
Prof.ssa Elisa Ercolessi

Presentata da:
Aldo Perri

Anno Accademico 2020/2021

Abstract

In questo elaborato viene discusso il modello teorico computazionale conosciuto come macchina di Turing, formulato nella sua versione classica da Alan Turing nel 1936, dando il via alla moderna teoria computazionale. Viene esposta anche la sua versione nel contesto della teoria quantistica, seguendo principalmente il lavoro del fisico David Deutsch che nel 1985 formalizza la macchina di Turing quantistica, primo modello computazionale basato sui risultati della meccanica quantistica. Si analizzano le principali proprietà di un calcolatore quantistico universale, mettendo in mostra come sia possibile ottenere generazioni di numeri casuali, simulare sistemi fisici arbitrari e sfruttare il principio di sovrapposizione degli stati per effettuare computazioni parallele.

Indice

Introduzione **3**

1 La macchina di Turing classica **5**
1.1 Descrizione informale 5
1.1.1 Esempio: funzione costante 7
1.2 Macchine di Turing deterministiche 8
1.3 Macchina di Turing Universale 9
1.4 Problemi computazionali e tesi di Church-Turing 12
1.5 Indecibilità: problemi insolubili 14
1.6 Macchine non deterministiche 18

2 La macchina di Turing Quantistica **20**
2.1 Interpretazione fisica della tesi di Church-Turing 20
2.2 Qubit 22
2.3 Calcolatore quantistico 23
2.4 Macchina quantistica Universale 26

3 Proprietà del computer quantistico universale **32**
3.1 Numeri casuali 32
3.2 Simulazione di sistemi fisici arbitrari finiti 33
3.3 Parallelismo quantistico 35
3.3.1 Algoritmo di Deutsch 37

Conclusioni **39**

A Matrici densità **44**

B Sistemi aperti **47**

Introduzione

Nel corso del ventesimo secolo assistiamo a due grandissime rivoluzioni concettuali: la formulazione della meccanica quantistica e la nascita del computer. La moderna teoria computazionale nasce nel 1936, anno in cui il matematico Alan Turing formalizza idealmente quello che oggi chiamiamo *computer programmabile* [10] attraverso la formulazione della cosiddetta macchina di Turing. La caratteristica principale di questo modello computazionale è l'universalità: Turing è in grado di dimostrare l'esistenza di una macchina universale in grado di simulare qualsiasi altra macchina di Turing.
Con la *tesi di Church-Turing* ci si spinge ancora oltre, arrivando ad affermare che una qualsiasi computazione è fisicamente realizzabile se e solo è eseguibile da una macchina di Turing universale: tale modello, dunque, cattura completamente la nozione di computabilità e di algoritmo.
Negli anni '40 John Von Neumann sviluppa un modello teorico per realizzare concretamente un dispositivo programmabile in grado di funzionare come una macchina di Turing universale [10]: nascono i primi computer. Un grandissimo passo avanti viene fatto nel 1947 con la realizzazione dei primi transistor, che diventeranno la base per l'hardware dei computer digitali. Da allora l'evoluzione dei componenti hardware ha fatto grandi passi in avanti: attraverso la miniaturizzazione delle componenti costituenti i circuiti integrati, oggi siamo in grado di ottenere processori sempre più piccoli e sempre più potenti. A tal proposito nel 1965 viene enunciata la *legge di Moore*: tale legge empirica afferma che all'incirca ogni due anni la potenza di calcolo dei computer raddoppia. Tuttavia, accedendo a scale di grandezza sempre più piccole, gli effetti quantistici iniziano a interferire con i processi computazionali. Questa considerazione di carattere tecnologico può essere vista come prima motivazione per cercare un modello computazionale alternativo basato sui principi della meccanica quantistica.
Un'altra motivazione per un nuovo paradigma computazionale può essere ricercata nell'ambito della simulazione di sistemi fisici. In un articolo del 1982 [6] Richard Feynman dimostra che nessuna macchina di Turing universale può simulare alcuni sistemi fisici senza andare incontro ad un rallentamento esponenziale delle sue prestazioni: nello stesso articolo sostiene la necessità di formulare un modello computazionale quantistico.
Nel 1985 David Deutsch pubblica un articolo [3] in cui formalizza la macchina di Turing quantistica universale, che altro non è che l'analogo quantistico del modello classico di

Turing: questo modello computazionale ha dato il via allo studio della computazione quantistica e al successivo sviluppo di computer quantistici.
Nel primo capitolo di questa tesi viene introdotta la macchina di Turing nella sua formulazione classica, viene discussa la tesi di Church-Turing e si dimostra l'esistenza di problemi computazionali irrisolubili: questo pone un *limite logico* [4] a ciò che può essere calcolato tramite una macchina computazionale.
Nel secondo capitolo si analizza principalmente l'articolo [3] di Deutsch. Prima della descrizione della macchina di Turing quantistica viene discussa la promozione della tesi di Church-Turing a status di principio fisico: in questo modo ogni processo di calcolo viene visto come un esperimento fisico (cioè un processo di misura) e il *limite logico* per la computazione si può pensare anche come un *limite fisico* [4]. Viene dunque descritto il modello quantistico della macchina di Turing attraverso l'introduzione del concetto di qubit.
Nel terzo capitolo vengono esplorate le proprietà principali della macchina di Turing quantistica universale. Viene in particolare trattata la possibilità di generare numeri casuali, simulare sistemi fisici arbitrari e sfruttare la sovrapposizione di stati quantistici per eseguire computazioni parallele.

Capitolo 1

La macchina di Turing classica

Alla base della scienza della computazione troviamo la nozione di algoritmo: una lista di istruzioni da eseguire per portare a termine un compito assegnato. Storicamente la nozione di algoritmo risale all'antichità [10]: lo stesso Euclide formulò un algoritmo in grado di trovare il massimo comune divisore fra due numeri interi. Tuttavia per una descrizione formale del concetto di algoritmo bisogna aspettare fino al 1936, anno in cui vengono pubblicati i lavori indipendenti di Alan Turing e Alonzo Church, pionieri della moderna teoria del computer. Per comprendere i lavori di Church e Turing bisogna fare un passo indietro. Nel 1928 il matematico David Hilbert [10] propone alla comunità scientifica l'*Entscheidungsproblem* (problema della decisione): Hilbert chiede se esiste un algoritmo in grado di risolvere in principio tutti i problemi matematici. Turing e Church riescono a dimostrare che tale problema non ha una risposta affermativa: per dimostrarlo i due studiosi sono costretti a formalizzare rigorosamente e matematicamente la nozione di algoritmo. In particolare Turing definisce una classe di macchine, chiamate macchine di Turing, che si rivelano essere un modello universale di computazione: questo è il punto di partenza per la moderna teoria degli algoritmi e delle scienze computazionali.

1.1 Descrizione informale

Una macchina di Turing può essere descritta in modo informale, dal momento che i meccanismi che regolano il suo agire sono abbastanza elementari. La funzione principale di una macchina di Turing è produrre un output a partire da un particolare input. Come illustrato in fig. 1.1. la macchina ha bisogno di [10]:

1. Un programma: una serie di istruzione che dica alla macchina cosa fare;
2. Un nastro infinito formato da infinite caselle quadrate
3. Una punta mobile che può spostarsi lungo il nastro e che può scrivere o cancellare un simbolo nella casella su cui è posizionata

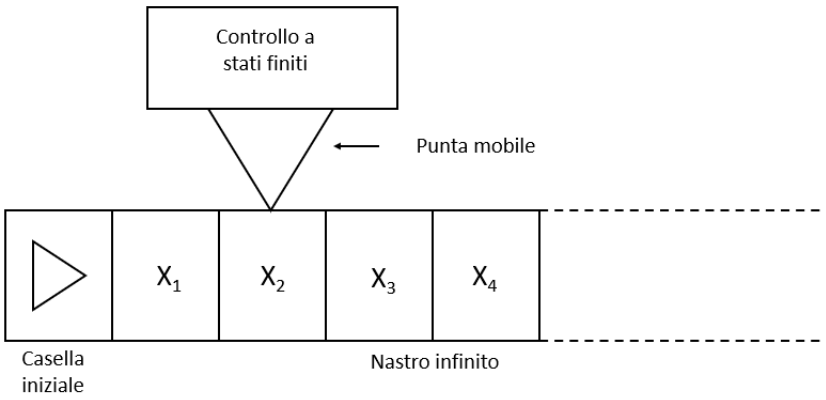

Figura 1.1: Schematizzazione di una macchina di Turing

4. Un'unità di controllo che può memorizzare una collezione finita di *stati interni*

Siano $\{q_0, q_1, \ldots, q_n, q_h\}$ gli stati interni della macchina. Particolare importanza hanno gli stati q_0 e q_h, rispettivamente stato iniziale e stato finale: all'inizio della computazione la macchina si trova nello stato q_0 e l'esecuzione del Programma causerà cambiamenti dello stato interno della macchina fino a che, se la macchina riesce a fornire un output, la computazione avrà fine con la macchina nello stato q_h. Il nastro è infinito lungo una sola direzione, per cui le caselle sono numerate da 0 a infinito. Sulle caselle possono essere presenti simboli di un certo alfabeto predefinito Γ che presenta un numero finito di elementi. Per semplicità possiamo scegliere:

$$\Gamma = \{0, 1, *, \triangleright\}$$

dove $*$ indica una casella vuota, mentre $\triangleright$ è il simbolo riservato alla casella in posizione 0. La punta mobile è in grado di spostarsi lungo il nastro e accedere alle varie caselle quadrate con la possibilità di sovrascriverle. Arrivati a questo punto siamo in grado di dire che una macchina di Turing è una macchina in grado di formulare una stringa in output a partire da una stringa in input contenente un numero variabile di simboli dell'alfabeto prescelto Γ. Ma come fa la macchina a produrre l'output? É fondamentale definire un programma che dica alla macchina che cosa fare. Possiamo formulare un programma come una lista finita di istruzioni nella forma

$$< q, x, q', x', s >$$

dove q e q' indicano due stati possibili della macchina, x e x' sono simboli presi dall'alfabeto Γ, mentre s può assumere i valori -1, 0, +1. Come viene eseguita dunque un'istruzione? Per prima cosa la macchina cerca una linea di programma nella forma

$< q, x, \bullet, \bullet, \bullet >$ con q e x corrispondenti rispettivamente allo stato attuale della macchina e al simbolo presente nella casella sulla quale è posta la punta mobile. A questo punto la macchina cambia il suo stato interno da q a q', sovrascrive x con x' e sposta la punta mobile a destra o sinistra o rimane nella casella corrente in base al valore di s: il valore -1 presuppone uno spostamento verso sinistra, il valore $+1$ verso destra, mentre il valore 0 fa rimanere il cursore nella sua posizione. Se la macchina non riesce a trovare alcuna linea di programma eseguibile nello stato q, si pone nello stato q_h e l'esecuzione ha termine.

1.1.1 Esempio: funzione costante

Vediamo come è possibile utilizzare una macchina di Turing per un semplice compito computazionale come calcolare la banale funzione $f(x) = 1$. Diamo in input alla macchina il numero binario 101101: sul nastro della macchina dunque apparirà la sequenza di input:

$$[\triangleright, 1, 0, 1, 1, 0, 1, *, *, \dots]$$

Siano $\{q_0, q_1, q_2, q_3, q_h\}$ gli stati accessibili alla macchina e utilizziamo l'alfabeto Γ definito in precedenza. Definiamo il programma da far eseguire alla macchina nel seguente modo:

$$\begin{aligned}
1: &\ < q_0, \triangleright, q_1, \triangleright, +1 > \\
2: &\ < q_1, 0, q_1, *, +1 > \\
3: &\ < q_1, 1, q_1, *, +1 > \\
4: &\ < q_1, *, q_2, *, -1 > \\
5: &\ < q_2, *, q_2, *, -1 > \\
6: &\ < q_2, \triangleright, q_3, \triangleright, +1 > \\
7: &\ < q_3, *, q_h, 1, 0 >
\end{aligned}$$

All'inizio dell'esecuzione viene chiaramente eseguita la linea 1 e la macchina entra nello stato q_1. Le linee del programma 2, 3 fanno in modo che la macchina continua ad avanzare sul nastro e sovrascrive gli 0 e 1 che incontra con caselle vuote finchè non raggiunge la prima casella con il simbolo $*$. A questo punto entra nello stato q_2, torna indietro fino alla casella iniziale, entra nello stato q_3 e scrive la cifra 1 nella casella adiacente a quella iniziale e termina la computazione. Alla fine l'output mostrato sarà:

$$[\triangleright, 1, *, * \dots]$$

Possiamo dare alla macchina come input un qualsiasi numero binario al posto di 101101: l'output sarà 1 indipendentemente dall'input scelto.

1.2 Macchine di Turing deterministiche

Cerchiamo ora di dare una definizione rigorosa di macchina di Turing. Per la trattazione seguente seguiremo sia [8] che [14].

Definizione 1.2.1. *Si definisce Macchina di Turing una quintupla* $(K, \Gamma, \delta, s, H)$ *dove*
K *è una collezione finita di stati interni*
Γ *è una collezione finita di simboli detta* ***alfabeto*** *che include anche i simboli* $*$ *e* $\triangleright$
$s \in K$ *è lo stato iniziale*
$H \subseteq K$ *è una sottofamiglia di stati finali detti* ***stati di halt***
$\delta : (K - H) \times \Gamma \to K \times \Gamma \times \{\leftarrow, \rightarrow, -\}$ *è detta* ***funzione di transizione*** *tale che:*
(a) per ogni $q \in H - K$*, se* $\delta(q, \triangleright) = (q', x', d)$ *allora* $x' = \triangleright$ *e* $d = \rightarrow$
(b)per ogni $q \in H - K$ *e per ogni* $x \in \Gamma$*, se* $\delta(q, x) = (q', x', d)$ *allora* $x' \neq \triangleright$*.*

Possiamo fare subito alcune osservazioni. Notiamo che la funzione δ non è definita su H, per cui se la macchina entra in uno stato finale, non viene eseguita più alcuna azione: in questo caso si dice che la macchina *si arresta*. La condizione (a) obbliga la macchina a spostare il cursore a destra quando la casella corrente è la casella iniziale senza sovrascrivere $\triangleright$, mentre la condizione (b) impedisce alla macchina la scrittura di altri caratteri $\triangleright$. In altre parole $\triangleright$ individua in modo univoco la casella iniziale e evita che la macchina sposti il cursore oltre la fine del nastro. La funzione δ formalizza la nozione precedente di programma: determina in modo univoco l'azione della macchina una volta specificato lo stato corrente e il simbolo che viene letto. Per questo motivo si parla di *macchine deterministiche*.
Affinchè la macchina possa arrestarsi è sufficiente definire un solo stato di halt h. In alcune applicazioni però vogliamo che la macchina possa rispondere in modo affermativo o negativo a un dato problema decisionale e a tal fine definiamo due stati di halt aggiuntivi che chiamiamo y e n. In questo modo il set di stati finali è dato da:

$$H = \{h, y, n\} \tag{1.1}$$

In particolare se lo stato di halt della macchina è y oppure n diremo che l'output prodotto è "Si" oppure "No"; se lo stato di halt è h l'output prodotto dalla macchina sarà la stringa presente sul nastro al termine della computazione.
Formalizziamo ora l'operato di una macchina di Turing deterministica.

Definizione 1.2.2. *Indichiamo con* Γ^* *l'insieme di tutte le sequenze finite di simboli appartenenti a* Γ*. Sia* $a \in \Gamma^*$*. Indichiamo gli elementi di* a *con* $a_0, a_1, \ldots, a_{n-1}$ *con* $n = |a|$ *la lunghezza della stringa* a*.*
Si dice ***configurazione*** *ad un dato istante di tempo di una macchina di Turing una tripletta ordinata* $(a, q, k) \in \Gamma^* \times K \times \mathbb{N}$*, con* a *che indica la stringa presente sul nastro,* q *è lo stato corrente della macchina e* k *individua la posizione della casella in lettura.*

Poniamo alcune restrizioni sulle configurazioni accettabili: la posizione k deve essere tale per cui $0 \leq k < |a|$ e le stringhe accettabili devono iniziare con $\triangleright$ e terminare con $*$.

Sia M una macchina di Turing nella configurazione (a, q, k) a un dato istante di tempo. Come determinare la configurazione (a', q', k') della macchina all'istante di tempo successivo? Applichiamo la funzione di transizione e otteniamo $\delta(q, a_k) = (p, b_k, d)$. Risulta perciò naturale identificare q' con p, a' con la stringa a con il simbolo b_k al posto di a_k e k' con $k-1, k, k+1$ rispettivamente se $d = \leftarrow, -, \rightarrow$. Se due configurazioni (a, q, k) e (a', q', k') sono tali da rispettare tali condizioni allora scriveremo:

$$(a, q, k) \quad \xrightarrow{M} \quad (a', q', k')$$

Definizione 1.2.3. *Definiamo* ***computazione*** *di una macchina di Turing una sequenza di configurazioni* (a_i, q_i, k_i) *con* $i = 0, \ldots, T$ *tali che:*

1. *la macchina inizia da una valida configurazione iniziale, ovvero* $q_0 = s$ *e* $k_0 = 0$*;*
2. *ogni coppia di configurazioni successive devono essere valide, ovvero per ogni* $i = 0, \ldots, T-1$ *si ha che* $(a_i, q_i, k_i) \xrightarrow{M} (a_{i+1}, q_{i+1}, k_{i+1})$*;*
3. *se* $T = \infty$ *allora si dice che la computazione non ha termine*
4. *se* $T < \infty$ *allora* $q_T \in H$ *e la macchina si arresta.*

Possiamo definire anche macchine di Turing multinastro in modo molto simile alla definizione 1.2.1: in questo caso una macchina a k nastri agisce su k stringhe simultaneamente e in modo indipendente e la funzione di transizione dipende dai k simboli letti simultanemante in un dato istante di tempo. Si può comunque mostrare che una macchina di Turing a singolo nastro può eseguire qualsiasi computazione eseguibile su una macchina multinastro, con lo svantaggio di impiegare più tempo[8][14]. Già da questo fatto intravediamo una caratteristica di fondamentale importanza: l'universalità delle macchine di Turing.

1.3 Macchina di Turing Universale

La caratteristica principale delle macchine di Turing che vogliamo mostrare è la loro *universalità*: possiamo costruire un'unica macchina in grado di simulare qualsiasi altra macchina di Turing. Chiameremo tale macchina *Macchina di Turing Universale*. Per poter definire una tale macchina, dobbiamo inanzitutto capire come dare in input a una macchina la *descrizione* di un'altra.

Sia M una macchina di Turing con un alfabeto Γ e insieme di stati interni K. Poniamo

$$l = \lceil \log_2(|\Gamma| + |K| + 3) \rceil$$

e assumiamo che qualsiasi elemento appartenente a $\Gamma \bigcup K \bigcup \{\leftarrow, -, \rightarrow\}$ venga identificato univocamente da una stringa binaria in $\{0,1\}^l$. Mostriamo come effettuare questa codifica. Sapendo che Γ contiene un numero finito di elementi, possiamo individuare ciascun simbolo dell'alfabeto con un numero naturale da 0 a $|\Gamma| - 1$ ed esprimendo tale numero in binario utilizzando l bit otteniamo la codifica del carattere in l bit. Facciamo la stessa cosa per il set di stati K, identificando ciascuno stato con un numero naturale compreso tra $|\Gamma|$ e $|\Gamma| + |K| - 1$, mentre gli elementi del set $\{\leftarrow, -, \rightarrow\}$ vengono identificati con i numeri $|\Gamma| + |K|$, $|\Gamma| + |K| + 1$ e $|\Gamma| + |K| + 2$. Convertendo poi i numeri in binario con l bit otteniamo anche la codifica di tali elementi. In questo modo ciascun elemento di $\Gamma \bigcup K \bigcup \{\leftarrow, -, \rightarrow\}$ è identificato univocamente da una stringa a l bit. Possiamo anche codificare le regole di transizione che specificano la funzione di transizione della macchina. Ad esempio possiamo codificare la regola $\delta(q, \sigma) = (p, \tau, d)$ con la stringa $\delta = (q, \sigma, p, \tau, d)$ in cui ogni parametro è codificato in l bit come apena descritto. Siamo ora in grado di definire formalmente la descrizione di una macchina di Turing[14].

Definizione 1.3.1 (Descrizione)**.** *La **descrizione** di una macchina di Turing M è una stringa finita dell'alfabeto {0, 1, '(' , ')', ',' } definita come*

$$M = m, n, \delta_1 \delta_2 \dots \delta_N$$

dove m e n sono rispettivamente i numeri $|\Gamma|$ e $|K|$ scritti in binario e ciascuna δ_i è una stringa codificante una delle regole di transizione che definiscono la funzione di transizione δ. Tali regole vengono codificate nella stringa

$$\delta_i = (q, \sigma, p, \tau, d)$$

dove ciascuno dei cinque parametri è una stringa in $\{0,1\}^l$ codificante un elemento di $\Gamma \bigcup K \bigcup \{\leftarrow, -, \rightarrow\}$.

Per come abbiamo definito la descrizione, essa individua in modo univoco una macchina di Turing, ovvero attraverso le descrizioni siamo in grado di codificare qualsiasi macchina di Turing. Siamo in grado ora di dare la seguente:

Definizione 1.3.2 (Macchina di Turing Universale)**.** *Una **macchina di Turing universale** è una macchina di Turing U con l'alfabeto $\Gamma_U = \{0, 1, "(", ")", ",", ";"\}$ che prende input nella forma $\triangleright M; x*$, con M una valida descrizione di una macchina di Turing e x una stringa dell'alfabeto di M codificata con blocchi a l-bit.*

La computazione di U sull'input $\triangleright M; x*$ termina nello stesso stato con cui termina la computazione di M sull'input x e inoltre l'output fornito da U equivale all'output fornito da M codificato in stringhe a l-bit.
Possiamo implementare una macchina Universale U tramite la costruzione di una macchina di Turing a 5 nastri [14]:

- nastro di input: contiene l'input che non viene mai sovrascritto;
- nastro descrittivo: contiene la descrizione della macchina M che non viene mai sovrascritta;
- nastro di lavoro: il contenuto è lo stesso del nastro di M codificato in blocchi a l-bit;
- nastro di stato: descrive lo stato corrente di M codificato in una stringa a l-bit
- nastro speciale: contiene la codifica degli stati di Halt h, y, n e dei simboli direzionali.

Specificare in modo formale l'operato di questa macchina è troppo complicato [14], per cui proveremo a dare una spiegazione qualitativa del suo funzionamento.
La macchina inizia in una fase di inizializzazione, durante la quale vengono eseguite le seguenti azioni:

1. La macchina copia la descrizione M dal nastro di input al nastro descrittivo;
2. Viene copiata la stringa x sul nastro di lavoro, inserendo virgole tra ogni blocco a l-bit codificante un carattere di x;
3. Lo stato iniziale di M viene copiato sul nastro di stato;
4. Vengono codificati sul nastro speciale gli stati di halt e i simboli direzionali;
5. Su ogni nastro il cursore si posizione sulla casella iniziale;

Una volta terminata la fase di inizializzazione, la macchina è pronta per la simulazione: per farlo esegue una serie di azioni ciclicamente. Ogni iterazione del ciclo corrisponde ad uno step computazionale della macchina simulata. All'inizio di ogni iterazione sul nastro di lavoro e sul nastro di stato sono codificati rispettivamente il contenuto del nastro di M ed il suo stato all'inizio del corrispondente step computazionale; il cursore sul nastro di lavoro punta è puntato sulla virgola che precede la codifica binaria del simbolo letto correntemente da M, mentre su tutti gli altri nastri il cursore è sulla casella iniziale. Ognivolta che U inizia un'iterazione controlla se si è arrivati alla fine della computazione: per fare ciò U confronta il contenuto del nastro di stato con il contenuto del nastro speciale codificante gli stati di halt. Se U non trova alcuna corrispondenza procede con la simulazione dello step computazionale di M: la macchina universale inizia a leggere il segmento del nastro descrittivo che codifica le regole di transizione. Ogni volta che viene letta la coppia (q, σ) la macchina verifica se q corrisponde allo stato presente sul nastro di stato e se la stringa σ è identica alla stringa del blocco a l-bit corrente sul nastro di lavoro. Quando la verifica va a buon fine la macchina procede con la lettura

dei corrispondenti (p, τ, d) e sovrascrive q con p sul nastro di stato, σ con τ sul nastro di lavoro e sposta il cursore come sul nastro di lavoro come specificato da d. Tutto questo viene ripetuto finchè sul nastro di stato non è presente uno stato di halt: a questo punto U si arresta e come output otteniamo lo stesso risultato che avremmo ottenuto dalla computazione di M.
Quello appena descritto è solo un esempio di una possibile macchina di Turing universale, ad esempio si poteva definire una macchina universale con un numero minore di nastri. Tuttavia nella sezione precedente abbiamo affermato che qualsiasi macchina di Turing multinastro è equivalente dal punto di vista computazionale a una macchina a singolo nastro, per cui in principio è possibile costruire una macchina universale con un solo nastro.
Abbiamo mostrato che è possibile costruire una singola macchina di Turing che possa simulare tutte le macchine che possiamo definire.

1.4 Problemi computazionali e tesi di Church-Turing

Cerchiamo ora di determinare le condizioni affinchè una macchina di Turing riesca a portare a termine un dato compito, ovvero cerchiamo di caratterizzare i problemi risolvibili secondo Turing e cerchiamo di capire se esistono problemi insolubili. Principalmente vogliamo che il modello computazionale della macchina di Turing ci permetta di risolvere due classi di problemi:

1. Problemi decisionali: sono problemi che ammettono come soluzione una risposta binaria (si-no), ad esempio *"il numero decimale X è divisibile per 17?"*;

2. Problemi di calcolo: sono problemi che richiedono come output una risposta numerica, ad esempio *"calcola la funzione* $x^2 + 1$ *nel punto* $x = 3$ *"*.

Per formalizzare il concetto di problema decisionale definiremo tali problemi in termini di *linguaggi*, mentre per caratterizzare i problemi di calcolo daremo la definizione di *funzione computabile* [8].
Sia $M = (K, \Gamma, \delta, s, H)$ una macchina di Turing, con $H = \{y, n, h\}$ tali che:

- se lo stato finale della macchina è y (n) allora diremo che l'output della macchina è "si" ("no");

- se lo stato finale della macchina è h allora l'output prodotto è la stringa presente sul nastro al termine della computazione.

Definizione 1.4.1. *Sia* $\Gamma_0 = \Gamma \setminus \{\triangleright, *\}$. *Chiamiamo* ***linguaggio*** *qualsiasi insieme* $L \subseteq \Gamma_0^*$. *Inoltre diremo che:*

1. *M* ***decide*** *L se qualsiasi computazione di M termina nello stato y o nello stato n e L è l'insieme di stringhe che date in input portano allo stato y;*

2. *L è decidibile se esiste M che decide L;*

3. *M* ***semi-decide*** *L se L è l'insieme di stringhe presenti nella configurazione iniziale che portano M ad arrestarsi;*

4. *L è* ***enumerabile ricorsivamente*** *se esiste una macchina M che semi-decide L;*

Una macchina decide un linguaggio L quando riconosce se un dato input appartiene al linguaggio oppure no: nel primo caso si arresta nello stato y, nel secondo si arresta nello stato n. Una macchina semidecide L quando si ha la seguente condizione: la macchina si arresta se e solo se la stringa in input appartiene al linguaggio, in caso contrario non termina mai la propria esecuzione e non riesce a fornire un output. Ad esempio per il problema *"il numero X è divisibile per 17?"* *L* è costituito dai numeri interi divisibili per 17. Se siamo in grado di costruire una macchina che decide L, allora la macchina riesce determinare se un dato numero in input X è o non è divisibile per 17 e lo comunica all'utente. Se invece costruiamo una macchina che semidecide L, allora la macchina si arresta se e solo se l'input fornito è divisibile per 17, in caso contrario continua la computazione per un tempo infinito.
Diamo ora la definizione di *funzione computabile*:

Definizione 1.4.2. *Sia* $f : \Gamma_0^* \to \Gamma_0^*$. *Si dice che M* ***computa*** *f se per ogni* $x \in \Gamma_0^*$ *ogni computazione con configurazione iniziale* $(\triangleright x*, s, 0)$ *termina nella configurazione* $(\triangleright f(x)*, h, 0)$. *f si dice* ***computabile*** *se esiste M che computa f.*

Tali definizioni ci permettono di identificare una macchina di Turing che riconosce un linguaggio L o che computa una funzione f con un *algoritmo* che esegue correttamente e in modo affidabile un certo compito computazionale. D'altra parte non possiamo dire lo stesso per una macchina che semiriconosce L: infatti, come abbiamo visto, se l'input non appartiene a tale linguaggio la macchina non terminerà mai la propria computazione e in tal caso non riusciremo mai a capire se *effettivamente* la macchina non si arresterà mai o se non è ancora passato un tempo sufficiente per completare la computazione. Si può enunciare il seguente teorema[8]:

Teorema 1.4.1. *Sia L un linguaggio decidibile. Allora L è enumerabile ricorsivamente.*

Dimostrazione. Sia $M = (K, \Gamma, \delta, s, \{y, n, h\})$ una macchina di Turing che decide L. Possiamo costruire una macchina di Turing M' che semidecide L togliendo lo stato n dal set di stati di halt e introducendo la regola di transizione $\delta(n, a) = (n, a, -)$. In questo modo M' si arresta se e solo se l'input appartiene a L. □

Abbiamo dimostrato che possiamo trasformare una macchina in grado di decidere un linguaggio in una macchina che semidecide lo stesso linguaggio. Più interessante e difficile[8] è il problema opposto: un linguaggio enumerabile ricorsivamente è sempre decidibile? Mostreremo nel prossimo paragrafo che ciò non è vero.
Mostriamo un'altra importante carratteristica dei linguaggi decidibili:

Teorema 1.4.2. *Sia L un linguaggio decidibile e sia $\bar{L}$ il complementare di L. Allora anche $\bar{L}$ è decidibile.*

Dimostrazione. Sia $M = (K, \Gamma, \delta, s, \{y, n, h\})$ una macchina di Turing che decide L. Costruiamo $M' = (K, \Gamma, \delta', s, \{y, n, h\})$ con

$$\delta'(q,a) = \begin{cases} (n, a, -) & \text{se } \delta(q,a) = (y, a, -) \\ (y, a, -) & \text{se } \delta(q,a) = (n, a, -) \\ \delta(q,a) & \text{in tutti gli altri casi} \end{cases} \tag{1.2}$$

In questo modo M' si arresta in y (n) se M si arresta in n (y) dato lo stesso input. □

Siamo giunti a formalizzare la nozione intutitiva di algoritmo attraverso la definizione di macchine di Turing in grado di arrestarsi su qualsiasi tipo di input. Tale equivalenza fra un concetto matematico rigoroso e un concetto puramente intuitivo è alla base della *tesi di Church-Turing*, che può essere enunciata come segue: *"tutto ciò che è computabile è computabile da una macchina di Turing"* [8] oppure *"se un problema è umanamente calcolabile allora esiste una macchina di Turing in grado di risolverlo"* [10]. La tesi di Church-Turing è matematicamente indimostrabile, in quanto non è un vero teorema, ma solo un ipotesi. Al giorno d'oggi non si è ancora in grado di provare che tale tesi sia falsa. Inoltre tale assunzione rende esplicita l'universalità del modello di Turing: qualsiasi modello computazionale è equivalente a una macchina di Turing.
Arrivati a questo punto ci chiediamo se esistano problemi insolubili da una macchina di Turing e dunque, in accordo con la tesi di Church-Turing, impossibili da risolvere attraverso qualsiasi altro modello computazionale. La risposta è affermativa.

1.5 Indecibilità: problemi insolubili

Vogliamo ora dimostrare che esistono problemi insolubili per una macchina di Turing, in particolare lo si vuole fare mostrando che esistono "più" problemi che macchine [14].

Definizione 1.5.1 (Insieme numerabile)**.** *Un insieme X è **numerabile** se può essere messo in corrispondenza biunivoca con l'insieme dei numeri naturali.*

Lemma 1.5.1. *Sia Γ un insieme finito. Allora Γ^* è numerabile.*

Dimostrazione. Se $|\Gamma| = 1$ possiamo naturalmente identifcare ogni stringa di Γ^* con la propria lunghezza che permette la costruzione di una corrispondenza con i numeri naturali. Nel caso più generico in cui $\Gamma = \{x_0, x_1, \ldots, x_{k-1}\}$ ogni numero naturale può essere espresso in base k, cioè con una stringa finita di numeri da 0 a $k-1$ identificabili con gli elementi di Γ. In questo modo si costruisce corrispondenza 1-1 tra numeri naturali e elementi di Γ^*. □

Lemma 1.5.2. *Sia X un insieme numerabile e sia 2^X l'insieme delle parti di X. Allora 2^X non è numerabile.*

Dimostrazione. Tale lemma si dimostra attraverso il metodo di diagonalizzazione. Sia $f : X \to 2^X$ e sia $x_0, x_1, x_2, \ldots$ una successione numerabile di elementi di X. Mostriamo che esiste $T \in 2^X$ tale che non esiste alcun $x \in X$ tale che $T = f(x)$. Per fare ciò costruiamo una tabella infinita in cui nella riga i-esima rappresentiamo la funzione $f(x_i)$ con una stringa infinita di 0,1. In particolare nella posizione ij abbiamo un 1 se in $f(x_i)$ compare l'elemento x_j, 0 altrimenti. Un esempio di tale tabella è dato da

	x_0	x_1	x_2	x_3	$\ldots$
$f(x_0)$	0	1	0	1	$\ldots$
$f(x_1)$	1	1	0	0	$\ldots$
$f(x_2)$	0	1	1	0	$\ldots$
$f(x_3)$	1	0	0	1	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

A questo punto prendiamo la diagonale $\{0, 1, 1, 1, \ldots\}$ e sostituiamo gli 0 con gli 1 e viceversa, ottenendo $\{1, 0, 0, 0, \ldots\}$ che codifica un altro elemento di 2^X. Tale elemento può essere definito come

$$T = \{x \in X | x \notin f(x)\}$$

Mostriamo che $T \neq f(x)$ per qualsiasi $x \in X$. Supponendo $T = f(x)$ con $x \in T$ stiamo contraddicendo il fatto che $x \notin f(x)$ per definizione. Inoltre supponendo $T = f(x)$ con $x \notin T$ stiamo violando il fatto che $x \in f(x)$ per ogni $x \notin T$. Perciò non esiste alcun $x \in X$ tale che $T = f(x)$. □

Se applichiamo questi due lemma al nostro caso notiamo che prendendo un alfabeto Γ finito allora l'insieme di tutte le stringhe Γ^* è un insieme numerabile. A questo punto è facile identificare l'insieme dei linguaggi con l'insieme delle parti di Γ^*, pertanto tale insieme è non numerabile. Inoltre l'insieme delle macchine di Turing con alfabeto Γ è insieme numerabile, in quanto ogni macchina è identificabile da una descrizione che è una stringa finita dell'alfabeto $\{0, 1, "(", ")", ",", ")\}$: fissato un alfabeto ci sono più linguaggi che macchine perciò sicuramente vale il seguente:

Teorema 1.5.1. *Per ogni alfabeto finito* Γ *esiste un linguaggio* $L \subseteq \Gamma^*$ *che non è enumerabile ricorsivamente.*

Vediamo un esempio concreto [14]. Sia M una macchina di Turing e sia $L(M)$ l'insieme degli input x tali che

$$L(M) = \{x | M\, si\, arresta\, in\, y\}$$

Supponiamo di avere un alfabeto finito Γ e una mappa Φ dal set $\{0, 1, "(", ")", ",", ")\}$ all'insieme di stringhe di Γ^* di lunghezza fissata, in modo tale che la macchina M abbia una descrizione tradotta in stringhe concatenate di Γ^*. Definiamo ora per ogni $x \in \Gamma^*$ il linguaggio $L(x) \subseteq \Gamma_0^*$ tale che: se x è una descrizione di M allora $L(x) = L(M)$, altrimenti $L(x) = \emptyset$. A questo punto possiamo definire una tabella come quella definita nella dimostrazione del lemma 1.5.2:

	x_0	x_1	x_2	x_3	...
$L(x_0)$	0	1	0	1	...
$L(x_1)$	1	1	0	0	...
$L(x_2)$	0	1	1	0	...
$L(x_3)$	1	0	0	1	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

in cui la casella ij contiene un 1 se $x_j \in L(x_i)$, 0 altrimenti. Ora possiamo prendere la diagonale e sostituire gli 0 con gli 1 e viceversa ottenendo il linguaggio diagonale D:

$$D = \{x \in \Gamma_0^* | x \notin L(x)\}$$

che per il lemma 1.5.2 non è in corrispondenza con alcun elemento di Γ_0^*, ovvero non esiste alcun x tale per cui $D = L(x)$. Perciò per definizione D non è un linguaggio ricorsivamente enumerabile. Abbiamo dunque dimostrato il seguente

Teorema 1.5.2. *Il linguaggio diagonale D non è ricorsivamente enumerabile.*

Tale teorema comporta due importanti corollari [14]:

Corollario 1.5.1. *Esiste un linguaggio L ricorsivamente enumerabile il cui linguaggio complementare non lo è.*

Dimostrazione. Sia L il linguaggio complementare del linguaggio diagonale D. Avremo dunque L = L(x), che per definizione è ricorsivamente enumerabile. □

Corollario 1.5.2. *Esiste un linguaggio L ricorsivamente enumerabile che non è decidibile.*

Dimostrazione. Abbiamo visto che il linguaggio complementare di un linguaggio decidibile è decidibile. Per il corollario 1.5.2 sappiamo che esistono linguaggi L il cui complementare non è nemmeno ricorsivamente enumerabile. Dunque tali linguaggi L non sono decidibili. □

Abbiamo dimostrato che la classe dei linguaggi decidibili forma un sottoinsieme stretto della classe dei linguaggi ricorsivamente enumerabili.
Possiamo applicare questi risultati al *problema dell'arresto*

Definizione 1.5.2. *Si dice* ***problema dell'arresto*** *il problema di decidere se una data macchina di Turing M si arresta dato un qualsiasi input x.*

In base a quanto visto finora tale problema ha soluzione se esiste una macchina di Turing che riesca a decidere il linguaggio H definito come

$$H = \{M; x | M \text{ è una valida descrizione di una macchina di Turing,} \\ x \text{ è un input che porta M ad arrestarsi}\}$$

Si ottiene che:

Teorema 1.5.3. *Il problema dell'arresto è irrisolvibile, ovvero non esiste alcun algoritmo in grado di risolverlo.*

Dimostrazione. Procediamo per assurdo. Assumiamo il problema dell'arresto risolvibile, dunque assumiamo che il linguaggio H sia decidibile da una macchina di Turing M. Se H è decidibile allora lo è anche H_1 definito come:

$$H_1 = \{"M" | \text{ la macchina M si arresta con l'input } "M"\}$$

Per il teorema 1.4.2 se H_1 è decidibile allora lo deve essere anche il suo complementare

$$\bar{A}_1 = \{x | x \text{ o non è una descrizione valida di una macchina di Turing} \\ \text{oppure è la descrizione di una macchina di Turing} \\ \text{che non termina sull'input } "M"\}$$

Per definizione $\bar{A}_1$ è il linguaggio diagonale definito in precedenza, che abbiamo visto non essere ricorsivamente enumerabile, quindi nemmeno decidibile. Siamo giunti a una contraddizione, quindi l'ipotesi di partenza è falsa. □

Il problema dell'arresto è del tutto equivalente all'Entscheidungsproblem di Hilbert [10]: Turing utilizzò questi argomenti per dimostrare che il problema di Hilbert è insolubile. Abbiamo dunque posto un limite alla potenzialità della computazione.

1.6 Macchine non deterministiche

Ci possiamo chiedere se possiamo apportare migliorie al nostro modello computazionale inserendo elementi aleatori nel modello e vedere se riusciamo a risolvere problemi insolubili da una macchina deterministica.

Definizione 1.6.1. *A nalogamente al caso deterministico, una macchina di Turing non deterministica è una quintupla* $(K, \Gamma, \Delta, s, H)$ *. L'unica modifica è la sostituzione della funzione di transizione* δ *con la relazione di transizione* Δ*.*

Mentre la funzione di transizione δ è definita in modo tale da mappare univocamente la coppia (q, x) in (q', x', d), la relazione di transizione Δ permette alla macchina di scegliere fra più opzioni in base a una distribuzione di probabilità prescelta. In termini di configurazioni questo vuol dire che, mentre per una macchina deterministica la configurazione ad un dato istante di tempo determina in modo univoco la configurazione all'istante successivo, per una macchina probabilistica si può solo calcolare la probabilità di avere una certa configurazione all'istante di tempo successivo. Per capire se tale impostazione incrementa il potere computazionale della nostra macchina dobbiamo formalizzare anche in questo caso che cosa vuol dire risolvere un problema. Ancora una volta lo faremo in termini di linguaggi [8].

Definizione 1.6.2. *Sia* $M = (K, \Gamma, \Delta, s, H)$ *una macchina di Turing probabilistica.* M accetta *un input* $x \in \Gamma_0^*$ *se la configurazione iniziale può arrivare in qualche modo ad una configurazione d'arresto.* M semidecide *un linguaggio* $L \subseteq \Gamma_0^*$ *se per ogni* $x \in \Gamma_0^*$ *accade che:* $x \in L$ *se e solo se* M *accetta* x*.*

Notiamo che nel caso probabilistico una macchina accetta un input se esiste almeno una computazione che si arresti: lo stesso input può portare a numerose computazioni che non si fermano mai. Possiamo anche definire in modo più stringente che cosa voglia dire *decidere* un linguaggio e *computare* una funzione.

Definizione 1.6.3. M ***decide*** *un linguaggio* $L \subseteq \Gamma_0^*$ *se per ogni* $x \in \Gamma_0^*$ *valgono le seguenti condizioni:*

- *La configurazione iniziale non porta ad alcuna computazione infinita;*
- $x \in L$ *se e solo se esiste una computazione che termina nello stato* y*.*

M ***computa*** *una funzione* $f : \Gamma_0^* \to \Gamma_0^*$ *se per ogni* $x \in \Gamma_0^*$ *valgono le seguenti condizioni:*

- *La configurazione iniziale non porta ad alcuna computazione infinita;*
- *Ogni possibile computazione con input* x *porta all'output* $f(x)$*.*

Una macchina probabilistica appare dunque molto diversa da una macchina deterministica. Tuttavia si può dimostrare il seguente [8]

Teorema 1.6.1. *Se una macchina di Turing non probabilistica semidecide o decide un linguaggio o computa una funzione, allora esiste una macchina di Turing deterministica che decide o semidecide lo stesso linguaggio o computa la stessa funzione.*

Questo fatto sottolinea ancora una volta l'universalità del modello computazionale della macchina di Turing.
Siamo giunti alla conclusione che una macchina deterministica può simulare una macchina probabilistica, anche se per farlo deve eseguire tutte le computazioni possibili della macchina non detrministica. Come risultato si ha che per simulare una computazione probabilistica in n step si ha bisogno di un numero di step dell'ordine di grandezza di e^n [8]. Dunque in realtà il modello probabilistico non ci permette di risolvere più problemi del modello deterministico, piuttosto ci permette di eseguire computazioni più veloci. Tale modello probabilistico è alla base della formulazione quantistica della Macchina di Turing.

Capitolo 2

La macchina di Turing Quantistica

Nel capitolo precedente abbiamo analizzato il modello computazionale noto come Macchina di Turing, che è in grado grazie alla sua universalità di simulare qualsiasi tipo di computazione possibile. Tale modello è un modello completamente *astratto* di macchina computazionale. In realtà una macchina computazionale è un *sistema fisico* la cui evoluzione dinamica lo porta da uno stato iniziali di input a uno stato finale di output. Implicitamente una macchina di Turing idealizza un dispositivo meccanico che funziona secondo i principi della fisica classica. Nella pratica una macchina di Turing trova la sua realizzazione concreta nei moderni computer digitali [5]. Tali dispositivi, pur basandosi nel loro funzionamento su principi della meccanica quantistica, sono realizzati in modo tale da sopprimere ogni effetto legato al mondo quantistico durante la computazione. Tuttavia la progressiva miniaturizzazione dei componenti fisici costituenti i computer digitali porta inevitabilmente alla presenza di effetti quantistici non trascurabili. Da qui nasce l'esigenza di un modello computazionale che si basi sui principi della meccanica quantistica.
Nel 1985 David Deutsch pubblica un articolo in cui formalizza la macchina di Turing quantistica universale, dando il via al paradigma moderno di computazione quantistica. Di seguito seguiremo principalmente [3].

2.1 Interpretazione fisica della tesi di Church-Turing

Nella sezione 1.4 abbiamo enunciato la congettura di Church-Turing, che pone un limite a ciò che può essere computato:"ogni funzione naturalmente considerabile computabile è computabile da una macchina di Turing universale". Essenzialmente tale tesi sembra non avere alcun significato fisico: è una congettura quasi-matematica che rende equivalenti fra di loro tutte le possibili nozioni intuitive di algoritmo o computazione. Tuttavia Deutsch mostra che è possibile enunciare tale tesi attribuendole un significato fisico ed elevandola a status di *principio fisico* con un significato epistemologico non diverso ad

esempio dal principio di equivalenza. Per enunciare tale principio si deve pensare alle *funzioni naturalmente computabili* come *funzioni computabili da un sistema fisico reale.*

Definizione 2.1.1 (simulazione perfetta)**.** *Diciamo che una macchina M* simula perfettamente *un sistema fisico S se esiste un programma $\pi(S)$ per M che la rende equivalente dal punto di vista computazionale al sistema S.*

Si può ora enunciare il *principio di Church-Turing*:

> ogni sistema fisico realizzabile in modo finito può essere perfettamente simulato da un modello universale di macchina computazionale che opera in modo finito.

Il "sistema fisico realizzabile in modo finito" è qualsiasi sistema che include oggetti fisici sui quali si possano eseguire procedure di misura. In questo senso una qualsiasi procedura computazionale viene vista come l'esecuzione di un particolare esperimento fisico e il risultato della computazione può essere visto come il risultato di una misura effettuata su tale sistema fisico [5].
Ora chiariamo il concetto di macchina che "opera in modo finito". Consideriamo una macchina che esegue la computazione in una sequenza di step la cui durata ha un limite inferiore non nullo. Tale macchina opera in modo finito se:

1. solo una porzione finita della macchina è in moto durante ciascuno step;
2. tale moto dipende solo dallo stato del sottosistema finito;
3. le regole che specificano l'evoluzione della macchina sono formalizzabili matematicamente in modo finito.

Le macchine di Turing descritte nel precedente capitolo rispettano tali condizioni, per cui esse sono un esempio di macchina che opera in modo finito. Sottolineiamo la natura empirica di tale principio: per certi versi è simile nella formulazione al terzo principio della termodinamica, il quale afferma che " *nessun processo finito può ridurre la tempertura o l'entropia di un sistema fisico finito a zero*". Tali principi hanno in comune il fatto di non poter essere direttamente falsificabili. Infatti sappiamo bene che una misura fisica ha un'accuratezza finita, per cui nessuna misura fisica può distinguere lo zero assoluto da una temperatura piccola a piacere. Similmente nel caso del nostro principio, dal momento che per una macchina computazionale esistono infiniti programmi possibili, nessun esperimento può verificare che nessuno di tali programmi possa simulare perfettamente un dato sistema fisico.
Il principio di Church-Turing enunciato da Deutsch è più forte della congettura iniziale: il modello classico della macchina di Turing non soddisfa tale principio. Infatti nella dinamica classica i possibili stati di un sistema formano un insieme continuo, cioè non numerabile, mentre abbiamo visto che abbiamo un numero di modi infinito numerabile per preparare un input finito per una macchina di Turing. Perciò una macchina di

Turing non è in grado di simulare perfettamente un sistema fisico classico: nel contesto della fisica classica la macchina di Turing non soddisfa il principio di Church-Turing. In questo senso è evidente la necessità di riformulare il modello della macchina di Turing nell'ambito della teoria quantistica.

2.2 Qubit

L'ingrediente fondamentale per la costruzione di un modello computazionale quantistico è il qubit [10], l'equivalente quantistico del bit classico. Un qubit è un qualsiasi sistema quantistico che può avere accesso a solo due stati, che indicheremo con $|0\rangle$ e $|1\rangle$ [5]. Lo stato generico $|\psi\rangle$ di un qubit è dunque esprimibile come:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

con α e β coefficienti complessi. Per chi lo spazio di Hilbert associato ad un qubit è $\mathbb{C}^2$. Possiamo identificare gli stati $|0\rangle$ e $|1\rangle$ con i vettori della base canonica di $\mathbb{C}^2$:

$$|0\rangle \equiv \begin{pmatrix}1\\0\end{pmatrix} \quad |1\rangle \equiv \begin{pmatrix}0\\1\end{pmatrix} \tag{2.2}$$

per cui lo stato generico di un qubit è un vettore bidimensionale a coefficienti complessi nella forma

$$|\psi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix} \tag{2.3}$$

Imponiamo per $|\psi\rangle$ la condizione di ket unitario, per cui avremo che $|\alpha|^2+|\beta|^2 = 1$. Un qubit rimane in una sovrapposizione di stati finché non viene eseguita alcuna procedura di osservazione su di esso: una volta eseguita una misura su $|\psi\rangle$ avremo la probabilità $|\alpha|^2$ di trovare il qubit nello stato $|0\rangle$ e una probabilità $|\beta|^2$ di trovarlo nello stato $|1\rangle$.
La condizione di normalizzazione $|\alpha|^2+|\beta|^2 = 1$ permette di riscrivere (2.1) come

$$|\psi\rangle = e^{i\gamma}(\cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle) \tag{2.4}$$

con γ, θ e ϕ parametri reali [10]. Possiamo ignorare il fattore di fase $e^{i\gamma}$ in quanto non produce alcun effetto misurabile [10] e possiamo esprimere lo stato di un qubit come:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle \tag{2.5}$$

che permette un'interpretazione geometrica dello stato di un qubit. Infatti θ e ϕ individuano un punto sulla sfera a raggio unitario in 3 dimensioni. Tale sfera prende il nome di *sfera di Bloch*, rappresentata in fig. 2.1. Le trasformazioni applicabili a un singolo qubit sono infinite e corrispondono a rotazioni sulla sfera di Bloch [5].
In natura esistono diversi sistemi che realizzano concretamente un qubit:

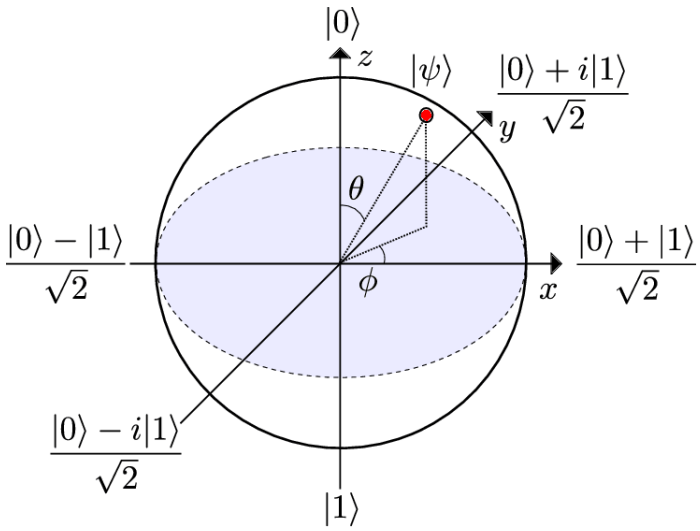

Figura 2.1: La sfera di Bloch è un utile strumento che permette di visualizzare lo stato di un qubit e le operazioni eseguibili su di esso. Ciascun punto della sfera rappresenta una combinazione lineare degli stati $|0\rangle$ e $|1\rangle$ con coefficienti complessi.

- un elettrone, identificando i due stati accessibili con gli stati di spin up o spin down;
- un fotone, identificando i due stati accessibili con i due possibili stati di polarizzazione;
- un atomo, che può realizzare concretamente un qubit se i primi due stati energetici sono vicini fra loro e ben separati dai livelli energetici più alti.

2.3 Calcolatore quantistico

Formalizziamo in questa sezione un modello computazionale quantistico che indicheremo con Q. Come per la macchina di Turing classica, il nostro modello prevede la presenza di due elementi fondamentali: un processore finito che permette la gestione degli stati interni della macchina e un nastro infinito che rappresenta la memoria di cui verrà utilizzata solamente una porzione finita. In questo caso si considera però un nastro infinito lungo entrambe le direzioni. La computazione procede in step dalla durata fissa T e durante ciascuno step interagiscono fra loro solamente il processore e una porzione finita di memoria.
Il processore è un set finito di M qubit:

$$\{|n_i\rangle\} \quad (i = 0, 1, \ldots, M-1)$$

mentre la memoria consiste in una sequenza infinita di qubit:

$$\{|m_i\rangle\} \quad (i \in \mathbb{Z})$$

Indichiamo lo stato del processore e della memoria con i ket $|\boldsymbol{n}\rangle$ e $|\boldsymbol{m}\rangle$ definiti come

$$|\boldsymbol{n}\rangle \equiv |n_0, n_1, \ldots, n_{M-1}\rangle = |n_0\rangle\,|n_1\rangle \ldots |n_{M-1}\rangle \tag{2.6}$$

$$|\boldsymbol{m}\rangle \equiv |\ldots, m_{-1}, m_0, m_1, \ldots\rangle = \ldots |m_{-1}\rangle\,|m_0\rangle\,|m_1\rangle \ldots \tag{2.7}$$

Definiamo l'osservabile $\hat{x}$ che rappresenta quello che nel modello classico era la posizione della casella corrente. Tale osservabile ha come spettro l'intero insieme $\mathbb{Z}$: indichiamo con x l'autovalore posizione e con $|x\rangle$ il corrispondente autoket.
Possiamo individuare a questo punto lo stato di Q mediante un ket unitario in uno spazio di Hilbert H, la cui base è formata dagli ket nella forma

$$|x; \boldsymbol{n}; \boldsymbol{m}\rangle \equiv |x; n_0, n_1, \ldots, n_{M-1}; \ldots, m_{-1}, m_0, m_1, \ldots\rangle = |x\rangle\,|\boldsymbol{n}\rangle\,|\boldsymbol{m}\rangle \tag{2.8}$$

Chiamiamo gli stati (2.8) *stati della base computazionale.*
In meccanica quantistica sappiamo che (in rappresentazione di Schroedinger) l'evoluzione temporale di un sistema è definita da un operatore unitario $\mathbf{U}$ che agisce sul ket di stato [10]. Sia $|\psi(t)\rangle$ lo stato del sistema al tempo t, allora lo stato del sistema al tempo t' è dato da:

$$|\psi(t')\rangle = \mathbf{U}(t' - t)\,|\psi(t)\rangle \tag{2.9}$$

con la condizione di unitarietà

$$\mathbf{U}\mathbf{U}^\dagger = \mathbf{U}^\dagger\mathbf{U} = \hat{1} \tag{2.10}$$

con $\hat{1}$ operatore identità.
Cerchiamo una relazione che esprima l'evoluzione temporale dello stato della nostra macchina. Sappiamo che ogni step computazionale ha una durata finita T: possiamo dunque dire che due stati consecutivi di Q sono legati da un operatore unitario $\mathbf{U}$ costante nel tempo:

$$|\psi(sT)\rangle = \mathbf{U}\,|\psi((s-1)T)\rangle = \mathbf{U}^2\,|\psi((s-2)T)\rangle = \cdots = \mathbf{U}^s\,|\psi(0)\rangle \tag{2.11}$$

dove si pone $t = 0$ l'istante in cui ha inizio la computazione e s indica l's-esimo step computazionale. In generale lo stato della macchina Q è una combinazione lineare di stati della base computazionale.
All'istante iniziale la macchina viene preparata in modo da avere x e $\boldsymbol{n}$ nulli mentre un numero finito di qubit della memoria è preparato in modo tale da rappresentare un input:

$$|\psi(0)\rangle = \sum_m \lambda_m\,|0; \mathbf{0}; \boldsymbol{m}\rangle \tag{2.12}$$

con la condizione di normalizzazione

$$\sum_m |\lambda_m|^2 = 1 \tag{2.13}$$

con un numero finito di λ_m non nulli. Tale costruzione dello stato iniziale ricorda molto la configurazione iniziale di una macchina di Turing classica: $x = 0$ indica che la casella che si sta leggendo è la casella iniziale; $\boldsymbol{n} = \mathbf{0}$ individua lo stato iniziale della macchina; la stringa ottenuta con gli autovalori degli osservabili preparati di $\hat{\boldsymbol{m}}$ è l'equivalente della stringa in input per la macchina classica.
Per garantire che Q soddisfi le tre caratterizzazioni di "macchina operante in modo finito" l'elemento di matrice di $\mathbf{U}$ deve avere la seguente forma:

$$\langle x'; \boldsymbol{n'}; \boldsymbol{m'} | \mathbf{U} | x; \boldsymbol{n}; \boldsymbol{m} \rangle = [\delta_{x'}^{x+1} \mathrm{U}^+(\boldsymbol{n}, m_x) + \delta_{x'}^{x-1} \mathrm{U}^-(\boldsymbol{n}, m_x)] \prod_{y \neq x} \delta_{m'_y}^{m_y} \tag{2.14}$$

Vediamo che cosa comporta tale richiesta. La produttoria a destra assicura che solo il qubit x-esimo della memoria viene coinvolto nello step considerato. I termini $\delta_{x'}^{x\pm1}$ assicurano che la posizione x può solo aumentare o diminuire di un'unità, ovvero ci garantisce che ad ogni step computazionale la macchina si sposta di una sola casella avanti o indietro. $\mathrm{U}^\pm$ sono due funzioni dipendenti esclusivamente dallo stato del processore $|\boldsymbol{n}\rangle$ e dal valore dell' x-esimo qubit della memoria $|m_x\rangle$ che rappresentano l'evoluzione dinamica e possono essere scelte in modo arbitrario con l'unico vincolo di garantire l'unitarietà di $\mathbf{U}$. Ciascuna scelta di $\mathrm{U}^\pm$ definisce univocamente un computer quantistico che indicheremo con $Q[\mathrm{U}^+, \mathrm{U}^-]$, così come la scelta della funzione di transizione classica determina in modo univoco una macchina di Turing classica.
Notiamo che dalla (2.11) due stati consecutivi sono uguali se e solo se la computazione è banalmente l'identità, ovvero non si esegue più alcuna operazione sullo stato della macchina. Perciò diremo che Q arresta la computazione quando si hanno due stati consecutivi identici. Inoltre diremo che un programma è *valido* se porta Q ad arrestarsi in un tempo finito, ovvero in un numero finito di step computazionali.
Ma come si fa a sapere quando una macchina quantistica si arresta? Sappiamo che qualunque osservazione (misura) dello stato di Q altera lo stesso stato della macchina e si interferirebbe con la computazione. Per cui Q deve poter dire autonomamente di aver finito la computazione, ma come può farlo?
Bisogna predisporre uno dei qubit interni della macchina, ad esempio $|n_0\rangle$, per tale scopo. Ogni programma che sia valido non deve poter interagire in alcun modo con $|n_0\rangle$, inizializzato a $|n_0\rangle = |0\rangle$ all'inizio della computazione, finchè la macchina non si arresti: in tal caso il programma pone $|n_0\rangle = |1\rangle$. Inoltre tale osservabile deve poter essere *misurato* periodicamente da un utente esterno senza che tale misura alteri lo stato della macchina e interferisca con la computazione.
Dal momento che l'evoluzione temporale di Q è governata da un operatore unitario, la sua

dinamica deve essere reversibile. In generale una macchina di Turing classica non è reversibile, in quanto la funzione di transizione non è sempre invertibile. C.H. Bennet dimostra in [1] che è possibile costruire una classe di macchine reversibili equivalenti dal punto di vista computazionale alla macchina di Turing universale. La macchina $Q[\mathrm{U}^+, \mathrm{U}^-]$ diventa equivalente a una qualsiasi macchina di Turing reversibile se scegliamo:

$$U^{\pm}(\boldsymbol{n'}, m'_x; \boldsymbol{n}, m_x) = \delta_{\boldsymbol{n'}}^{\boldsymbol{A}(\boldsymbol{n},m_x)} \delta_{m'_x}^{B(\boldsymbol{n},m_x)} (1 \pm C(\boldsymbol{n}, m_x)) \tag{2.15}$$

con $\boldsymbol{A}(\boldsymbol{n}, m_x)$, $B(\boldsymbol{n}, m_x)$ e $C(\boldsymbol{n}, m_x)$ funzioni arbitrarie che hanno valori rispettivamente in $\mathbb{Z}^M$, $\mathbb{Z}$ e $\{-1, 1\}$. Per garantire l'unitarietà è necessario e sufficiente che la mappa

$$\{(\boldsymbol{n}, m_x)\} \mapsto \{(\boldsymbol{A}(\boldsymbol{n}, m_x), B(\boldsymbol{n}, m_x), C(\boldsymbol{n}, m_x))\}$$

sia biiettiva.
Tale scelta garantisce che la dinamica di Q rimanga all'interno della base computazionale scelta: abbiamo in questo modo una macchina di Turing quantistica. In questo modo gli unici elementi di matrice dell'operatore evoluzione $\mathbf{U}$ non nulli sono della forma $\langle x \pm 1; \mathbf{n}'; m'_x; m_{y\neq x}|\mathbf{U}|x, \mathbf{n}, m_x, m_{y\neq x}\rangle$[11]. Con una scelta opportuna delle funzioni $\boldsymbol{A}(\boldsymbol{n}, m_x)$, $B(\boldsymbol{n}, m_x)$ e $C(\boldsymbol{n}, m_x)$ possiamo rendere Q equivalente ad una macchina di Turing universale, ottenendo così un calcolatore quantistico universale.

2.4 Macchina quantistica Universale

Similmente alla macchina universale classica, la macchina di Turing quantistica universale Q ammette come input un programma eseguibile e i dati sui quali si vuole fare agire il programma. Il programma che si vuole simulare viene codificato da un set di qubit della memoria che indicheremo con $|\pi\rangle$, mentre l'input per il programma viene codificato dal set $|a\rangle$. I restanti qubit della memoria vengono inizializzati a zero durante la preparazione dello stato iniziale.
Diciamo che Q computa la funzione $f(x)$ se esiste un programma codificato da $|\pi(f)\rangle$ tale che:

$$\mathbf{U}^s |0; \mathbf{0}; \pi(f), a, \mathbf{0}\rangle = |0; 1, \mathbf{0}; \pi(f), a, f(a), \mathbf{0}\rangle \tag{2.16}$$

con un numero s di step computazionali finito. Esplicitare la forma delle funzioni $\mathtt{U}^{\pm}$ che caratterizzano la macchina di Turing quantistica universale è piuttosto complicato [3], per cui descriveremo i processi computazionali studiando direttamente gli stati che la macchina assume durante la computazione.
Possiamo generalizzare la nozione di programma per qualsiasi funzione computabile. Sia f una funzione computabile e siano i e j due interi. Allora esiste un programma $\pi(f, i, j)$ che calcola la funzione f sul numero codificato dai qubit nello slot i e scrive il risultato

nello slot j:

$$\left|\pi(f,i,j);\overbrace{a}^{slot\ i};b\right\rangle \mapsto |\pi(f,i,j);a;\overbrace{b\oplus f(a)}^{slot\ j}\rangle \tag{2.17}$$

dove l'input a è codificato dai qubit nello slot i e assumiamo che i qubit b nello slot j siano inizializzati a zero durante la preparazione della macchina. L'operazione $\oplus$ è l'operazione associativa e commutativa tale che:

$$x \oplus 0 = x \tag{2.18}$$
$$x \oplus x = 0 \tag{2.19}$$

cioè altro non è che l'addizione modulo 2. Tale operazione è introdotta per tenere traccia dei qubit che sovrascriviamo durante la computazione per mantenere il nostro processo reversibile, garantendo così l'evoluzione unitaria del nostro sistema [3][10].
Si possono concatenare fra loro diversi programmi, ad esempio $\pi_1 \cdot \pi_2$ indica un programma il cui effetto è quello di π_1 seguito da π_2. Mostriamo che attraverso la concatenazione di programmi possiamo definire per ogni funzione biiettiva g un programma $\Phi(g,i)$ che calcoli il valore di g sull'input espresso nello slot i e scriva l'output nello stesso slot. E' sufficiente definire il programma nel modo seguente:

$$\Phi(g,i) = \pi(g,i,j)\cdot\pi(g^{-1},j,i)\cdot\pi(I,j,i)\cdot\pi(I,i,j) \tag{2.20}$$

dove I è detta *funzione di misura perfetta* definita come

$$|\pi(I,i,j);a;b\rangle \mapsto |\pi(I,i,j);a;a\oplus b\rangle \tag{2.21}$$

Infatti inizializzando lo slot i in modo da codificare un input a e inizializzando lo slot j a $|0\rangle$ avremo che (2.20) agisce nel seguente modo:

$$\begin{aligned}
|\pi(g,i,j),a,0\rangle &\mapsto |\pi(g,i,j),a,0\oplus g(a)\rangle = |\pi(g,i,j),a,g(a)\rangle \\
\left|\pi(g^{-1},j,i),a,g(a)\right\rangle &\mapsto \left|\pi(g^{-1},j,i),a\oplus a,g(a)\right\rangle = \left|\pi(g^{-1},j,i),0,g(a)\right\rangle \\
|\pi(I,j,i),0,g(a)\rangle &\mapsto |\pi(I,j,i),0\oplus g(a),g(a)\rangle = |\pi(I,j,i),g(a),g(a)\rangle \\
|\pi(I,i,j),g(a),g(a)\rangle &\mapsto |\pi(I,i,j),g(a),g(a)\oplus g(a)\rangle = |\pi(I,i,j),g(a),0\rangle
\end{aligned} \tag{2.22}$$

dimostrando così che

$$|\Phi(g,i),a,0\rangle \mapsto |\Phi(g,i),g(a),0\rangle \tag{2.23}$$

Programmi così definiti hanno la caratteristica di far evolvere la macchina in stati della stessa base computazionale di partenza.

Si può generalizzare la nozione di programma anche nel caso di operatori. Sia $\hat{A}$ trasformazione unitaria che agisce su un singolo qubit. Dal momento che lo spazio di Hilbert associato al nostro qubit è bidimensionale complesso, lo spazio degli operatori unitari a singolo qubit è uno spazio complesso quadrimensionale. Rappresentando le trasformazioni unitarie tramite matrici, abbiamo che:

$$\hat{A} = \begin{pmatrix} A^{00} & A^{01} \\ A^{10} & A^{11} \end{pmatrix} \tag{2.24}$$

Applicando $\hat{A}$ ad un qubit nello stato $|a\rangle = \alpha|0\rangle + \beta|1\rangle$ si ottiene:

$$\begin{aligned} \hat{A}|\psi\rangle = \begin{pmatrix} A^{00} & A^{01} \\ A^{10} & A^{11} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha A^{00} + \beta A^{01} \\ \alpha A^{10} + \beta A^{11} \end{pmatrix} = \\ = (\alpha A^{00} + \beta A^{01})|0\rangle + (\alpha A^{10} + \beta A^{11})|1\rangle \end{aligned} \tag{2.25}$$

Se applichiamo la trasformazione $\hat{A}$ agli elementi $|0\rangle$ e $|1\rangle$ si ottiene:

$$\begin{aligned} \hat{A}|0\rangle &= A^{00}|0\rangle + A^{10}|1\rangle \\ \hat{A}|1\rangle &= A^{01}|0\rangle + A^{11}|1\rangle \end{aligned} \tag{2.26}$$

Per fare eseguire alla nostra macchina universale una trasformazione unitaria su un qubit dobbiamo definire un programma $\Phi(\hat{A}, i)$ tale per cui la trasformazione $\hat{A}$ viene eseguita sul least significant qubit dello slot i-esimo che indichiamo con $|a\rangle$:

$$\left|\Phi(\hat{A}, i); a\right\rangle \mapsto \left|\Phi(\hat{A}, i)\right\rangle \hat{A}|a\rangle \tag{2.27}$$

Particolare importanza hanno le *matrici di Pauli* $\sigma_x, \sigma_y, \sigma_z$ che insieme all'operatore identità formano una base per lo spazio degli operatori che agiscono sullo spazio di Hilbert di un singolo qubit:

$$\begin{aligned} \hat{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \\ \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \end{aligned}$$

Un altro operatore molto importante per le applicazioni in computazione quantistica è l'*operatore di Hadamard* [10], definito nel seguente modo:

$$H = \frac{\sigma_x + \sigma_z}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.28}$$

Le matrici di Pauli danno origine alle tre classi di *operatori di rotazione* definiti dalle eq:

$$R_x(\theta) \equiv e^{-i\theta\sigma_x/2} = \cos\tfrac{\theta}{2} I - i \sin\tfrac{\theta}{2}\sigma_x = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{2.29}$$

$$R_y(\theta) \equiv e^{-i\theta\sigma_y/2} = \cos\tfrac{\theta}{2} I - i \sin\tfrac{\theta}{2}\sigma_y = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{2.30}$$

$$R_z(\theta) \equiv e^{-i\theta\sigma_z/2} = \cos\tfrac{\theta}{2} I - i \sin\tfrac{\theta}{2}\sigma_z = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \tag{2.31}$$

dove $R_x(\theta), R_y(\theta), R_z(\theta)$ rappresentano una rotazione sulla sfera di Bloch di un angolo θ rispettivamente attorno all'asse x, y, z, come mostrato in fig().
Attraverso gli operatori di rotazione possiamo rappresentare qualsiasi trasformazione unitaria a singolo qubit a meno di una fase totale, infatti vale il seguente [10]:

Teorema 2.4.1. *Sia U operatore unitario su un singolo qubit. Allora esistono quattro parametri reali $\alpha, \beta, \gamma, \delta$ tali che:*

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta) \tag{2.32}$$

Tale risultato è importante per l'universalità del nostro modello. Infatti possiamo definire programmi nella forma (2.27) che rappresentino gli operatori di rotazione e attraverso la loro concatenazione siamo in grado di far eseguire qualsiasi trasformazione che sia nella forma (2.32): siamo in grado di simulare sulla nostra macchina universale qualsiasi trasformazione unitaria a singolo qubit. Questo è vero anche per un numero finito qualsiasi L di bit? Mostriamo che esiste un programma che effettua una trasformazione unitaria su L qubit arbitrariamente vicina a una trasformazione desiderata. Seguiremo il formalismo adottato in [11].
Indichiamo lo stato della nostra macchina di Turing quantistica come $|Q_{x,\mathbf{n}}\rangle\,|D\rangle\,|P\rangle\,|\Sigma\rangle$ dove $|Q_{x,\mathbf{n}}\rangle$ indica lo stato del processore includendo l'informazione del numero di casella in lettura x, $|D\rangle$ è lo stato che rappresenta i qubit manipolati durante la computazione, $|P\rangle$ è lo stato dei qubit predisposti alla rappresentazione del programma mentre $|\Sigma\rangle$ è lo stato del nastro che non viene coinvolto nella computazione. Vogliamo dimostrare che, data una qualsiasi trasformazione unitaria $\mathcal{U}$ che agisce su un numero finito di qubit, esiste un programma $|P(\mathcal{U},\epsilon)\rangle$ e un numero intero $s(\mathcal{U},\epsilon)$ tale che:

$$\mathbf{U}^{s(\mathcal{U},\epsilon)}\,|Q_{x,\mathbf{n}}\rangle\,|D\rangle\,|P(\mathcal{U},\epsilon)\rangle\,|\Sigma\rangle = \left|Q'_{x,\mathbf{n}}\right\rangle\,|D'\rangle\,|P'(\mathcal{U},\epsilon)\rangle\,|\Sigma\rangle \tag{2.33}$$

con $|D'\rangle$ arbitrariamente vicino a $\mathcal{U}\,|D\rangle$, ovvero deve valere la seguente disuguaglianza:

$$||\,|D'\rangle - \mathcal{U}\,|D\rangle\,||^2 < \epsilon \tag{2.34}$$

Per dimostrarlo useremo un ragionamento induttivo che dimostri l'esistenza di un programma che faccia evolvere lo stato di un set di L qubit $|D\rangle = |\psi_{1-L}\rangle$ in uno stato $|D'\rangle$ arbitrariamente vicino a $|0_{1-L}\rangle$, dove con la notazione $|\psi_{1-L}\rangle$ indichiamo $|\psi_0\rangle\,|\psi_1\rangle \ldots |\psi_L\rangle$.

Possiamo scrivere

$$|\psi_{1-L}\rangle = c_0 |0_1\rangle \left|\psi_{2-L}^{(0)}\right\rangle + c_1 |1_1\rangle \left|\psi_{2-L}^{(1)}\right\rangle \tag{2.35}$$

Facciamo la seguente ipotesi induttiva: esistono due programmi ρ_0 e ρ_1 che evolvono rispettivamente $\left|\psi_{2-L}^{(0)}\right\rangle$ e $\left|\psi_{2-L}^{(1)}\right\rangle$ nello stato $|0_{2-L}\rangle$. Definiamo un programma ρ nel seguente modo: se il qubit n. 1 è 0 viene eseguito ρ_0, altrimenti viene eseguito ρ_1. Applicando tale programma a (2.35) si ottiene:

$$(c_0 |0_1\rangle + c_1 |1_1\rangle) |0_{2-L}\rangle \tag{2.36}$$

che si può far evolvere nello stato $|0_{1-L}\rangle$ attraverso una trasformazione unitaria a singolo qubit.
Utilizzando un procedimento più formale, riformuliamo la nostra dimostrazione. La nostra ipotesi è la seguente: esiste un programma rappresentato dal set di qubit $|P_0\rangle$ che agisce nel modo seguente:

$$\mathbf{U}^{s_0} |Q_{x,\mathbf{n}}\rangle |\Phi_0\rangle \left|\psi_{2-L}^{(0)}\right\rangle |P_0\rangle |\Sigma\rangle = \left|Q'_{x,\mathbf{n}}\right\rangle \left|D^{0\prime}\right\rangle |P'_0\rangle |\Sigma\rangle \tag{2.37}$$

ed un programma rappresentato dal set di qubit $|P_1\rangle$ tale che:

$$\mathbf{U}^{s_1} |Q_{x,\mathbf{n}}\rangle |\Phi_1\rangle \left|\psi_{2-L}^{(1)}\right\rangle |P_1\rangle |\Sigma\rangle = \left|Q'_{x,\mathbf{n}}\right\rangle \left|D^{1\prime}\right\rangle |P'_1\rangle |\Sigma\rangle \tag{2.38}$$

dove $|D^{0\prime}\rangle$ e $|D^{1\prime}\rangle$ sono arbitrariamente vicini a $|0_{2-L}\rangle$, $|\Phi_1\rangle$ indica lo stato del qubit n.1 e s_0 ed s_1 sono i rispettivi tempi di esecuzione dei due programmi.
L'esistenza del prgramma ρ descritto in precedenza implica l'esistenza di un set di qubit $|P\rangle$ tale che:

$$\mathbf{U}^{s_0} |Q_{x,\mathbf{n}}\rangle |0_1\rangle \left|\psi_{2-L}^{(0)}\right\rangle |P\rangle |\Sigma\rangle = \left|Q'_{x,\mathbf{n}}\right\rangle \left|D^{0\prime}\right\rangle |P'_0\rangle |\Sigma\rangle \tag{2.39}$$

$$\mathbf{U}^{s_1} |Q_{x,\mathbf{n}}\rangle |1_1\rangle \left|\psi_{2-L}^{(1)}\right\rangle |P\rangle |\Sigma\rangle = \left|Q'_{x,\mathbf{n}}\right\rangle \left|D^{1\prime}\right\rangle |P'_1\rangle |\Sigma\rangle \tag{2.40}$$

Notiamo che se $s_0 = s_1 \equiv s$ possiamo scrivere:

$$\mathbf{U}^{s} |Q_{x,\mathbf{n}}\rangle (c_0 |0_1\rangle \left|\psi_{2-L}^{(0)}\right\rangle + c_1 |1_1\rangle \left|\psi_{2-L}^{(1)}\right\rangle) |P\rangle |\Sigma\rangle = \left|Q'_{x,\mathbf{n}}\right\rangle (c_0 |0_1\rangle + c_1 |1_1\rangle) |D'\rangle |P'\rangle |\Sigma\rangle \tag{2.41}$$

dove $|D'\rangle$ è arbitrariamente vicino a $|0_{2-L}\rangle$. In questo caso è sufficiente eseguire una trasformazione a singolo bit per far evolvere anche l'ultimo qubit rimasto nello stato $|0\rangle$. Perciò siamo ora in grado di far evolvere arbitrariamente un set di L qubit $|D\rangle$ in $|D'\rangle$ arbitrariamente vicino a $\mathcal{U} |D\rangle$ con $\mathcal{U}$ trasformazione unitaria arbitraria. Infatti partendo da $|D\rangle$, sappiamo in base alla dimostrazione precedente che possiamo far evolvere

tale stato in $|0_{1-L}\rangle$ con un programma il cui stato denotiamo con $|P\rangle$. Sappiamo anche che esiste un programma in grado di far evolvere lo stato $|D'\rangle$ in $|0_{1-L}\rangle$, che indichiamo con $|P'\rangle$ Dal momento che l'evoluzione unitaria comporta una dinamica reversibile della macchina quantistica, siamo in grado di definire $|P'^{-1}\rangle$ in grado di far evolvere lo stato $|0_{1-L}\rangle$ nello stato $|D'\rangle$. A questo punto tramite la concatenazione dei programmi rappresentati da $|P\rangle$ e $|P'^{-1}\rangle$ siamo in grado di far evolvere lo stato $|D\rangle$ nello stato $|D'\rangle$ simulando l'azione di $\mathcal{U}$ sul ket iniziale $|D\rangle$.
Abbiamo mostrato che il nostro calcolatore quantistico universale può simulare con precisione arbitraria qualsiasi macchina di Turing quantistica $Q[\mathrm{U}^+, \mathrm{U}^-]$.

Capitolo 3

Proprietà del computer quantistico universale

3.1 Numeri casuali

La generazione di numeri casuali trova molte applicazioni nella fisica moderna e non solo: i numeri casuali sono essenziali nella crittografia, nelle simulazioni Monte Carlo, in alcuni processi risolutivi di problemi numerici, nelle ricerche statistiche [13]. Storicamente la generazione di numeri casuali viene implementata attraverso i generatori di numeri pseudocasuali, basati su algoritmi matematici in grado di produrre in output sequenze di numeri periodiche con un periodo molto grande. In linea di principio dunque tali numeri non sono veramente casuali, ma sono generati in modo deterministico tramite formule matematiche [13].
Mostriamo che un calcolatore quantistico è in grado di generare *veri* numeri casuali, imprevedibili conoscendo solamente l'input [3]. Si riesce a generare numeri random sfruttando uno dei postulati della meccanica quantistica, ovvero la riduzione dello stato di un sistema quanto-meccanico in seguito ad un processo di misura [10]. Prendiamo in considerazione il seguente programma per una macchina di Turing quantistica:

$$\Phi(H,i)\cdot\pi(I,i,j)$$

con $\Phi(H,i)$ e $\pi(I,i,j)$ definiti come in (2.27) e in (2.21), dove H è la matrice di Hadamard. Prepariamo la macchina nel seguente stato iniziale (per maggiore chiarezza specifichiamo solo gli stati dei qubit predisposti per rappresentare l'input e il programma):

$$\left|\Phi(H,i)\cdot\pi(I,i,j);\overbrace{0}^{i},\overbrace{0}^{j}\right\rangle$$

Vediamo come agisce su tale stato la prima parte del programma $\Phi(H, i)$, come specificato in (2.27):

$$|\Phi(H,i),0,0\rangle \mapsto |\Phi(H,i)\rangle\, H\,|0\rangle\,|0\rangle = \frac{1}{\sqrt{2}}\,|\Phi(H,i)\rangle\,(|0\rangle + |1\rangle)\,|0\rangle = \tag{3.1}$$

$$= \frac{1}{\sqrt{2}}\,|\Phi(H,i),0,0\rangle + \frac{1}{\sqrt{2}}\,|\Phi(H,i),1,0\rangle \tag{3.2}$$

Ora applichiamo la seconda parte del programma $\pi(I, i, j)$ e otteniamo, come specificato in (2.21):

$$|\Phi(H,i)\cdot\pi(I,i,j)\rangle\,(\frac{1}{\sqrt{2}}\,|0,0\oplus 0\rangle + \frac{1}{\sqrt{2}}\,|1,0\oplus 1\rangle) = \tag{3.3}$$

$$= |\Phi(H,i)\cdot\pi(I,i,j)\rangle\,(\frac{1}{\sqrt{2}}\,|0,0\rangle + \frac{1}{\sqrt{2}}\,|1,1\rangle) \tag{3.4}$$

A questo punto una qualsiasi misura del qubit nello slot j comporterà la riduzione ad uno solo dei due stati comportando una misura equiprobabile di 0 o 1.
Si possono generare numeri casuali secondo una distribuzione di probabilità desiderata. Ad esempio il programma

$$|\pi(I,i,j)\rangle\,(\cos\theta\,|0,0\rangle + \sin\theta\,|1,0\rangle) \tag{3.5}$$

porta alla sovrapposizione di stati

$$\cos\theta\,|\pi(I,i,j),0,0\oplus 0\rangle + \sin\theta\,|\pi(I,i,j),1,0\oplus 1\rangle = \tag{3.6}$$

$$= \cos\theta\,|\pi(I,i,j),0,0\rangle + \sin\theta\,|\pi(I,i,j),1,1\rangle \tag{3.7}$$

per cui una misura del bit nello slot j ha come risultato 0 con probabilità $\cos^2\theta$ oppure 1 con probabilità $\sin^2\theta$. Tutti i possibili stati nella forma (3.5) rappresentano programmi validi per la macchina di Turing quantistica [3]. Per cui possiamo in linea di principio creare programmi validi con distribuzione di probabilità irrazionali arbitrarie. Di conseguenza ogni sistema stocastico discreto e finito può essere simulato perfettamente da Q [3].

3.2 Simulazione di sistemi fisici arbitrari finiti

Abbiamo visto che l'evoluzione dinamica di un calcolatore quantistico è governata da un operatore unitario nello spazio di Hilbert $\mathcal{H}$. Questo è vero idealmente, in quanto l'evoluzione unitaria è prerogativa dei *sistemi chiusi*: i principi della Termodinamica impediscono la costruzione di tali sistemi. Perciò non si possono costruire sistemi che non interagiscano con l'ambiente esterno e nel descrivere l'evoluzione dinamica bisogna

tener conto dell'accoppiamento fra sistema e ambiente [3]. Il formalismo di Hilbert dei ket di stato non ci permette di affrontare tale problema, poichè l'ambiente presenta un elevato numero di gradi di libertà la cui descrizione risulta troppo complicata [2]. Bisogna dunque ricorrere al formalismo delle *matrici densità*.[1]
Consideriamo un sistema finito A e denotiamo con $\mathcal{H}_\mathcal{A}$ lo spazio di Hilbert associato di dimensione L. Poniamo A a contatto con l'ambiente esterno, e sia $\mathcal{H}_\mathcal{B}$ lo spazio di Hilbert dell'ambiente che dato l'elevato numero di gradi di libertà possiamo assumere come spazio di dimensione infinita. Lo spazio di Hilbert dei due sistemi combinati è dato dal prodotto tensoriale dei due spazi individuali [9]:

$$\mathcal{H} = \mathcal{H}_\mathcal{A} \otimes \mathcal{H}_\mathcal{B} \tag{3.8}$$

Lo stato del sistema A è completamente descritto dalla matrice densità ρ_A, che in notazione indiciale indicheremo con $(\rho_A)_a{}^b$. L'evoluzione dinamica del sistema non è governata da operatori unitari, ma dalla matrice di superscattering [3] $\mathbf{S}$ (si utilizza la convenzione di Einstein degli indici ripetuti) [2]:

$$(\rho_A(t))_a{}^b = \mathbf{S}_a{}^{bc}{}_d(\rho_A(0))_c{}^d \tag{3.9}$$

Tale equazione rappresenta la proiezione sullo spazio $\mathcal{H}_\mathcal{A}$ dell'evoluzione unitaria del sistema composto nello spazio $\mathcal{H}$ [3].
In generale l'operatore di superscattering ha la forma seguente [3]:

$$\mathbf{S}_a{}^{bc}{}_d = \mathbf{U}_{ae'}{}^{cf'}\mathbf{U}^{be'}{}_{dg'}(\rho_B(0))_{f'}{}^{g'} \tag{3.10}$$

dove l'operatore $\mathbf{U}$ è operatore unitario nello spazio $\mathcal{H}$, tale per cui

$$\mathbf{U}_{ab'}{}^{cd'}\mathbf{U}^{ef'}{}_{cd'} = \delta_a{}^e\delta_{b'}{}^{f'} \tag{3.11}$$

Il termine $\rho_B(0)$ rappresenta la matrice densità dell'ambiente all'istante iniziale, assumendo che al tempo $t = 0$ sistema e ambiente siano completamente disaccoppiati. Possiamo riscrivere l'operatore di superscattering in una base ortonormale di $\mathcal{H}_\mathcal{B}$ in cui $\rho_B(0)$ ha un'espressione diagonale:

$$\begin{aligned} \mathbf{S}_a{}^{bc}{}_d &= p_{f'}\mathbf{U}_{ae'}{}^{cf'}\mathbf{U}^{be'}{}_{df'} \\ \textstyle\sum_{f'} p_{f'} &= 1 \end{aligned} \tag{3.12}$$

dove $p_{f'}$ sono gli autovalori di $\rho_B(0)$. Chiamiamo $\mathbb{G}$ l'insieme degli operatori di superscattering. Tale insieme è contenuto in uno sottospazio Λ di $\mathcal{H}_\mathcal{A} \times \mathcal{H}^*_\mathcal{A} \times \mathcal{H}^*_\mathcal{A} \times \mathcal{H}_\mathcal{A}$, i cui elementi soddisfano:

$$\sum_a \mathbf{S}_a{}^{bc}{}_c = \delta_c{}^b \tag{3.13}$$

[1] Per un'introduzione di tale formalismo si veda l'appendice A.

[2] Per una breve trattazione dei sistemi aperti si veda l'appendice B.

[3] L'operazione di abbassamento e innalzamento degli indici indica l'operazione di coniugazione complessa, in particolare $\mathbf{U}^{be'}{}_{dg'} \equiv (\mathbf{U}^\dagger)_{dg'}{}^{be'}$

Siano ora $\rho^{(1)}$ e ρ^2 due matrici densità arbitrarie. Gli elementi di $\mathbb{G}$ soddisfano il vincolo [3]:

$$0 \leq (\rho^{(1)})^a{}_b \mathbf{S}_a^{bc}{}_d (\rho^{(2)})_c{}^d \leq 1 \tag{3.14}$$

L'uguaglianza a destra si ottiene se e solo se l'operatore di superscattering è operatore unitario, cioè se consideriamo un sistema isolato; l'uguaglianza a sinistra è impossibile da ottenere fisicamente [3]. Perciò gli elementi fisicamente realizzabili di $\mathbb{G}$ formano un insieme aperto in Λ. Sappiamo che la macchina di Turing quantistica universale Q può simulare qualsiasi trasformazione unitaria. Per cui computando trasformazioni unitarie come in (3.12) possiamo computare ogni elemento di un certo sottoinsieme denso numerabile di $\mathbb{G}$. Inoltre, assumendo $\mathbf{S}^{(1)}$ e $\mathbf{S}^{(2)}$ computabili, qualsiasi combinazione lineare convessa nella forma

$$p_1 \mathbf{S}^{(1)} + p_2 \mathbf{S}^{(2)} \tag{3.15}$$

è computabile grazie al generatore di numeri casuali definito nella sezione precedente. Utilizzando il fatto che ogni elemento di un insieme aperto contenuto in uno spazio vettoriale di dimensione finita può essere espresso come una combinazione lineare convessa finita di elementi di qualsiasi sottoinsieme denso di tale spazio [3], ne consegue che Q può simulare perfettamente qualsiasi sistema fisico il cui spazio degli stati sia finito dimensionale. Perciò possiamo dire che la teoria quantistica è compatibile con il principio di Church-Turing [3].

3.3 Parallelismo quantistico

Il parallelismo quantistico è una caratteristica fondamentale di molti algoritmi quantistici [10]. Euristicamente parlando, con il rischio di un'eccessiva semplificazione, il parallelismo quantistico permette a una macchina quantistica di valutare il valore di una funzione $f(x)$ per diversi valori di x simultaneamente. Classicamente una computazione parallela può essere ottenuta utilizzando più calcolatori contemporaneamente, mentre il parallelismo quantistico permette l'utilizzo di un solo calcolatore, sfruttando il principio della sovrapposizione degli stati [10].
Consideriamo f funzione computabile e costruiamo il programma

$$\frac{1}{\sqrt{N}} \sum_{x=1}^{N} |\pi(f, i, j), x, 0\rangle \tag{3.16}$$

che per linearità e per (2.17) porta alla sovrapposizione di stati

$$\frac{1}{\sqrt{N}} \sum_{x=1}^{N} |\pi(f, i, j), x, f(x)\rangle \tag{3.17}$$

Tale computazione richiede lo stesso tempo e le stesse risorse [3] dell'esecuzione del programma (2.17), ma contiene il risultato di un numero arbitrario N di computazioni separate. Tuttavia per il processo di riduzione dello stato a seguito della misura dei qubit predisposti alla rappresentazione del risultato, solo uno degli N possibili risultati è accessibile all'utente esterno con una probabilità 1/N. Eseguendo diverse volte il programma (3.16) il tempo medio richiesto per ottenere tutti gli N possibili risultati della computazione, che d'ora in poi indicheremo con $\mathbf{f}$, non può essere inferiore al tempo richiesto per ottenere gli N risultati eseguendo serialmente il programma (2.17). Più in generale dimostriamo che il tempo medio richiesto per ottenere ogni risultato della computazione di qualsiasi funzione G($\mathbf{f}$) attraverso (3.16) non può essere inferiore al tempo di esecuzione della computazione seriale tramite (2.17).
Assumiamo per semplicità [3] che τ, tempo di esecuzione di (2.17), sia indipendente dall'input e che il tempo per combinare tutti i risulati $\mathbf{f}$ per determinare $G(\mathbf{f})$ sia trascurabile. Per cui stimiamo il tempo di esecuzione della computazione in serie come $\tau_{serie} = N\tau$. Assumiamo inoltre l'esistenza di un programma P rappresentato da $|P\rangle$ che per ogni funzione f ha la probabilità $|\beta|^2$ di estrarre il valore $G(\mathbf{f})$.Possiamo stimare il tempo medio per ottenere il risultato voluto dalla nostra computazione parallela come $\tau_{parallelo} = \tau/|\beta|^2$. Tale programma agisce nel seguente modo:

$$\frac{1}{\sqrt{N}} \sum_{x=1}^{N} |P, x, f(x)\rangle \mapsto \beta \, |P; 0, G(\mathbf{f})\rangle + \sqrt{1 - |\beta|^2} \, |P; 1, \lambda(\mathbf{f})\rangle \tag{3.18}$$

dove lo stato $|\lambda\rangle$ non contiene alcuna informazione su $G(\mathbf{f})$. A questo punto una qualsiasi misura del primo slot comporterà la riduzione ad uno dei dei due stati. Si ha la probabilità $|\beta|^2$ di ottenere 0 nel primo slot e di conseguenza nel secondo slot si avrà l'informazione sul valore di $G(\mathbf{f})$.
Sia $g(x)$ un'altra funzione computabile. Usando lo stesso programma possiamo ottenere:

$$\frac{1}{\sqrt{N}} \sum_{x=1}^{N} |P, x, g(x)\rangle \mapsto \beta \, |P; 0, G(\mathbf{g})\rangle + \sqrt{1 - |\beta|^2} \, |P; 1, \lambda(\mathbf{g})\rangle \tag{3.19}$$

Calcoliamo i seguenti prodotti bra-ket:

$$\frac{1}{N} \sum_{x=1}^{N} \langle f(x)|g(x)\rangle = \frac{1}{N} \sum_{x=1}^{N} \delta(f(x), g(x)) \tag{3.20}$$

$$\begin{aligned}&(\beta^* \, \langle P; 0, G(\mathbf{f})| + \sqrt{1 - |\beta|^2} \, |P; 1, \lambda(\mathbf{f})\rangle)(\beta \, |P; 0, G(\mathbf{g})\rangle + \sqrt{1 - |\beta|^2} \, |P; 1, \lambda(\mathbf{g})\rangle) = \\ &= |\beta|^2 \delta(G(\mathbf{f}), G(\mathbf{g})) + (1 - |\beta|^2) \, \langle \lambda(\mathbf{f})|\lambda(\mathbf{g})\rangle\end{aligned} \tag{3.21}$$

Dal momento che (3.21) si ottiene come evoluzione unitaria da (3.20) otteniamo:

$$\frac{1}{N} \sum_{x=1}^{N} \delta(f(x), g(x)) = |\beta|^2 \delta(G(\mathbf{f}), G(\mathbf{g})) + (1 - |\beta|^2) \, \langle \lambda(\mathbf{f})|\lambda(\mathbf{g})\rangle \tag{3.22}$$

Possiamo costruire la funzione $G(\mathbf{f})$ in modo tale che per ogni $f(x)$ esiste una funzione $g(x)$ tale che $G(\mathbf{f}) \neq G(\mathbf{g})$ e $f(x) = g(x)$ per N-1 valori di x. Per cui (3.22) si riduce a

$$1 - \frac{1}{N} = (1 - |\beta|^2) \langle \lambda(\mathbf{f}) | \lambda(\mathbf{g}) \rangle \tag{3.23}$$

da cui segue [3] che $|\beta|^2 < 1/N$. Perciò il tempo medio richiesto per la computazione parallela di $G(\mathbf{f})$ è almeno $\tau_{parallelo} = N\tau = \tau_{serie}$. Notiamo dunque che il parallelismo quantistico non è sufficiente per rendere la computazione quantistica più efficiente di quella classica: dobbiamo combinarlo con altre proprietà emergenti dalla teoria quantistica in modo tale da riuscire ad estrarre più informazione da una singola sovrapposizione di stati [10].

3.3.1 Algoritmo di Deutsch

Analizzeremo ora quello che viene indicato con il nome di *algoritmo di Deutsch* , che offre un esempio della potenzialità del parallelismo quantistico combinato con la proprietà della meccanica quantistica conosciuta come *interferenza* [10].
Poniamo $N = 2$ e definiamo la nostra funzione $G(\mathbf{f})$ come

$$G(\mathbf{f}) \equiv f(0) \oplus f(1) \tag{3.24}$$

Prepariamo la nostra macchina nel seguente stato iniziale:

$$|0, 1\rangle \tag{3.25}$$

dove il qubit nello slot i è inizializzato a zero, mentre il qubit nello slot j è inizializzato a 1. Definiamo il programma seguente:

$$\Phi(H, i) \cdot \Phi(H, j) \cdot \pi(f, i, j) \cdot \Phi(H, i) \tag{3.26}$$

dove H è la matrice di Hadamard. Per cui avremo che (mostriamo per semplicità solo gli slot i e j):

$$|0, 1\rangle \xrightarrow{\Phi(H,i)} \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,|1\rangle \xrightarrow{\Phi(H,j)} \tfrac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = \tag{3.27}$$

$$= \tfrac{1}{2}(|0, 0\rangle - |0, 1\rangle + |1, 0\rangle - 1\,|1, 1\rangle \tag{3.28}$$

A questo punto eseguiamo la computazione della funzione $f(x)$ ottenendo:

$$\frac{1}{2}(|0, f(0)\rangle - |0, 1 \oplus f(0)\rangle + |1, f(1)\rangle - |1, 1 \oplus f(1)\rangle) \tag{3.29}$$

Possiamo raccogliere nel seguente modo:

$$\frac{1}{2}[|0\rangle\,(|f(0)\rangle - |1 \oplus f(0)\rangle) + |1\rangle\,(|f(1)\rangle - |1 \oplus f(1)\rangle)] \tag{3.30}$$

Notiamo che possiamo avere 4 casi differenti, in base ai valori di $f(0)$ e $f(1)$:

1. $f(0) = f(1) = 0$

$$\frac{1}{2}[|0\rangle (|0\rangle - |1\rangle) + |1\rangle (|0\rangle - |1\rangle)] = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) \tag{3.31}$$

2. $f(0) = f(1) = 1$

$$\frac{1}{2}[|0\rangle (|1\rangle - |0\rangle) + |1\rangle (|1\rangle - |0\rangle)] = -\frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) \tag{3.32}$$

3. $f(0) = 0, f(1) = 1$

$$\frac{1}{2}[|0\rangle (|0\rangle - |1\rangle) + |1\rangle (|1\rangle - |0\rangle)] = \frac{1}{2}(|0\rangle - |1\rangle)(|0\rangle - |1\rangle) \tag{3.33}$$

4. $f(0) = 1, f(1) = 0$

$$\frac{1}{2}[|0\rangle (|1\rangle - |0\rangle) + |1\rangle (|0\rangle - |1\rangle)] = -\frac{1}{2}(|0\rangle - |1\rangle)(|0\rangle - |1\rangle) \tag{3.34}$$

Possiamo raggruppare questi quattro risultati nel seguente modo:

$$\pm\frac{1}{2}(\overbrace{|0\rangle + |1\rangle}^{slot\ i})(\overbrace{|0\rangle - |1\rangle}^{slot\ j}) \quad \text{se} \quad f(0) = f(1) \tag{3.35}$$

$$\pm\frac{1}{2}(|0\rangle - |1\rangle)(|0\rangle - |1\rangle) \quad \text{se} \quad f(0) \neq f(1) \tag{3.36}$$

Ora eseguiamo l'ultima parte del programma applicando la trasformazione di Hadamard H al qubit dello slot i-esimo e otteniamo:

$$\pm\frac{1}{\sqrt{2}}|0\rangle (|0\rangle - |1\rangle) \quad \text{se} \quad f(0) = f(1) \tag{3.37}$$

$$\pm\frac{1}{\sqrt{2}}|1\rangle (|0\rangle - |1\rangle) \quad \text{se} \quad f(0) \neq f(1) \tag{3.38}$$

Notiamo che il contenuto dello slot i-esimo altro non è che il risultato di $G(\mathbf{f}) = f(0) \oplus f(1)$, per cui il nostro stato finale è dato da:

$$\pm|f(0) \oplus f(1)\rangle \frac{(|0\rangle - |1\rangle)}{\sqrt{2}} \tag{3.39}$$

In questo modo, sfruttando il parallelismo quantistico combinato con l'azione della trasformazione H, siamo in grado di determinare una *proprietà globale* di $f(x)$ (in questo caso il valore di $f(0) \oplus f(1)$) eseguendo solamente una valutazione di $f(x)$. Classicamente avremmo dovuto eseguire due valutazione separate per $f(x)$, impiegando più tempo per ottenere l'esito voluto [10].

Conclusioni

La macchina di Turing quantistica descritta da Deutsch in [3] è il primo vero modello completo di computazione quantistica [10], così come la macchina di Turing classica è il primo vero modello per la computazione classica. Tuttavia tali modelli sono altamente idealizzati: dobbiamo cercare un modello computazionale più concreto.

I computer classici vengono descritti con il modello dei *circuiti logici*: grazie alla congettura di Church-Turing questo modello è completamente equivalente dal punto di vista computazionale al modello della macchina di Turing, ma è di natura più concreta ed è più versatile in molte applicazioni. Gli elementi essenziali del modello a circuiti logici sono due: le porte logiche (chiamate anche gate), che eseguono i compiti computazionali e i collegamenti fra esse, che trasmettono l'informazione codificata in bit [10]. Una porta logica in generale può essere definita come una funzione $f : \{0,1\}^n \mapsto \{0,1\}^m$ che prende in input n bits e produce un output a m bits. In Fig.3.1 sono indicati i simboli circuitali delle porte più comuni .
La porta logica più semplice è la porta NOT che ammette in input un solo bit a e restituisce in output $f_{NOT}(a) = 1 \oplus a$, dove $\oplus$ indica l'addizione modulo 2. Tra le porte logiche più comuni che ammettono in input due bit troviamo:

- Porta AND: restituisce in output 1 se entrambi i bit in ingresso sono 1 altrimenti restituisce 0.

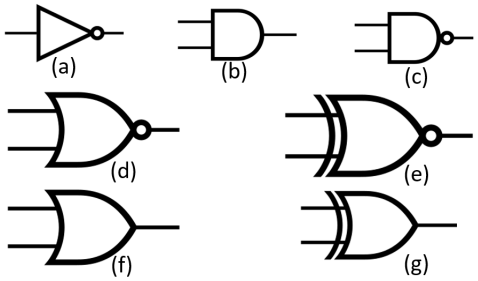

Figura 3.1: Simboli circuitali per le porte logiche più comuni:(a)-NOT, (b)-AND, (c)-NAND, (d)-NOR, (e)-XNOR, (f)-OR, (g)-XOR.

- Porta OR: restituisce in output 1 se almeno un bit in ingresso è 1.
- Porta XOR: calcola l'addizione modulo 2 tra i due bit in ingresso.

Applicando all'output di queste tre porte la porta NOT otteniamo rispettivamente le porte logiche NAND, NOR e XNOR. Particolare importanza ha la porta NAND: si dimostra che è una porta logica universale, cioè con una combinazione opportuna di porte NAND possiamo ottenere qualsiasi funzione logica.

Mostriamo in Fig.3.2 un esempio di circuito logico chiamato *full-adder*. Tale circuito

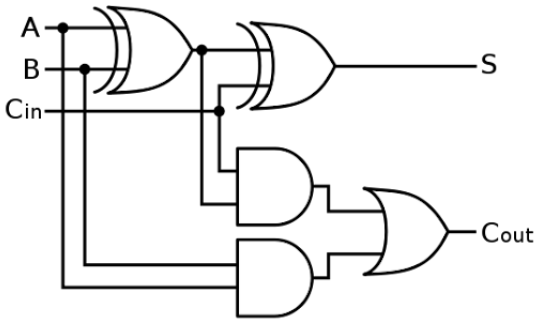

Figura 3.2: Esempio di un modello circuitale: in questo caso viene implementato un sommatore (full-adder).

ammette in input tre bit (A, B, C_{in} e restituisce in output due bit (S, C_{out}): A e B sono due bit di cui vogliamo calcolare la somma S, C_in è un eventuale riporto proveniente da somme precedenti mentre C_{out} è l'eventuale riporto della somma di A, B, C_{in}. Formalizzare un algoritmo equivalente per una macchina di Turing sarebbe molto più complicato.

Per quanto riguarda la computazione quantistica, troviamo un modello analogo ai circuiti logici classici: i circuiti quantistici. Anche in questo caso gli elementi principali sono due: le porte logiche quantistiche e i collegamenti fra esse. In questo caso l'informazione è codificata in qubit.
Una porta logica quantistica a singolo qubit è una qualsiasi trasformazione unitaria a singolo qubit: a differenza del caso classico in cui abbiamo una sola porta logica a singolo bit, nel caso quantistico abbiamo infinite porte quantistiche a singolo qubit. Una porta logica quantistica viene rappresentata da una matrice unitaria. Ad esempio vediamo come costruire la porta NOT quantistica (QNOT). SI richiede che:

$$|0\rangle \xrightarrow{QNOT} |1\rangle \tag{3.40}$$

$$|1\rangle \xrightarrow{QNOT} |0\rangle \tag{3.41}$$

e più in generale

$$\alpha |0\rangle + \beta |1\rangle \xrightarrow{QNOT} \alpha |1\rangle + \beta |0\rangle \tag{3.42}$$

Notiamo che la matrice di Pauli σ_x rappresenta correttamente la porta quantistica QNOT:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3.43}$$

Ad ogni matrice unitaria complessa 2x2 è associato il relativo gate quantistico a singolo qubit. Particolare importanza ha il gate di Hadamard, che applica al qubit in input la trasformazione di Hadamard:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.44}$$

Il gate di Hadamard offre un primo esempio della peculiarità della computazione quantistica: infatti permette di mappare gli stati $|0\rangle$ e $|1\rangle$ in una sovrapposizione di stati

$$|0\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle \tag{3.45}$$

$$|1\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle \tag{3.46}$$

che classicamente non sarebbe possibile ottenere.
Grazie al teorema (2.32) siamo in grado di definire un set finito di porte logiche a singolo qubit in grado di generare qualsiasi trasformazione unitaria.
Anche la funzione di misura perfetta definita in (2.21) trova una sua corrispondente porta logica: il CNOT (*controlled NOT*). Il CNOT è rappresentato da una matrice unitaria complessa 4x4, dal momento che agisce su due qubit e nella base computazionale $\{|0,0\rangle, |0,1\rangle, |1,0\rangle, |1,1\rangle\}$ ha la seguente forma:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.47}$$

Notiamo che l'azione del CNOT riproduce la funzione di misura perfetta:

$$|a, b\rangle \xrightarrow{CNOT} |a, a \oplus b\rangle \tag{3.48}$$

Attraverso il formalismo dei circuiti logici si può dare una dimostrazione [10] del fatto che sia possibile implementare circuiti quantistici in grado di simulare con precisione arbitraria qualsiasi trasformazione unitaria a L-qubit utilizzando porte logiche da un set

finito. In particolare si dimostra che la porta di Hadamard e la porta CNOT insieme ai seguenti gate quantistici

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \tag{3.49}$$

formano un set universale. Grazie alla combinazione di H, CNOT, S e T siamo in grado di simulare l'azione di qualsiasi trasformazione unitaria a L-qubit con precisione arbitraria [10], risultato equivalente alla dimostrazione data alla fine della sezione (2.4).

In Fig. 3.3 è mostrato il circuito quantistico che permette di realizzare l'algoritmo di Deutsch esposto nella sezione 3.3.1. Notiamo che con U_f viene indicata la porta logica che permette di valutare la funzione f sul qubit di input $|x\rangle$, in particolare tale gate è equivalente al programma (2.17) per una macchina di Turing:

$$U_f \, |x, y\rangle = |x, y \oplus f(x)\rangle \tag{3.50}$$

Il circuito in figura è del tutto equivalente al programma definito nella sezione 3.3.1. Lo stato in input è dato da

$$|\psi_0\rangle = |0, 1\rangle \tag{3.51}$$

mentre lo stato in output è dato da:

$$|\psi_3\rangle = \pm \, |f(0) \oplus f(1)\rangle \, (\frac{|0\rangle - |1\rangle}{\sqrt{2}}) \tag{3.52}$$

Abbiamo visto dunque che il modello a circuiti è decisamente più semplice e concreto del modello computazionale della macchina di Turing.
Il modello della macchina di Turing viene ancora utilizzato nella disciplina nota come *teoria della complessità computazionale*, che ha lo scopo di determinare le risorse spaziali,

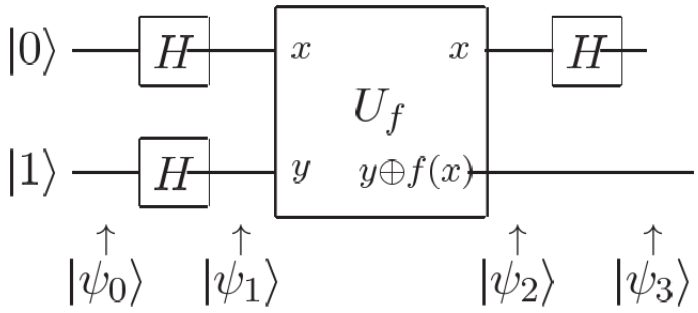

Figura 3.3: Esempio di circuito logico quantistico che implementa l'algoritmo di Deutsch. In particolare con H vengono indicati i gate di Hadamard, mentre con U_f viene indicata una porta logica in grado di valutare la funzione f.

temporali ed energetiche per eseguire un dato algoritmo [10].

L'algoritmo di Deutsch rappresenta solo un piccolo esempio della capacità di calcolo di un computer quantistico. Nel corso degli anni sono stati sviluppati algoritmi quantistici in grado di eseguire in modo efficiente alcuni compiti computazionali considerati intrattabili dal punto di vista classico [10]. Un esempio importante è dato dal problema della fattorizzazione di numeri interi. Classicamente tale problema può essere risolto solamente in un tempo che aumenta esponenzialmente con la grandezza del numero da fattorizzare [10]. Nel contesto della computazione quantistica l'*algoritmo di fattorizzazione di Shor* [12] permette invece di fattorizzare un numero intero in un tempo che è *esponenzialmente* più veloce del tempo di esecuzione di qualsiasi algoritmo di fattorizzazione classico. L'ingrediente fondamentale di tale algoritmo e in generale di molti algoritmi quantistici è la *quantum Fourier transform* che permette di eseguire la trasformata di Fourier su ampiezze di probabilità quantistiche. Classicamente la trasformata di Fourier discreta è utilizzata in applicazioni di analisi di dati reali: ad esempio gli algoritmi di riconoscimento vocale come primo step eseguono la trasformata del suono digitalizzato [10]. La versione classica di tale algoritmo ha un tempo di esecuzione esponenziale, mentre la quantum Fourier transform richiede un tempo polinomiale. Tuttavia la versione quantistica dell'algoritmo non può essere utilizzata per velocizzare la versione classica [10]. L'utilità dell'algoritmo di quantum Fourier transform risiede nel fatto che è la chiave per eseguire la procedura nota come *phase estimation* che permette di approssimare gli autovalori di operatori unitari arbitrari: tale procedura non solo è interessante da un punto di vista fisico (permette di risolvere problemi agli autovalori) ma anche da un punto di vista applicativo, in quanto si dimostra che alcuni problemi computazionali, come il già citato problema di fattorizzazione, sono riconducibili ad essa [10].
Un altro importante algoritmo quantistico è l'*algoritmo di ricerca di Grover* che permette di eseguire ricerche in un database indifferenziato in modo più efficiente rispetto agli algoritmi classici. Ad esempio volendo cercare un numero di telefono in un elenco telefonico di N persone tramite un algoritmo classico, in media abbiamo bisogno di $N/2$ valutazioni. L'algoritmo quantistico di Grover sfruttando la sovrapposizione di stati quantistici permette di risolvere il problema in un numero di operazioni nell'ordine di $\sqrt{N}$, risultando più veloce dell'algoritmo classico [7].

Possiamo concludere con l'affermare che il campo di studi della computazione quantistica è un campo molto vasto e comprendere fino in fondo le potenzialità di tale paradigma è fondamentale per gli sviluppi della tecnologia del futuro. Ciò rende lo studio di tale disciplina affascinante e stimolante.

Appendice A

Matrici densità

La meccanica quantistica può essere formulata utilizzando il formalismo delle matrici densità, al posto del formalismo bra-ket [10]: dal punto di vista matematico questi due formalismi sono completamente equivalenti.
Consideriamo un sistema quanto-meccanico chiuso che può esistere in un determinato stato $|\psi_i\rangle$ con probabilità p_i. L'insieme di tutte le possibili coppie $\{p_i, |\psi_i\rangle\}$ per il nostro sistema è chiamato *ensemble di stati puri* [10]. La matrice densità del sistema è definita come:

$$\rho \equiv \sum_i p_i \, |\psi_i\rangle \, \langle\psi_i| \tag{A.1}$$

In rappresentazione di Schroedinger l'evoluzione del ket di stato è determinata da un operatore unitario $\mathbf{U}(t)$, tale per cui:

$$|\psi(t)\rangle = \mathbf{U}(t) \, |\psi(0)\rangle \tag{A.2}$$

Possiamo dunque dedurre l'evoluzione temporale della matrice densità del sistema:

$$\rho(t) = \sum_i p_i \, |\psi_i(t)\rangle \, \langle\psi_i(t)| = \sum_i p_i \mathbf{U}(t) \, |\psi_i(0)\rangle \, \langle\psi_i| \, \mathbf{U}^\dagger(t) = \mathbf{U}(t)\rho(0)\mathbf{U}^\dagger(t) \tag{A.3}$$

I processi di misura possono essere descritti nel seguente modo. Supponiamo che lo stato iniziale del nostro sistema sia $|\psi_i\rangle$. Vogliamo eseguire una misura descritta dall'operatore $\mathbf{M}_m$. La probabilità di ottenere il risultato m è data da:

$$p(m|i) = \langle\psi_i|\mathbf{M}_m^\dagger \mathbf{M}_m|\psi_i\rangle = tr(\mathbf{M}_m^\dagger \mathbf{M}_m \, |\psi_i\rangle \, \langle\psi_i|) \tag{A.4}$$

e applicando la legge della probabilità totale [10] si ottiene

$$p(m) = \sum_i p(m|i)p_i = \sum_i tr(\mathbf{M}_m^\dagger \mathbf{M}_m \, |\psi_i\rangle \, \langle\psi_i|)p_i = tr(\mathbf{M}_m^\dagger \mathbf{M}_m \rho) \tag{A.5}$$

A questo punto possiamo chiederci quale sia la matrice densità del sistema a seguito del processo di misura. Utilizzando il formalismo bra-ket [10] sappiamo che l'esecuzione di una misura $\mathbf{M}_m$ sullo stato $|\psi_i\rangle$ porta allo stato:

$$|\psi_i^m\rangle = \frac{\mathbf{M}_m |\psi_i\rangle}{\sqrt{\langle\psi_i|\mathbf{M}_m^\dagger \mathbf{M}_m|\psi_i\rangle}} \tag{A.6}$$

Dunque al termine del processo di misura abbiamo l'ensemble $\{p(m|i), |\psi_i^m\rangle\}$, la cui matrice densità associata è per definizione:

$$\rho_m = \sum_i p(m|i)\, |\psi_i^m\rangle \langle\psi_i^m| = \sum_i p(m|i) \frac{\mathbf{M}_m |\psi_i\rangle \langle\psi_i^m| \mathbf{M}_m^\dagger}{\langle\psi_i|\mathbf{M}_m^\dagger \mathbf{M}_m|\psi_i\rangle} = \frac{\mathbf{M}_m \rho \mathbf{M}_m^\dagger}{tr(\mathbf{M}_m^\dagger \mathbf{M}_m \rho)} \tag{A.7}$$

Attraverso il formalismo delle matrici di densità possiamo riformulare i postulati alla base della meccanica quantistica. Inoltre possiamo rendere questo nuovo formalismo totalmente indipendente dal formalismo dei ket di stato. Per come abbiamo introdotto le matrici densità, esse emergono da un ensemble di stati. Possiamo sviluppare tale formalismo in modo intrinseco, caratterizzando direttamente l'operatore densità senza fare affidamento al concetto di ensemble. In particolare vale il seguente

Teorema A.0.1. *Un qualsiasi operatore ρ autoaggiunto è una matrice densità associabile a qualche ensemble di stati puri $\{p_i, |\psi_i\rangle\}$ se e solo se:*

1. *$tr(\rho) = 1$;*
2. *ρ è un operatore positivo.*

Tale teorema offre una caratterizzazione delle matrici densità intrinseca agli operatori stessi: possiamo definire una matrice densità ρ come un operatore positivo autoaggiunto la cui traccia sia uguale a uno. Con tale definizione siamo in grado di riformulare i postulati della teoria quantistica:

1. **Postulato 1:** ad ogni sistema fisico isolato è associato uno spazio di Hilbert complesso, conosciuto come spazio degli stati del sistema. Lo stato del sistema è completamente descritto dalla matrice densità ρ che agisce sugli elementi dello spazio degli stati;
2. **Postulato 2:** l'evoluzione temporale di un sistema quanto-meccanico chiuso è governata da trasformazioni unitarie $\mathbf{U}$. In particolare si ha:

$$\rho(t_2) = \mathbf{U}(t_2 - t_1)\rho(t_1)\mathbf{U}^\dagger(t_2 - t_1) \tag{A.8}$$

3. **Postulato 3:** i processi di misura sono descritti da una collezione $\{\mathbf{U}_m\}$ di operatori di misura. Tali operatori agiscono sullo spazio degli stati del sistema. L'indice m si riferisce all'esito della misura che si può ottenere durante un esperimento. Se lo stato del sistema prima della misura è descritto da ρ, allora la probabilità di ottenere il risultato m è dato da:

$$p(m) = tr(\mathbf{M}_m^\dagger \mathbf{M}_m \rho) \tag{A.9}$$

e lo stato del sistema dopo la misura è

$$\rho_m = \frac{\mathbf{M}_m \rho \mathbf{M}_m^\dagger}{tr(\mathbf{M}_m^\dagger \mathbf{M}_m \rho)}. \tag{A.10}$$

Gli operatori di misura soddisfano la relazione di completezza:

$$\sum_m \mathbf{M}_m^\dagger \mathbf{M}_m = \hat{1}. \tag{A.11}$$

4. **Postulato 4:** lo spazio degli stati di un sistema composto è dato dal prodotto tensoriale degli spazi degli stati di ciascuna componente del sistema. Supponendo di avere n sottosistemi, lo stato complessivo del sistema è descritto da $\rho = \rho_1 \otimes \rho_2 \otimes \cdots \otimes \rho_n$.

All'interno di tale formalismo troviamo la distinzione fra due tipi di stato:

1. **Stati puri**: un sistema è in uno stato puro quando possiamo identificare con certezza il suo stato tramite un ket $|\psi\rangle$. In questo caso la matrice densità è semplicemente $\rho = |\psi\rangle \langle\psi|$. Si dimostra che $tr(\rho^2) = 1$.

2. **Stati misti**: un sistema è in uno stato misto quando è in una *miscela* di stati puri. In questo caso si dimostra che $tr(\rho^2) < 1$.

Appendice B

Sistemi aperti

Consideriamo un sistema finito A e denotiamo con $\mathcal{H}_\mathcal{A}$ lo spazio di Hilbert associato di dimensione L. Poniamo A a contatto con l'ambiente esterno, e sia $\mathcal{H}_\mathcal{A}$ lo spazio di Hilbert dell'ambiente che dato l'elevato numero di gradi di libertà possiamo assumere come spazio di dimensione infinita. Lo spazio di Hilbert dei due sistemi combinati è dato dal prodotto tensoriale dei due spazi individuali [9]:

$$\mathcal{H} = \mathcal{H}_\mathcal{A} \otimes \mathcal{H}_\mathcal{B} \tag{B.1}$$

All'interno di $\mathcal{H}$ possiamo individuare un ensemble $\{p_a, |\psi_a\rangle\}$ di stati puri $|\psi_a\rangle \in \mathcal{H}$. Tali stati possono essere espansi in una base di $\mathcal{H}$:

$$|\psi_a\rangle = \sum_{i,j} \lambda_{a;i,j} |i\rangle_A \otimes |j\rangle_B \tag{B.2}$$

In questo caso la matrice densità è data da:

$$\rho = \sum_a p_a |\psi_a\rangle \langle\psi_a| = \sum_a p_a (\sum_{i,j} \lambda_{a;i,j} |i\rangle_A \otimes |j\rangle_B)(\sum_{i,j} \lambda^*_{a;i,j} \langle i|_A \otimes \langle j|_B) \tag{B.3}$$

Perciò possiamo scrivere ogni matrice densità nello spazio $\mathcal{H}$ come:

$$\rho = \sum_{ij\mu\nu} k_{ij\mu\nu} |i\rangle_A \langle j| \otimes |\mu\rangle_B \langle\nu| \tag{B.4}$$

con $k_{ij\mu\nu} = \sum_a p_a \lambda_{a;i,\mu} \lambda^*_{a;j,\nu}$.
Il nostro interesse è quello di descrivere solo il sistema A, per cui dobbiamo trovare un modo per togliere l'ambiente dalla nostra descrizione. A tale scopo definiamo l'operatore *traccia parziale*.

Definizione B.0.1. *Sia $O = M_A \otimes M_B$ operatore nello spazio $\mathcal{H} = \mathcal{H}_\mathcal{A} \otimes \mathcal{H}_\mathcal{B}$. Definiamo* traccia parziale *l'operatore:*

$$Tr_B(M_A \otimes M_B) \equiv M_A Tr(M_B) = M_A \sum_j \langle j|M_B|j\rangle = \sum_j \langle j|M_A \otimes M_B|j\rangle$$

dove $\{|j\rangle\}$ *indica una base dello spazio* $\mathcal{H}_\mathcal{B}$.

Per definizione la traccia parziale è un operatore lineare che mappa operatori che agiscono sullo spazio $\mathcal{H}$ in operatori che agiscono su $\mathcal{H}_\mathcal{A}$.
I ket $|\psi_a\rangle \in \mathcal{H}$ hanno un'evoluzione unitaria, in quanto consideriamo il sistema composto come un sistema chiuso. Dunque avremo:

$$|\psi(t)\rangle = \mathbf{U}(t)\,|\psi(0)\rangle \implies \rho(t) = \mathbf{U}(t)\rho(0)\mathbf{U}^\dagger(t) \tag{B.5}$$

Per descrivere l'evoluzione dinamica di A dobbiamo determinare $\rho_a(t)$ utilizzando l'operatore traccia parziale:

$$\rho_A(t) = Tr_B(\rho(t)) = Tr_B(\mathbf{U}(t)\rho(0)\mathbf{U}^\dagger(t)) \tag{B.6}$$

Assumiamo che all'istante iniziale sistema e ambiente siano completamente disaccoppiati [9]:

$$\rho(0) = \rho_A(0) \otimes \rho_B(0) \tag{B.7}$$

cioè assumiamo che durante la preparazione dello stato iniziale sia trascurabile l'interazione fra apparato preparatore, sistema e ambiente [3]. Dal momento che la matrice densità dell'ambiente è positiva e normalizzata possiamo effettuare una decomposizione spettrale in una base ortonormale di $\mathcal{H}_\mathcal{B}$ con autovalori non negativi [9]:

$$\rho_B(0) = \sum_\nu p_\nu\,|\nu\rangle_B\,\langle\nu| \tag{B.8}$$

Per cui calcolando la matrice densità del nostro sistema otteniamo:

$$\begin{aligned}\rho_A(t) &= \sum_\mu \langle\mu|\,[\mathbf{U}(t)\rho_A(0) \otimes \sum_\nu p_\nu\,|\nu\rangle\,\langle\nu|\,\mathbf{U}^\dagger(t)]\,|\mu\rangle = \\ &= \sum_{\mu\nu} \sqrt{p_\nu}\,\langle\mu|\,\mathbf{U}(t)\,|\nu\rangle\,\rho_A(0)\sqrt{p_\nu}\,\langle\nu|\,\mathbf{U}^\dagger(t)\,|\mu\rangle = \\ &= \sum_{\mu\nu} \mathbf{K}_{\mu\nu}(t)\rho_A(0)\mathbf{K}^\dagger_{\mu\nu}(t)\end{aligned} \tag{B.9}$$

dove definiamo la famiglia di operatori $\{\mathbf{K}_{\mu\nu}(t)\}$ detti operatori di Kraus [9] che agiscono sulla matrice densità del sistema, con $\mathbf{K}_{\mu\nu}(t) = \sqrt{p_\nu}\,\langle\mu|\,\mathbf{U}(t)\,|\nu\rangle$. Mostriamo un'importante proprietà di tale famiglia di operatori. Per definizione abbiamo che:

$$\sum_{\mu\nu} \mathbf{K}^\dagger_{\mu\nu}(t)\mathbf{K}_{\mu\nu}(t) = \sum_{\mu\nu} p_\nu\,\langle\nu|\,\mathbf{U}^\dagger(t)\,|\mu\rangle\,\langle\mu|\,\mathbf{U}(t)\,|\nu\rangle = \tag{B.10}$$

$$= \sum_\nu p_\nu\,\langle\nu|\,\mathbf{U}^\dagger(t)(\sum_\mu |\mu\rangle\,\langle\mu|\,)U(t)\,|\nu\rangle = \sum_\nu p_\nu\,\langle\nu|\,\mathbf{U}^\dagger(t)\mathbf{U}(t)\,|\nu\rangle = \tag{B.11}$$

$$= \sum_\nu p_\nu\,\langle\nu|\nu\rangle = \sum_\nu p_\nu = 1 \tag{B.12}$$

per cui gli operatori di Kraus soddisfano:

$$\sum_{\mu\nu} \mathbf{K}^{\dagger}_{\mu\nu}(t)\mathbf{K}_{\mu\nu}(t) = \hat{1} \tag{B.13}$$

con $\hat{1}$ operatore identità. Utilizzando la notazione indiciale e la convenzione di Einstein per la somma di indici ripetuti possiamo riscrivere (B.9) come:

$$(\rho_A(t))_a{}^b = \sum_{\mu\nu} (\mathbf{K}_{\mu\nu}(t))_a{}^c (\rho_A(0))_c{}^d (\mathbf{K}_{\mu\nu}(t))^b{}_d \tag{B.14}$$

Definiamo a questo punto l'operatore di superscattering [3]:

$$\mathbf{S}_a{}^{bc}{}_d = \sum_{\mu\nu} (\mathbf{K}_{\mu\nu}(t))_a{}^c (\mathbf{K}_{\mu\nu}(t))^b{}_d \tag{B.15}$$

e otteniamo

$$(\rho_A(t))_a{}^b = \mathbf{S}_a{}^{bc}{}_d (\rho_A(0))_c{}^d \tag{B.16}$$

per cui la dinamica del sistema viene descritta in modo completo dall'operatore di superscattering, che altro non è che l'operatore $\mathbf{U}$ mappato sullo spazio $\mathcal{H}_{\mathcal{A}}$.

Bibliografia

[1] Charles Bennett. «Logical Reversibility of Computation». In: *IBM Journal of Research and Development* 17 (dic. 1973), pp. 525–532. DOI: 10.1147/rd.176.0525.

[2] Heinz-Peter Breuer e Francesco Petruccione. *Concepts and methods in the theory of open quantum systems*. URL: http://arxiv.org/abs/quant-ph/0302047v1.

[3] David Deutsch. «Quantum theory, the Church-Turing principle and the universal quantum computer». In: *Proceedings of the Royal Society of London* (1985).

[4] David Deutsch, Artur Ekert e Rossella Lupacchini. «Machines, Logic and Quantum Physics». In: *Bulletin of Symbolic Logic* 6 (dic. 1999). DOI: 10.2307/421056.

[5] Joachim Draeger. *Quantum Turing Machines*. Giu. 2017. DOI: 10.13140/RG.2.2.21211.36640.

[6] Richard Feynman. «Simulating Physics with Computers». In: *International Journal of Theoretical Physics* 21 (giu. 1982).

[7] Lov Grover. «Fast quantum mechanical algorithm for database search». In: *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing* (giu. 1996). DOI: 10.1145/237814.237866.

[8] Harry Lewis e Christos Papadimitriou. *Elements of the theory of computation*. Upper Sandle river, New Jersey: Prentice Hall, 1998.

[9] Daniel A. Lidar. *Lecture Notes on the Theory of Open Quantum Systems*. URL: arXiv:1902.00967v2.

[10] M. A. Nielsen e Chuang I. L. *Quantum Computation and Quantum Information*. Cambridge: Cambridge university press, 2010.

[11] Yu Shi. *Remarks on universal quantum computer*. URL: http://arxiv.org/abs/quant-ph/9908074v5.

[12] Peter W. Shor. «Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer». In: *SIAM Journal on Computing* 26.5 (ott. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172.

[13] Mario Stipević. «Quantum random number generators and their use in cryptography». In: *2011 Proceedings of the 34th International Convention MIPRO* (2011), pp. 1474–1479.

[14] Cornell University. *Notes on Turing Machines*. 2009. URL: https://www.cs.cornell.edu/courses/cs4820/2010sp/handouts/turingm.pdf.