# Introduzione

Nei quarantacinque anni trascorsi dall'invenzione del transistor il numero di dispositivi realizzabili su un singolo chip di silicio è aumentato di otto ordini di grandezza. I disegnatori della prossima generazione di dispositivi a semiconduttori commerciali potranno probabilmente utilizzare ancora i concetti classici di isolante, impedenza e le altre usuali proprietà elettriche dei materiali, ma nei laboratori di ricerca si sono già realizzati circuiti ad integrazione così elevata da rendere rilevanti effetti tipicamente quantistici, come ad esempio il tunnelling degli elettroni tra le piste [Keyes]. Questo fatto, in unione con la ricerca, nell'ambito della matematica computazionale, di metodi per il superamento dell'intrattabilità dei problemi a complessità esponenziale, ha portato ad una nuova attenzione per la fisica dei dispositivi preposti ad eseguire in pratica i calcoli. Ci si è in particolare domandato se sia possibile effettuare della computazione con un sistema puramente quantistico e quali eventuali vantaggi ciò comporti rispetto all'utilizzo di un sistema classico [Lloyd].

I principali aspetti della meccanica quantistica rilevanti ai fini del trattamento dell'informazione sono i seguenti [Bennett]:

1. La ***sovrapposizione*** degli stati: un computer quantistico può esistere in una combinazione lineare arbitraria a coefficienti complessi di stati booleani, che evolvono in parallelo secondo una trasformazione unitaria;

2. L'***interferenza***: le diverse computazioni effettuate in parallelo grazie alla sovrapposizione, analoghe ai diversi percorsi dei fotoni in un interferometro, possono rinforzarsi le une con le altre o annullarsi a vicenda;

3. La ***correlazione***: ad uno stato definito del calcolatore quantistico possono non corrispondere stati definiti delle sue singole parti;
4. La ***non-clonabilità*** e l'***indeterminazione***: uno stato quantistico sconosciuto non può essere copiato esattamente (clonato) né può essere osservato senza essere perturbato.

Queste caratteristiche danno origine ad una sorta di parallelismo intrinseco che in alcuni casi conduce ad una drastica riduzione della complessità computazionale di problemi classicamente intrattabili. Il successo più spettacolare finora conseguito è quello di Peter W. Shor, degli AT&T Bell Labs, che ha escogitato un algoritmo quantistico per la scomposizione di un intero in fattori primi con complessità polinomiale, ovvero una procedura il cui tempo di esecuzione è una funzione polinomiale del numero di cifre dell'ingresso. Per contrasto, i migliori algoritmi classici conosciuti hanno complessità esponenziale e impiegherebbero un tempo dell'ordine del miliardo di anni per fattorizzare un numero di duecento cifre.

Se ampie prospettive di sviluppo sembrano sorgere dall'incontro della teoria della computazione con la meccanica quantistica, quest'ultima pare sul punto di subentrare alla prima in un settore tradizionalmente legato a tematiche computazionali: la crittografia. Il principio di indeterminazione di Heisenberg, che stabilisce che l'osservazione di un sistema ne perturba necessariamente lo stato, è posto dalla crittografia quantistica a garanzia della totale riservatezza di una comunicazione. Due interlocutori che utilizzino un protocollo crittografico quantistico hanno infatti la certezza di poter rilevare le alterazioni introdotte nello stato dei portatori fisici dell'informazione (ad esempio fotoni singoli) da una osservazione non autorizzata del canale.

L'importanza della crittografia è sempre crescente nella società odierna basata sulle reti di dati e l'alternativa quantistica, già al presente realizzabile a livello sperimentale, non solo è resa attraente dalla sua assoluta sicurezza, ma è

anche supportata dalle conquiste teoriche della computazione quantistica. L'algoritmo di Shor, una volta implementato, permetterebbe infatti di violare facilmente il più comune cifrario oggi in uso, rendendolo inservibile.

Questo suaccennato non è l'unico punto di contatto tra la computazione e la crittografia quantistiche; nel corso della nostra trattazione metteremo in risalto altre interazioni tra i due campi di ricerca, quali lo studio dei codici di correzione d'errore quantistici o il problema dell'intercettazione quantistica.

Prima di procedere ad affrontare la computazione quantistica presentiamo ora nel seguente capitolo una breve introduzione alla teoria classica della computazione, senza la quale non sarebbe possibile apprezzare le differenze tra il modello di calcolo classico ed il nuovo modello quantistico.

# La teoria classica della computazione

## *Cenni storici*

La storia degli algoritmi [Knuth] è antichissima e risale almeno ad Euclide, che elaborò il ben noto algoritmo per il calcolo del massimo comun divisore di due interi. Tuttavia, come spesso è accaduto nello sviluppo della matematica, questo fondamentale concetto non è stato formalizzato che in epoca molto recente.

L'interesse sistematico in questa direzione può essere fatto risalire agli inizi di questo secolo, quando Hilbert propose all'attenzione dei matematici quello che divenne poi noto come "il decimo problema di Hilbert" o problema della decisione. Egli si domandava se esistesse una procedura meccanica generale mediante la quale stabilire, in un numero finito di passi, se una data equazione diofantea[1] possegga o meno soluzioni intere ([Hofstadter], [Davis]).

All'epoca non era neppure ben chiaro cosa potesse intendersi per "procedura meccanica". Estremamente interessante è la proposta di Turing, avanzata nell'articolo *On computable numbers with an application to the Entscheidungsproblem (Sui numeri computabili con un'applicazione al problema della decisione,* 1936-37), in cui egli analizza i procedimenti computistici della mente umana scomponendoli nei loro elementi fondamentali e formalizzandoli nella *"macchina di Turing"* [Mangione]. Creata così una descrizione assiomatica e completamente astratta di procedura meccanica, ovvero

[1] Si tratta di un'equazione in cui viene uguagliato a zero un polinomio con coefficienti ed esponenti costanti ed interi.

dato un criterio per la computabilità, Turing riconduce il problema di Hilbert a quello dell'arresto dei programmi, del quale dimostra l'insolubilità (per una dimostrazione, si veda [Penrose]).

Infine la constatazione dell'equivalenza delle diverse definizioni di computabilità elaborate da Turing, Gödel, Markov e Church corroborò l'idea della sostanziale indifferenza del modello di calcolo meccanico adottato, espressa già nel 1936 nella famosa ***tesi di Church***, che possiamo enunciare così: "*Ogni funzione che può essere naturalmente ritenuta computabile può essere computata dalla macchina di Turing*". Questa non è, evidentemente, una asserzione matematica, ma esprime la convinzione che la macchina di Turing costituisca una "buona definizione" del concetto di computabilità.

## *La complessità computazionale*

La macchina di Turing deve il suo successo, tra gli altri possibili modelli, al fatto di costituire un prototipo essenziale abbastanza vicino al funzionamento effettivo di un calcolatore sequenziale. Si potrebbe dire, con più esattezza, che un moderno calcolatore ne è, per molti fini pratici, una buona approssimazione. Le memorie attualmente a disposizione sono infatti, in molti casi, un buon surrogato del nastro infinito della MdT, e la rapidità degli odierni processori è tale da rendere spesso trascurabili i tempi di attesa. Tuttavia, sotto questi aspetti, la semplice nozione di computabilità è troppo astratta e, spesso, insufficiente. La sensazione che qualsiasi problema computazionale ben posto possa essere affrontato non appena si disponga della potenza di calcolo sufficiente è fallace: tale potenza potrebbe in pratica richiedere risorse che l'intero universo non è in grado di fornire. Si rende perciò necessario uno studio della complessità, ovvero delle risorse di spazio e di tempo consumate dagli algoritmi, allo scopo di stabilirne l'*efficienza*. La ricerca di algoritmi efficienti costituisce, oltre ad una

necessità pratica, un modo di concepire il *principio di economia della scienza* suggerito alla fine del secolo scorso da Ernst Mach, secondo il quale la natura e la scienza cercano sempre le vie più semplici e dirette per raggiungere il loro scopo[1] [Chaitin].

Come affermato, gli studi di complessità sono rivolti alla determinazione delle risorse di tempo e spazio necessarie e sufficienti per risolvere un problema *rispetto a dati modelli di calcolo*. Ci occuperemo qui solo del tempo. Determinare un limite inferiore per la complessità di un problema è in generale molto difficile, in quanto richiede una dimostrazione esistenziale applicabile a tutti i possibili algoritmi. Una limitazione superiore si ottiene invece con la costruzione esplicita di una procedura risolutiva.

Per numerosi problemi, di natura essenzialmente combinatoria, si ritiene ragionevolmente che non esistano algoritmi (classici!) di complessità inferiore all'esponenziale. Essi sono perciò presumibilmente intrattabili, considerandosi trattabili, in teoria della complessità, i soli problemi per i quali è noto un algoritmo di complessità polinomiale nella dimensione delle istanze.

Per chiarire il significato pratico dell'*intrattabilità* consideriamo il ben noto problema del commesso viaggiatore: date $n$ città e le distanze che le separano tra loro, trovare il percorso di lunghezza minima che parte e arriva in una stessa città toccandole tutte una sola volta. La soluzione esaustiva si scontra con l'esplosione combinatoria del numero dei percorsi, che è pari a $(n-1)!$. Rapidi conti con un

[1] Non sorprende, a questo punto, l'esistenza di un apparato matematico ben sviluppato per lo studio della complessità degli algoritmi. Meno noti sono probabilmente i tentativi di fondare sul concetto di algoritmo una analisi quantitativa della complessità delle teorie scientifiche. Questo approccio è stato proposto nel 1960 da Ray J. Salomonoff [Leoncini]. Il suo studio muove dall'ipotesi che le osservazioni di uno scienziato siano codificabili come una stringa di bit. Lo scienziato cerca di spiegare i dati per mezzo di una teoria, che può essere considerata un algoritmo in grado di riprodurre la successione e di estenderla, ovvero di fornire previsioni. Data una serie di osservazioni, vi sono sempre più teorie possibili in competizione; il modello richiede che si scelga quella più semplice, il cui algoritmo è descrivibile da un minor numero di bit.

Tale concetto di "algoritmo minimale" è stato riproposto indipendentemente, nel 1965 circa, da G. Chaitin e N. Kolmogorov, ed è alla base della teoria algoritmica dell'informazione ([Chaitin2], [Chaitin3]).

calcolatore tascabile dimostrano che impiegando un microsecondo per analizzare ogni percorso e avendo a disposizione qualche minuto si potrebbe risolvere il problema per 13 città. Un computer mille volte più rapido impiegherebbe una ventina di minuti per 16 città, al prezzo di un notevole progresso tecnologico. I limiti del possibile non sono molto lontani: le minime dimensioni pensabili per un calcolatore sono ragionevolmente quelle delle particelle stabili presenti in natura ($10^{-15}$ metri), e anche in tal caso l'analisi di un percorso richiederebbe almeno il tempo necessario ad un raggio di luce ad attraversare il computer. Potremmo quindi analizzare un massimo di $3x10^{23}$ percorsi al secondo, e questo implica che il numero massimo di città trattabili è 27. Risulta quindi evidente come tempo esponenziale risulti sinonimo di intrattabilità.

Come accennato, un algoritmo è detto efficiente o veloce quando il tempo impiegato nell'esecuzione cresce non più rapidamente di una funzione polinomiale delle dimensioni dell'input, misurate in genere dal numero di bit *n* necessari a descriverlo. Un intero N, ad esempio, richiede $\log_2 N$ bit di memoria su un calcolatore; un algoritmo efficiente dovrà perciò soddisfare, supponendo che ogni operazione richieda un tempo costante, alla condizione

$$\#\text{ di passi per un ingresso } N \leq \text{poly}(\log N) \text{ per ogni } N^{1}$$

dove con poly(log N) indichiamo un polinomio nella variabile log N.

E' leggermente più difficile rendersi conto del motivo dell'equazione *polinomiale=trattabile*. In effetti, pur essendo $n^{100}$ un polinomio, sarebbe difficile definire efficiente un algoritmo che impieghi questo tempo per l'esecuzione. Ciononostante, l'esistenza di un algoritmo polinomiale per il problema del commesso viaggiatore dimostrerebbe l'esistenza di una qualche strategia "intelligente", diversa cioè dalla semplice enumerazione di tutte le possibili

[1] Il "per ogni N" è fondamentale. Fissato un intero N, l'algoritmo che risolve il problema decisionale "N è composto?" è la procedura : "print(SI')" oppure la procedura "print(NO)", entrambe con tempo di esecuzione indipendente da N. Uno di questi due programmi, anche se ignoriamo quale, *risolve* il problema per l'unico N fissato ed ha chiaramente la minima complessità computazionale.

alternative, che potrebbe in seguito essere ulteriormente migliorata[1]. Qualcosa del genere è accaduto, per esempio, per il problema della programmazione lineare.

Abbiamo in principio fatto riferimento ad un modello di calcolo prefissato. Tuttavia, non è difficile rendersi conto che un generico calcolo che richieda su un (modello di) calcolatore un tempo T può essere simulato da una macchina di Turing in un tempo $T^k$, dove k è una costante dipendente solo dal calcolatore ma non dal problema[2]. L'idea fondamentale è che la MdT può memorizzare sul nastro illimitato la descrizione completa dello stato interno del calcolatore e riprodurne l'evoluzione mediante la scomposizione dei suoi cicli complessi in passi elementari. Le nozioni di tempo polinomiale ed esponenziale hanno quindi un valore assoluto, indipendente dal modello di calcolo adottato e quindi anche da eventuali miglioramenti tecnologici nell'ambito della teoria classica. Quanto affermato ci porta alla formulazione moderna della ***tesi di Church: "ogni computazione effettuabile in qualsiasi modello può essere simulata su una macchina di Turing, e la simulazione conserva l'efficienza della computazione"*** [Ekert].

## *Classificazione dei problemi decisionali*

Per quanto riguarda lo studio della complessità, e soprattutto la determinazione di limitazioni inferiori della stessa, è sufficiente limitarsi all'analisi delle versioni decisionali dei problemi (ad esempio, in luogo del problema della fattorizzazione di un intero, consideriamo quello di decidere se è composto, che chiameremo COMPOSTO).

---

[1] L'Intelligenza Artificiale è in fondo quel settore dell'informatica che si occupa di elaborare strategie "intelligenti" per risolvere i problemi intrinsecamente combinatori tipicamente affrontati con successo dagli esseri viventi.

[2] Questo vale anche per i modelli circuitali, gli automi cellulari ecc.

La classe dei problemi decisionali risolvibile con tempo polinomiale nella dimensione viene normalmente denotata con **P**. Analogamente, la classe dei problemi decisionali risolvibile in tempo polinomiale con algoritmi non deterministici[1] viene indicata con **NP.** Essendo un algoritmo deterministico un caso particolare di algoritmo non deterministico si ha **P**$\subseteq$**NP**. **NP** è l'insieme dei teoremi la cui verità può essere verificata in tempo polinomiale.

Il problema COMPOSTO è evidentemente in **NP**[2]. Anche la versione decisionale del problema del commesso viaggiatore (TSP: Travelling Salesman Problem), enunciabile come "esiste un percorso di lunghezza inferiore a un certo valore?" è in **NP.** Infatti il seguente algoritmo non deterministico riceve in ingresso la lista D delle distanze, sceglie con una scelta "fortunata" il percorso richiesto e verifica che la sua lunghezza è minore del valore assegnato L:

```
Program TSP(D,L): begin
        Scegli nondeterm. un percorso P;
        if lunghezza (P) <L then return 'sì'
                                else return 'no'
end
```

[1] Un algoritmo non deterministico è una procedura che, posta di fronte a delle alternative, fa sempre "scelte fortunate". Si tratta chiaramente di un puro modello ideale, la cui importanza sta nel fatto che modellizza la "verifica" di un teorema, laddove un algoritmo deterministico ne costruisce la "dimostrazione". Presentiamo a titolo di esempio un algoritmo non deterministico per il problema "COMPOSTO":

```
Program COMPOSTO(N): begin
        scegli nondeterm. i, j ≤ ⌈sqrt(N)⌉
        if  ij =N then return 'p è composto'
                else return 'p è primo'
end
```

Poiché il programma è guidato da scelte fortunate, N è composto se e solo se i due numeri i e j, scelti nondeterministicamente, verificano la condizione ij=N.

[2] Questa pesante asimmetria tra fattorizzazione degli interi e verifica della correttezza del risultato è alla base della sicurezza di molti sistemi di crittografia a chiave pubblica, tra i quali l'RSA, il sistema più diffuso, il cui nome deriva da quello degli inventori Ron Rivest, Adi Shamir e Leonard Adleman (1979).

Il tempo richiesto per effettuare la verifica della lunghezza di P è proporzionale al numero *n* delle città.

Vi è comunque una differenza sostanziale tra i problemi COMPOSTO e TSP. A differenza del primo, TSP racchiude in qualche modo tutta la complessità della classe **NP**: è NP-completo. Un problema P si dice ***NP-completo*** quando:

1. è in **NP**, ovvero esiste un algoritmo polinomiale non deterministico per risolverlo;
2. dato un qualunque problema Q in **NP** esso è riducibile a P, ovvero esiste un algoritmo $A_Q$ di complessità polinomiale che fa corrispondere a istanze di Q istanze di P che hanno la stessa soluzione (sì o no).

Vi sono attualmente centinaia di problemi **NPC** (**NP**-completi) conosciuti. Essendo tutti i problemi **NP** riducibili ad uno qualunque di questi, trovare un algoritmo deterministico polinomiale per risolvere un problema **NP**-completo equivarrebbe a dimostrare l'eguaglianza **NP=P**. Tuttavia ogni sforzo in questo senso è a tutt'oggi stato vano (ricordiamo che il problema COMPOSTO, risolto dall'algoritmo quantistico di Shor, è NP ma *non* NP-completo). In sostanza, i problemi **NPC** sembrano sfuggire ad ogni possibilità di risoluzione efficiente a causa della loro natura puramente combinatoria che rende necessaria, almeno per quanto se ne sa oggi, l'analisi di tutte le possibili soluzioni. Dimostrare che un problema è **NPC** significa quindi porlo, molto probabilmente, ben al di là delle possibilità di risoluzione automatica, almeno per quanto riguarda gli algoritmi classici. Va comunque notato che **P≠NP** è solo una congettura, in quanto sono pure falliti, finora, tutti i tentativi di dimostrare una limitazione inferiore più che polinomiale per un qualsivoglia problema **NP**-completo [Leoncini].

# *Gli algoritmi probabilistici*

Un altro settore importante è quello degli algoritmi probabilistici. Accettando il risultato di un calcolo affetto da errore probabilistico o che può fornire una risposta inconcludente è spesso possibile sintetizzare algoritmi più efficienti rispetto a quelli deterministici noti.

Consideriamo un algoritmo $\mathcal{A}$ che fornisce un risultato esatto solo con probabilità $1-\varepsilon$, in modo tale che noi possiamo capire quando quel risultato è esatto (ad esempio, un algoritmo per la fattorizzazione potrebbe fornire un candidato fattore seguito da una divisione di prova). Qui $\varepsilon>0$ è indipendente dall'ingresso N. L'iterazione di $\mathcal{A}$ $k$ volte produce l'algoritmo $\mathcal{A}_k$, che avrà successo con probabilità $1-\varepsilon^\kappa$ che può essere resa arbitrariamente vicina ad uno scegliendo un $k$ fissato abbastanza grande. Inoltre se $\mathcal{A}$ è efficiente lo è anche $\mathcal{A}_k$, in quanto $k$ non dipende da N. In tale modo la probabilità di successo di qualunque algoritmo probabilistico efficiente può essere "amplificata" a piacimento preservando l'efficienza. In realtà anche nel caso che la probabilità di successo $1-\varepsilon$ decresca con N come 1/poly(log N) si può fare crescere $k$ come poly(log N) mantenendo l'efficienza e potendo pur sempre rendere la probabilità di successo arbitrariamente vicina ad uno. L'algoritmo quantistico di Shor per fattorizzare gli interi è in effetti di questo tipo, basato su un algoritmo efficiente che fornisce un fattore dell'ingresso N con probabilità che decresce come 1/poly(log N) [Ekert].

Per i problemi decisionali esiste una classe di algoritmi più fondamentalmente probabilistici, noti come algoritmi BPP [Papadimitriou]. Un algoritmo BPP $\mathcal{A}$ è una procedura efficiente che per ogni ingresso fissato fornisce una risposta corretta con probabilità maggiore di una costante $\delta>1/2$. In questo caso, tuttavia, non siamo in grado di sapere se la risposta è corretta o meno. Come sopra possiamo ripetere $\mathcal{A}$ un numero fissato $k$ di volte ed accettare per

buono il "voto a maggioranza" delle $k$ risposte. Per $k$ sufficientemente grande la probabilità che la risposta vincente sia corretta è arbitrariamente vicina ad uno (in effetti tale probabilità converge ad uno esponenzialmente con $k$). In teoria della complessità si considerano usualmente trattabili anche i problemi per i quali esiste un algoritmo BPP [1] .

Esistono diversi algoritmi probabilistici che risolvono il problema COMPOSTO. L'algoritmo di Rabin [Rabin], ad esempio, accetta in ingresso un intero N. Se al termine del calcolo risulta che N è composto, allora lo è realmente. Se invece risulta che N è primo, allora esso è primo con probabilità $1-0.25$ che non dipende da N. Essendo la procedura efficiente, possiamo ripeterla più volte per ottenere il grado di confidenza desiderato. Questo algoritmo non è comunque in grado di esibire un fattore del numero N [Ekert].

Non esiste attualmente alcun algoritmo efficiente, neppure probabilistico, per la fattorizzazione. I migliori algoritmi classici attualmente disponibili sono: il *Multiple Polynomial Quadratic Sieve* [Silverman] per numeri lunghi meno di 120 cifre decimali, ed il *Number Field Sieve* [Lenstra] particolarmente conveniente per numeri superiori a 110 cifre decimali. Tuttavia anche gli algoritmi più veloci richiedono tempi dell'ordine di $\exp((\log N)^{1/3}(\log\log N)^{2/3})$ e impiegherebbero due miliardi di anni per fattorizzare un numero di duecento cifre.

## *La macchina di Turing classica*

Una macchina di Turing $\mathcal{M}$ è un dispositivo *ideale* costituito da una testina di lettura/scrittura e da un nastro infinito, che funge da memoria. Il nastro è suddiviso in celle discrete ed ogni cella può contenere un simbolo di un alfabeto

[1] Anche l'approccio probabilistico ha dei limiti. In particolare non è noto alcun algoritmo probabilistico efficiente per problemi NP-completi, e si ritiene fondamentalmente che un tale algoritmo non possa esistere.

finito **Σ**. In questo alfabeto distinguiamo convenzionalmente un *carattere spazio*, che indica che la cella corrispondente non contiene alcuna informazione. Il tempo è discreto e la testina ha accesso, ad ogni passo della computazione, ad una e una sola cella del nastro[1] . Essa può inoltre trovarsi in un numero finito di stati interni, che indicheremo con **Q**=$\{q_0, q_1, ... q_h\}$. Ad ogni passo la sua azione è determinata interamente dal suo stato interno e dal simbolo presente sulla cella del nastro alla quale ha accesso in quel momento. Sulla base di questi due dati essa scrive un nuovo simbolo sul nastro, commuta il suo stato interno e si sposta di una locazione a destra o a sinistra. Questo comportamento può essere interamente codificato da un insieme *finito* (in quanto **Σ** e **Q** sono insiemi finiti) di quintuple della forma

(q, s, q', s', d).

I primi due caratteri descrivono lo stato della testina e il simbolo letto, mentre i rimanenti tre indicano lo stato in cui commutare (eventualmente uguale al precedente), il carattere da scrivere nella cella (anche s'=s) e la direzione in cui spostarsi (destra o sinistra). E' usuale distinguere, fra gli stati interni, uno *stato di arresto*, che denotiamo qui con $q_h$. Tale stato, una volta raggiunto, causa l'arresto della macchina e il termine della computazione. Esso compare perciò solo in quintuple del tipo ($q_h$, s, $q_h$, s, "stop"), che non comportano alcun ulteriore movimento della testina [Ekert].

All'inizio della computazione il nastro viene preparato con un numero *finito* di caratteri diversi dal carattere spazio, posti ad esempio a destra della testina e rappresentanti gli ingressi. Un computazione è *valida* (ovvero costituisce effettivamente una computazione) se la macchina raggiunge lo stato di arresto

[1] Nella versione originale di Turing, la testina è fissa ed è in grado di spostare il nastro durante la computazione. I fisici sembrano ritenere che una macchina *ideale* non abbia la potenza necessaria a muovere un nastro *infinito*, e preferiscono perciò fare muovere la testina. A mio avviso la questione è del tutto irrilevante. Siccome però questa seconda descrizione ha il pregio di rendere più semplice simulare i programmi con carta e penna, la adotterò senz'altro in quanto segue.

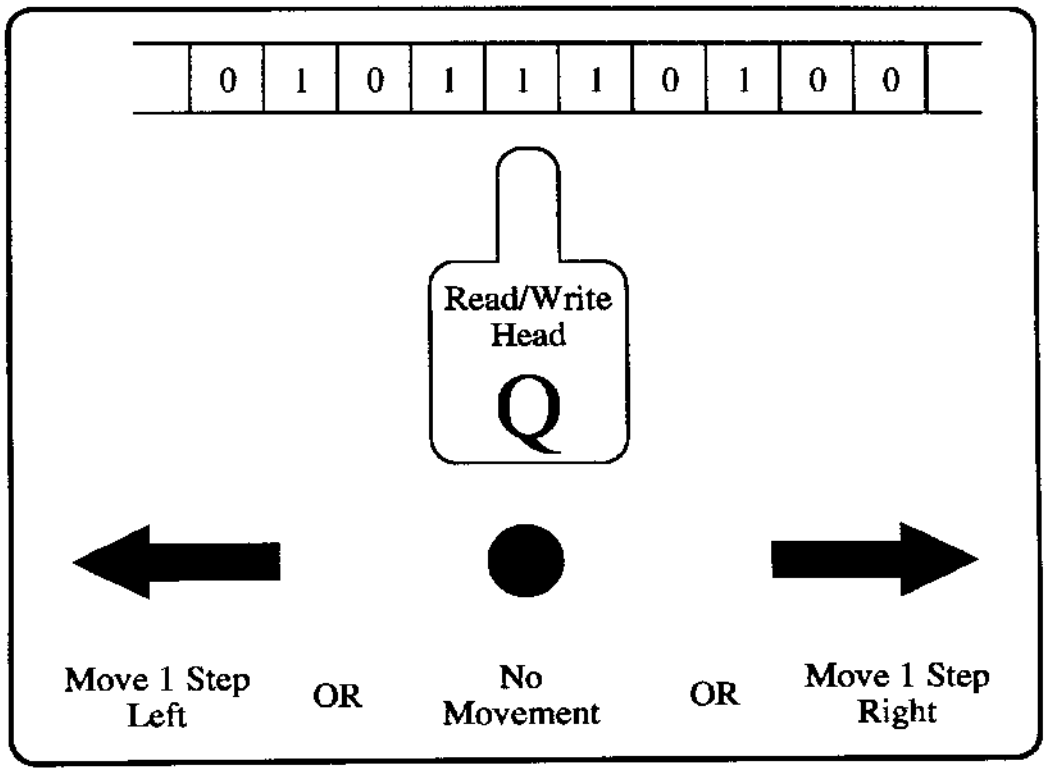


*Figura 1: Rappresentazione schematica di una Macchina di Turing.*

dopo un numero finito di passi. A quel punto, è possibile leggere il risultato in una posizione convenzionale rispetto alla testina. In ogni momento, perciò, soltanto un numero finito di celle del nastro contiene un carattere diverso dal carattere spazio, ovvero ogni computazione utilizza solo una quantità finita di memoria.

L'alfabeto $\Sigma$ può essere ridotto, senza alcuna perdita di generalità, all'insieme dei due simboli {0, 1}, dove lo 0 svolge anche la funzione di carattere spazio; è inoltre possibile identificare gli stati interni di una qualunque macchina con numeri binari e tradurre le regole di transizione in stringhe di simboli "0" e "1". In questo modo si può codificare *completamente* la descrizione di una macchina di Turing in un unico intero, il suo *numero di Gödel* (per una descrizione più dettagliata si veda [Penrose]). Questo dimostra, tra l'altro, che le macchine di Turing sono numerabili e che quindi le funzioni *ricorsive* (quelle da esse computabili) sono infinitamente meno di tutte le applicazioni di **N** in **N**.

Il significato profondo di questa rappresentazione si comprende alla luce dell'esistenza della ***macchina universale di Turing*** $\mathcal{T}$, la quale è in grado di emulare una qualunque MdT nel senso seguente: se m è il numero di Gödel della macchina $T_m$ allora per ogni ingresso $x$ di $T_m$ si ha $\mathcal{T}(m,x)= T_m(x)$. In altri termini $\mathcal{T}$ riceve in ingresso m, ovvero la descrizione di $T_m$ (assimilabile ad un programma), seguito dal valore di $x$ ed utilizza il nastro infinito per riprodurre gli stati interni di $T_m$ e simularne l'evoluzione. L'esistenza della macchina universale di Turing è dimostrabile costruttivamente[1] . Essendo a tutti gli effetti una MdT, essa stessa è identificata dal suo numero di Gödel. La macchina universale non è unica; l'ordine di grandezza dell'intero che la descrive è comunque quello di una quindicina di linee a stampa in base dieci, equivalenti ad un paio di pagine in notazione binaria (per il numero di Gödel di una macchina universale ed alcuni simpatici esercizi di programmazione in "linguaggio macchina" si veda [Penrose]).

## *Reversibilità della computazione*

Si è a lungo ritenuto che la computazione comportasse una duplice irreversibilità, sia dal punto di vista della teoria dell'informazione, sia dal punto di vista termodinamico. E' evidente che l'espressione 2+2 contiene più informazione del numero 4, che non codifica in alcun modo la memoria degli addendi. Analogamente, le porte logiche tradizionalmente usate distruggono informazione: è impossibile risalire dall'uscita di un AND al valore logico dei due ingressi. Questo è vero in generale per ogni porta logica che abbia più linee di ingresso che uscite. Dal punto di vista fisico la perdita di informazione comporta la dissipazione di energia nel dispositivo che effettua materialmente il calcolo, in

[1] L'esistenza della macchina universale di Turing rappresenta inoltre una conferma della bontà del modello di computazione. Calcolare l'azione della macchina $T_m$ sull'ingresso $x$ costituisce infatti sicuramente una procedura meccanica (algoritmica), e come tale ci aspettiamo che sia eseguibile da una qualche macchina di Turing.

quanto un sistema non dissipativo è termodinamicamente reversibile e mantiene perciò inalterato il suo contenuto di informazione.

Tuttavia, come indicato da Bennett nel 1973, la perdita di informazione non è affatto essenziale alla computazione ([Bennett2], [Bennett3]). Da allora questa conclusione è stata dimostrata dalla nascita di modelli reversibili di computazione, incluse le porte logiche reversibili (ad es. le porte di Fredkin). Bennett ha inoltre dimostrato la realizzabilità, almeno teorica, di una macchina di Turing reversibile "ad orologeria" costituita di parti rigide e prive di attrito, che sfrutta per effettuare le transizioni l'energia di agitazione termica dei suoi componenti. Tale macchina può effettuare, ad ogni istante, soltanto due movimenti, corrispondenti ad un passo avanti o ad un passo indietro nel calcolo. Essendo azionata dal moto browniano, essa ha la stessa probabilità di procedere come di tornare indietro lungo il percorso del calcolo; tuttavia, una forza esterna piccola a piacere è sufficiente ad orientare la computazione nella direzione voluta [Bennett4]. Questo modello dimostra che non esiste, in linea di principio, una quantità minima di energia necessaria ad effettuare una computazione.

La macchina di Turing è di per sé irreversibile, non solo in quanto la sua inversione temporale implica operazioni (come scrivere un carattere in una cella prima di leggerne il contenuto) non previste dal suo funzionamento, ma più fondamentalmente perché una configurazione non identifica, in genere, univocamente il suo predecessore. Bennett ha tuttavia dimostrato che data una MdT $\mathcal{M}$ esiste una macchina di Turing reversibile $\mathcal{M}'$ che effettua la stessa computazione. Inoltre $\mathcal{M}'$ è efficiente se $\mathcal{M}$ è efficiente, quindi richiedere che la computazione sia reversibile non impone alcuna restrizione alle possibilità di calcolo. Questa osservazione è importante, in quanto i modelli di computazione quantistica sotto esposti sono intrinsecamente reversibili.

Notiamo infine che la reversibilità impone di riconsiderare il concetto di stato di arresto, in quanto una macchina reversibile non può trovarsi in due istanti

successivi nella stessa configurazione. Una macchina di Turing reversibile potrebbe informarci del termine della computazione ad esempio scrivendo un 1 in una posizione prefissata del nastro e continuare poi ad evolvere in modo arbitrario senza alterare la zona contenente il risultato.

## *Algoritmi probabilistici sulla MdT*

Possiamo effettuare computazioni non deterministiche con la macchina di Turing mettendola in grado di compiere, ad ogni passo, scelte casuali. Questo genere di computazione può essere visualizzato come un grafo orientato ad albero ai vertici del quale corrispondono le possibili configurazioni della macchina (cioè valori completamente specificati di posizione, stato della testina e contenuti del nastro) e i cui rami descrivono le transizioni (Figura 2). La computazione ha inizio dal nodo radice, corrispondente alla configurazione iniziale, e gli altri vertici corrispondono a configurazioni raggiungibili con probabilità non nulla a partire dalla radice.

Il funzionamento della macchina è riassunto da una descrizione finita della forma:

$$\delta : Q \times \Sigma \times Q \times \Sigma \times \{Destra, Sinistra\} \rightarrow [0,1]$$

dove $\delta(q,s,q',s',d)$ è la probabilità che la testina, quando si trova nello stato q e legge il simbolo s, scriva s' e si muova in direzione d. Questa descrizione deve soddisfare gli assiomi della teoria della probabilità. Se associamo ad ogni ramo dell'albero la probabilità che la computazione segua tale ramo dobbiamo richiedere che la somma delle probabilità dei rami uscenti da un vertice sia sempre uguale ad uno. La probabilità di un particolare percorso dalla radice ad un nodo fissato è pari al prodotto delle probabilità associate ai rami percorsi. La probabilità che una particolare configurazione venga raggiunta in *n* passi è data

dalla somma delle probabilità di tutti i percorsi che in *n* passi raggiungono quel nodo dalla configurazione iniziale [Ekert].

Questo modello di calcolo probabilistico si presta ad una naturale generalizzazione quantistica, descritta nel seguente capitolo.

# La macchina di Turing quantistica

## *Dalle probabilità alle ampiezze di probabilità*

La generalizzazione quantistica della macchina di Turing probabilistica descritta nel precedente paragrafo è immediata. Facciamo corrispondere i vari stati interni della testina e i simboli del nastro a stati ortogonali in appropriati

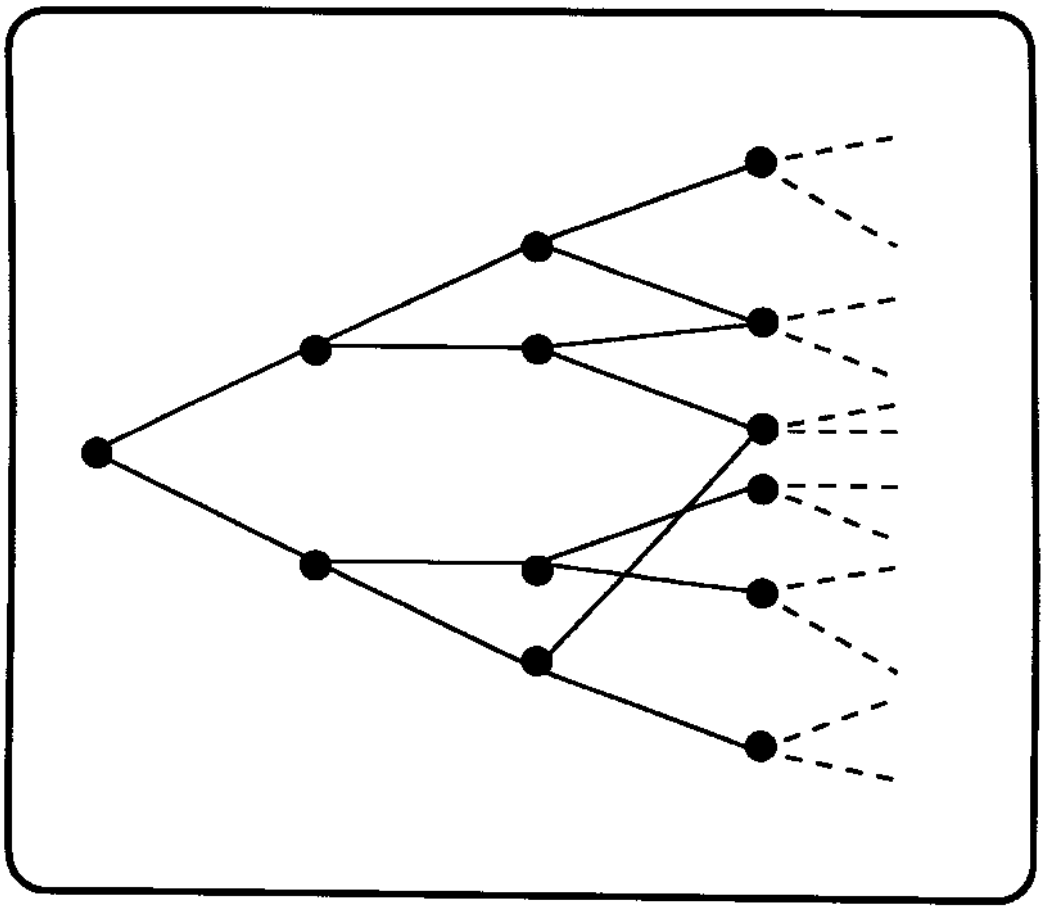

*Figura 2: Albero computazionale di una MdT probabilistica*

spazi di Hilbert e codifichiamo la posizione della testina con una osservabile con spettro intero. La computazione quantistica può essere descritta da un grafo simile a quello di una computazione probabilistica (Figura 2). In questo caso, associamo ad ogni ramo l'*ampiezza di probabilità* che la computazione lo segua. Come sopra, l'ampiezza di probabilità di un particolare percorso è il prodotto delle ampiezze dei suoi rami e l'ampiezza di probabilità di una particolare configurazione è la somma delle ampiezze di tutti i percorsi che la uniscono alla radice dell'albero. Le varie probabilità sono ovviamente il quadrato del modulo delle rispettive ampiezze.

Questa sostituzione introduce immediatamente un fenomeno tipicamente quantistico, ovvero l'***interferenza*** dei percorsi computazionali. Per esempio, se una particolare configurazione finale può essere raggiunta lungo due differenti percorsi con ampiezze $\alpha$ e $-\alpha$ allora la probabilità di raggiungerla è $|\alpha-\alpha|^2=0$, nonostante il fatto che la probabilità di seguire uno dei due percorsi separatamente sia $|\alpha|^2\neq 0$. La computazione può quindi procedere lungo la sovrapposizione di più percorsi computazionali diversi e fornire un risultato finale dipendente dalla loro interferenza[1]. Questa caratteristica, denominata ***"parallelismo quantistico",*** è del tutto estranea al funzionamento di una MdT probabilistica, che segue *un solo* percorso fra i possibili, ed è alla base di tutti gli algoritmi quantistici conosciuti.

L'azione di una macchina quantistica di questo genere è completamente e finitamente specificata da una funzione

$$\delta : Q \times \Sigma \times Q \times \Sigma \times \{Destra, Sinistra\} \to \mathbf{C}$$

[1] Potendo la macchina evolvere lungo una sovrapposizione di percorsi, il concetto di posizione della testina o del nastro viene fastidiosamente ad assomigliare a quello di posizione di un elettrone. Per questo motivo anche le considerazioni di Deutsch sui requisiti di località nello spostamento del nastro infinito mi sembrano poco significative.

dove $\delta(q, s, q', s', d)$ è l'ampiezza di probabilità che la macchina, trovandosi nello stato q e leggendo il simbolo s, cambi il suo stato in q', scriva s' e si muova nella direzione d. Tali ampiezze sono vincolate dal requisito di *unitarietà* della trasformazione da esse indotta.

La macchina di Turing quantistica (QTM), introdotta da Deutsch [Deutsch2] nel 1985, è il modello più studiato di computazione quantistica. In esso ogni singolo passo della computazione è il risultato di una trasformazione unitaria fissata indotta da $\delta$, che agisce sulla configurazione globale dell'intera macchina. Su questa trasformazione unitaria si pone un requisito di *località* permettendo transizioni solo fra configurazioni "vicine", nelle quali ad esempio la posizione del nastro cambia di una sola unità e l'unico simbolo a venire alterato è quello sotto la testina[1] . Sono comunque possibili altri modelli di computazione. Si può ad esempio ipotizzare una hamiltoniana locale nel senso descritto e fare evolvere il sistema per un certo tempo per generare la computazione [Feynman]. Sono stati anche considerati circuiti e automi cellulari quantistici; tuttavia, le relazioni esistenti tra le potenzialità di tutti questi modelli non sono ancora state chiarite. La QTM presenta in ogni caso, come vedremo, una notevole generalità.

Una QTM può essere simulata con precisione arbitraria da una macchina di Turing classica. Noti l'ingresso e le regole di evoluzione è infatti possibile programmare un calcolatore classico per iterare queste ultime e calcolare ad ogni passo le ampiezze delle configurazioni accessibili (che sono un numero finito); la distribuzione finale di probabilità può infine essere classicamente simulata "lanciando delle monetine". Tuttavia questa simulazione classica della QTM è *inefficiente*, in quanto il numero di ampiezze coinvolte cresce, in genere, esponenzialmente col numero di passi della computazione.

[1] Occorre in realtà tener conto del principio di sovrapposizione. Si veda più avanti per una enunciazione formale dei vincoli sulla trasformazione.

R. P. Feynman [Feynman2] ha osservato che *è in generale impossibile simulare l'evoluzione di un sistema quantistico su un calcolatore classico in maniera efficiente.* Questo fatto denota una profonda differenza tra la fisica classica e la fisica quantistica e costituisce un criterio di raffronto inedito tra le due teorie. Nel caso di una QTM, ad esempio, il numero di celle del nastro (sistemi quantistici a due stati) coinvolte in una computazione cresce in generale linearmente col numero $n$ di passi eseguiti, per cui la dimensione dello spazio di Hilbert interessato cresce esponenzialmente con $n$. Perciò la "naturale" simulazione classica - ovvero il calcolo dell'evoluzione di tutte le ampiezze secondo le regole di transizione della QTM - comporta un rallentamento esponenziale. Notiamo che una simile esplosione si verifica anche nel caso di una macchina di Turing probabilistica classica: il numero di vertici esplorati nel grafo computazionale cresce anch'esso esponenzialmente con $n$. Anche in questo caso, perciò, calcolare ad ogni istante l'intera distribuzione di probabilità comporta un rallentamento esponenziale. Possiamo però simulare la distribuzione di probabilità semplicemente "lanciando le monetine" ed effettuando le scelte probabilistiche man mano che si presentano, ovvero attraversare un singolo percorso computazionale rispettando ad ogni passo la corretta distribuzione di probabilità. Al contrario, nel caso della QTM l'*interferenza* dei diversi percorsi computazionali ci impedisce di seguirne uno solo preservando la corretta distribuzione finale di probabilità[1] . Evidentemente la Natura è in grado di tenere

[1] Questo fatto ha un profondo significato fisico, che affonda le sue radici ben oltre i limiti della teoria della computazione.

La macchina di Turing probabilistica descritta a pagina 17 può naturalmente essere associata, mediante numerazione delle configurazioni, ad un processo stocastico markoviano a tempo e stati discreti la cui matrice di transizione è direttamente determinata dalle regole di evoluzione della macchina stessa (le funzioni $\delta$). Tale descrizione è applicabile ad un qualunque automa classico. In particolare, le computazioni deterministiche corrispondono a catene di Markov banali la cui matrice di transizione contiene, in ogni riga, uno ed un solo "1".

Nel suo articolo del 1982, Feynman ha osservato che il tentativo di esprimere le equazioni di un generico automa quantistico come processo markoviano conduce a delle "probabilità" di transizione negative. Questa non è altro che una manifestazione dell'impossibilità di riprodurre le correlazioni quantistiche con una teoria a variabili nascoste.

efficientemente traccia del moltiplicarsi esponenziale delle ampiezze in un modo che non ci è possibile simulare classicamente. Questo è precisamente il fenomeno del "parallelismo quantistico" menzionato in precedenza [Ekert].

Possiamo quindi sospettare che le potenzialità computazionali di un dispositivo quantistico, in contrasto con l'enunciazione della tesi di Church (vedi pagina 8), superino quelle di qualsiasi calcolatore classico almeno riguardo all'efficienza della computazione.

## *Aspetti formali*

Presentiamo di seguito una formalizzazione della macchina di Turing quantistica, sulla quale sarà poi possibile fondare la *teoria quantistica della computazione*.

Analogamente ad una MdT classica, una QTM consiste in una testina di lettura-scrittura dotata di un numero finito di stati e in una memoria a nastro infinita, della quale viene sempre usata solo una parte finita. Durante un singolo passo della computazione il processore interagisce in genere, grazie alla sovrapposizione degli stati, con un numero *finito* di locazioni di memoria, mentre la rimanente parte non viene coinvolta.

Descriviamo gli stati interni della testina mediante N osservabili a due stati:

$$\{\mathbf{S}_i\} \qquad (i \in \mathbf{Z}_N)$$

dove $\mathbf{Z}_N$ è l'insieme degli interi da 0 a N-1. Il numero di osservabili necessarie a codificare un processore con P stati interni è il primo intero maggiore o uguale del logaritmo in base due di P.

---

In conclusione, *il vantaggio del parallelismo quantistico sul calcolo sequenziale classico* (vedi oltre nel testo) *dipende dalla non esistenza di teorie a variabili nascoste*. Se una tale teoria esistesse, anche il calcolo quantistico sarebbe riducibile ad un processo markoviano.

Analogamente, la memoria è rappresentata dalla successione infinita

$$\{\mathbf{M}_i\} \qquad (i \in \mathbf{Z})$$

di osservabili a due stati. Indicheremo collettivamente le $\{\mathbf{S}_i\}$ con **S** e le $\{\mathbf{M}_i\}$ con **M**. La "posizione del nastro" corrisponde all'ulteriore osservabile **X**, che ha come spettro gli interi **Z**.

Richiediamo inoltre che tutte le osservabili sopra introdotte siano compatibili, in modo che una configurazione della macchina corrisponda ad un raggio nello spazio di Hilbert $\mathcal{H}$ generato dagli autovettori simultanei

$$\left| x; \mathbf{s}; \mathbf{m} \right\rangle \equiv \left| x;\ s_0, s_1, \ldots, s_{N-1}; \ldots, m_{-1}, m_0, m_1, \ldots \right\rangle$$

delle osservabili **X**, **S** ed **M**. Indicheremo la base formata da questi autostati col nome di *base computazionale*. E' infine conveniente assumere che lo spettro delle osservabili a due stati sia $\{0,1\}$ anziché il più tradizionale $\{-½, ½\}$, in modo da permetterne una immediata interpretazione come bit quantistici.

La dinamica della QTM è descritta da un *operatore unitario costante* **U** su $\mathcal{H}$. **U** specifica l'evoluzione della generica configurazione $\left|\psi(t)\right\rangle \in \mathcal{H}$ ad ogni passo della computazione, che si assume richiedere un tempo T fissato indipendente dalla configurazione:

$$\left|\psi(n\,\mathrm{T})\right\rangle = \mathbf{U}^n \left|\psi(0)\right\rangle \qquad (n \in \mathbf{Z}^+)$$

L'analisi delle proprietà computazionali della QTM non richiede la conoscenza dello stato del sistema a tempi diversi da multipli interi positivi di T.

La computazione ha inizio al tempo t=0. In tale istante il processore si deve trovare in uno stato standard, ad esempio **S=0**, e in una posizione fissata del nastro, usualmente **X**=0. Un numero finito delle $\mathbf{M}_i$ codificherà inoltre l'*ingresso*[1] della QTM, mentre le rimanenti locazioni saranno azzerate. Il più

[1] Analogamente a quanto fatto per la macchina di Turing universale classica (vedi pagina 15), una parte dell'ingresso potrà eventualmente essere interpretata come *programma* della QTM.

generico stato iniziale, tenendo conto della possibilità di sovrapporre più ingressi, sarà quindi descritto dalle equazioni

$$\left|\psi(0)\right\rangle = \sum_{\mathbf{m}} \lambda_{\mathbf{m}} \left| 0; \mathbf{0}; \mathbf{m} \right\rangle$$

$$\sum_{\mathbf{m}} \left|\lambda_{\mathbf{m}}\right|^2 = 1$$

nelle quali solo un numero finito di $\lambda_{\mathbf{m}}$ è diverso da zero e $\lambda_{\mathbf{m}}$ si annulla ogniqualvolta il vettore **m** ha infinite componenti non nulle.

I requisiti di località dell'interazione impongono che gli elementi di matrice di **U** abbiano la forma seguente:

$$\left\langle x'; \mathbf{s}'; \mathbf{m}' \middle| \mathbf{U} \middle| x; \mathbf{s}; \mathbf{m} \right\rangle = \left[ \delta_{x'}^{x+1} \mathrm{U}^{+}(\mathbf{s}'; m'_x | \mathbf{s}; m_x) + \delta_{x'}^{x-1} \mathrm{U}^{-}(\mathbf{s}'; m'_x | \mathbf{s}; m_x) \right] \prod_{y \neq x} \delta_{m_y}^{m'_y}$$

La produttoria infinita sulla destra assicura che solo il bit *x*-esimo del nastro, sul quale la testina si trova posizionata, possa essere alterato. Le $\delta_{x'}^{x\pm1}$ permettono alla posizione della testina di variare, durante un singolo passo della computazione, di una e una sola unità in avanti o in indietro (o in una sovrapposizione delle due direzioni). Il comportamento della specifica QTM è interamente codificato dalle funzioni $\mathrm{U}^{\pm}(\mathbf{s}'; m'_x | \mathbf{s}; m_x)$, soggette al solo vincolo dell'unitarietà di **U**.

Come abbiamo visto (pagina 13) le macchine di Turing classiche si arrestano al termine di una computazione entrando in uno stato speciale, denominato stato di arresto, che poi mantengono indefinitamente. La reversibilità di una QTM impone però che essa non possa assumere, al termine di una computazione non banale, due stati consecutivi uguali. E' perciò necessario riservare ad una delle osservabili che descrivono lo stato del processore, convenzionalmente $\mathbf{S}_0$, la funzione di "flag di arresto", ponendo al tempo t=0 $s_0$=0 e definendo "computazioni valide" quelle al termine delle quali la macchina commuta il flag e continua poi ad evolvere in modo arbitrario senza alterare i risultati. Una misura periodica del flag di arresto ci informa così del

completamento della computazione. Questo accorgimento, al quale abbiamo fatto accenno nella trattazione delle MdT classiche reversibili ideate da Bennett (vedi pagina 16), va qui raffinato per evitare che l'osservazione prematura del flag di arresto faccia colassare lo stato della macchina su un autostato di $\mathbf{S}_0$, distruggendo quindi la coerenza della computazione. Questo può essere evitato richiedendo che la QTM interagisca col flag solo al termine del calcolo[1]. Nel seguito diremo comunque per semplicità che la macchina "si arresta" al termine di una computazione, sottintendendo le precedenti considerazioni.

L'insieme delle macchine di Turing classiche reversibili costituisce un sottoinsieme proprio della classe delle QTM caratterizzato da funzioni $U^{\pm}$ della forma

$$U^{\pm}(\mathbf{s}';m_x'|\mathbf{s};m_x)=\tfrac{1}{2}\delta_{\mathbf{s}'}^{\mathbf{A}(\mathbf{s};m_x)}\delta_{m_x'}^{\mathrm{B}(\mathbf{s};m_x)}\left[1\pm \mathrm{C}(\mathbf{s};m_x)\right].$$

**A** descrive lo stato in cui il processore deve portarsi quando si trova nello stato **s** e legge il bit $m_x$; il suo codominio è l'insieme $(\mathbf{Z}_2)^N$ degli stati della testina. B è una applicazione nell'insieme $\mathbf{Z}_2$ dei valori di verità di un bit e specifica il nuovo contenuto della cella corrente del nastro. C ha codominio $\{-1, 1\}$ e determina la direzione di spostamento della testina, che deve essere o destra o sinistra (classicamente le sovrapposizioni non hanno senso).

In altri termini *le MdT classiche sono quei computer quantistici la cui dinamica assicura che, se preparati in uno stato della base computazionale, rimangono al termine di ogni passo in uno stato di tale base*. Per garantire l'unitarietà è in tal caso necessario e sufficiente che la corrispondenza

[1] Non riesco a capire perché D. Deutsch proponga di considerare valide le computazioni per le quali l'*aspettazione* del tempo di esecuzione è finito. Questo ha sicuramente senso nel caso di MdT classiche probabilistiche; l'evoluzione della QTM è però completamente deterministica, almeno fino a quando il suo stato non viene misurato. L'unico modo per ottenere una durata variabile della computazione sarebbe perciò permettere al flag di arresto di trovarsi in una sovrapposizione di $|0\rangle$ ed $|1\rangle$; in tal caso, però, la computazione avrebbe probabilità non nulla di continuare anche dopo la prima interazione del calcolatore col flag di arresto, cosa che abbiamo visto non essere ammissibile.

$$(\mathbf{s}, m) \leftrightarrow (\mathbf{A}(\mathbf{s}, m), \mathrm{B}(\mathbf{s}, m), \mathrm{C}(\mathbf{s}, m))$$

sia bigettiva.

Esisterà in particolare una scelta delle funzioni **A**, B, C per cui la QTM si comporta come una macchina universale di Turing classica reversibile. Nel seguito indicheremo tale QTM con $\mathcal{T}$.

## *Verso una descrizione "ad alto livello"*

Il formalismo sopra sviluppato è in pratica troppo pesante non solo per permettere lo studio di algoritmi di utilità pratica, ma anche per consentire una indagine più approfondita delle potenzialità del modello. Per queste ragioni è opportuno, come nel caso classico, ricorrere ad un livello di descrizione più astratto.

La macchina di Turing si presenta di per sé come un dispositivo per l'elaborazione di informazione nel senso più ampio, senza alcun riferimento a enti matematici definiti. Tuttavia, il calcolo di funzioni di **N** in **N** cattura tutta la complessità dei problemi computazionali in una forma più direttamente trattabile. E' facile convincersi che la MdT, e quindi la QTM, può operare sui numeri naturali. La più semplice codifica possibile dei naturali è quella in base uno, che assegna ad esempio al numero 3 la rappresentazione ...◊◊◊↑↑↑↑◊◊◊... sul nastro[1] . Codifiche leggermente più elaborate [Penrose] permettono ad esempio la rappresentazione in base due dei naturali e la codifica di un alfabeto finito di simboli come quello originariamente utilizzato da Turing. Nel seguito supporremo perciò che il nastro della QTM venga allocato come una sequenza di locazioni di memoria [Deutsch] ciascuna delle quali è in grado di contenere un numero intero

[1] L'uso delle losanghe e delle frecce al posto dei simboli '0' ed '1' serve a ricordare che i simboli sul nastro costituiscono di per sé dei puri segni grafici e non hanno a priori alcuna relazione con i numeri zero ed uno. E' inoltre opportuno rappresentare il numero *n* con *n*+1 frecce, per evitare che lo zero sia indistinguibile dal nastro vuoto.

arbitrario. In particolare, essendo il programma di una macchina universale descrivibile da un numero intero (vedi pagina 15), esso potrà venire memorizzato in una di tali locazioni.

Se f è una funzione computabile esiste per definizione un programma $\pi$ di $\mathcal{T}$ che la calcola, ovvero esiste un intero $n$ per cui

$$\mathbf{U}^n \left| x=0; \mathbf{s}=\mathbf{0}; \mathbf{m}=\pi(\mathrm{f}), i, \mathbf{0} \right\rangle = \left| x=0; \mathbf{s}=1, \mathbf{0}; \mathbf{m}=\pi(\mathrm{f}), i, \mathrm{f}(i), \mathbf{0} \right\rangle$$

dove $\mathbf{U}$ descrive l'evoluzione di $\mathcal{T}$, $\mathbf{0}$ rappresenta una sequenza infinita di zeri e gli autovalori di $\mathbf{M}_i$ per $i<0$ non sono indicati esplicitamente; il primo bit del vettore di stato $\mathbf{s}$ rappresenta il flag di arresto, che vale uno al termine della computazione. La copia di $i$ nello stato finale ha lo scopo di assicurare la reversibilità della computazione anche nel caso che la funzione f sia non iniettiva. Tuttavia l'esistenza, per ogni funzione ricorsiva[1] f, di un calcolatore reversibile capace di computarla producendo come risultato esclusivamente il valore richiesto f($i$) e l'ingresso, senza ulteriore output spurio, non è a priori garantita e segue da uno specifico teorema ([Bennett3], [Feynman]). Nel caso particolare che la funzione f sia iniettiva si dimostra inoltre l'esistenza di un calcolatore reversibile che la computa senza mantenere nell'output una copia dell'ingresso [Bennett3].

Più in generale, se f è una funzione computabile ed $a$ e $b$ sono due locazioni di memoria, indicate ad esempio da due interi, esiste un programma $\pi$(f,$a$,$b$) che calcola f sul contenuto di $a$ e scrive il risultato in $b$, lasciando tutto il resto inalterato[2]. Alleggerendo ulteriormente la notazione, indicheremo questa computazione come segue:

$$\left| \pi(\mathrm{f},2,3), i, j \right\rangle \rightarrow \left| \pi(\mathrm{f},2,3), i, j \oplus \mathrm{f}(i) \right\rangle$$

[1] "Funzione ricorsiva" è, nella teoria classica, un sinonimo frequentemente usato di "funzione computabile".

[2] Questi programmi utilizzeranno ovviamente una parte del nastro come area di lavoro, ripulendola poi prima di terminare.

All'interno dei ket compare la descrizione delle prime tre locazioni del nastro; il programma $\pi$, memorizzato nella prima, opera sui contenuti della seconda e della terza. Nel caso più generale, se *b* non contiene uno zero, una scrittura diretta su *b* rappresenta una operazione irreversibile che come tale non può essere compiuta. Si rende quindi necessario combinare reversibilmente il risultato cercato col contenuto precedente di *b* mediante un operatore logico $\oplus$ che richiediamo sia associativo, commutativo e goda delle seguenti proprietà:

$$\begin{cases} i \oplus i = 0 \\ i \oplus 0 = i \end{cases}$$

Tipicamente si usa per $\oplus$ la funzione XOR, che nel caso di numeri naturali si intende applicata ai bit corrispondenti nelle rappresentazioni in base due degli operandi[1] .

## *Il calcolatore universale quantistico*

Abbiamo finora fatto riferimento alla QTM $\mathcal{T}$, che ha tutte le caratteristiche della macchina universale di Turing; la famiglia delle QTM include però una classe più vasta di dispositivi in grado di fare evolvere gli stati puri della base computazionale in una loro combinazione lineare secondo una qualsiasi delle trasformazioni unitarie **U** indicate a pagina 24. Per fondare su tali trasformazioni un modello di computazione non è tuttavia sufficiente supporne l'esistenza: occorre anche indicare un modo di realizzarle con mezzi finiti, cioè di "programmarle". Trattandosi di un insieme continuo, saremo in generale in grado di programmarle soltanto entro una certa approssimazione, arbitrariamente migliorabile con l'uso di programmi via via più lunghi.

[1] XOR è di per sé una funzione logica, e non un operatore su di uno spazio di Hilbert. Tuttavia la sua tavola di verità ne specifica l'azione sui quattro stati di una base hilbertiana dello spazio dei due bit operandi, determinandone così univocamente l'estensione ad operatore autoaggiunto.

Consideriamo a tale scopo la QTM $\mathcal{Q}$ in grado di compiere, oltre alle normali operazioni di $\mathcal{T}$, le seguenti trasformazioni unitarie sullo spazio degli stati di un singolo bit:

$$\mathbf{V}_0 = \begin{pmatrix} \cos\alpha & \operatorname{sen}\alpha \\ -\operatorname{sen}\alpha & \cos\alpha \end{pmatrix} \quad \mathbf{V}_1 = \begin{pmatrix} \cos\alpha & i\operatorname{sen}\alpha \\ i\operatorname{sen}\alpha & \cos\alpha \end{pmatrix}$$

$$\mathbf{V}_2 = \begin{pmatrix} e^{i\alpha} & 0 \\ 0 & 1 \end{pmatrix} \qquad \mathbf{V}_3 = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$$

con le loro inverse $\mathbf{V}_4$, $\mathbf{V}_5$, $\mathbf{V}_6$, $\mathbf{V}_7$. La costante $\alpha$ è un multiplo irrazionale fissato di $\pi$. Ad ognuna delle $\mathbf{V}_i$ corrispondono elementi della base computazionale che rappresentano il programma $\varphi(\mathbf{V}_i, a)$ che esegue $\mathbf{V}_i$ sul bit meno significativo della locazione $a$. Così se $j$ vale zero o uno questi stati di base evolvono in una loro sovrapposizione lineare come segue:

$$\left|\varphi\left(\mathbf{V}_i,2\right), j\right\rangle \rightarrow \sum_{k=0}^{1} \left\langle k\middle|\mathbf{V}_i\middle| j\right\rangle \left|\varphi\left(\mathbf{V}_i,2\right), k\right\rangle .$$

La composizione delle $\mathbf{V}_i$ può essere effettuata concatenando i programmi che generano ciascuna di esse. *Il gruppo di trasformazioni così generato è denso nel gruppo* U(2) *delle trasformazioni unitarie nello spazio di Hilbert di un bit*[1] . Inoltre per quasi ogni $\alpha$ multiplo irrazionale di $\pi$ *qualunque trasformazione* $\mathbf{U}$ *fissata può essere approssimata efficientemente* nel senso seguente: dato $\varepsilon>0$, $\mathbf{U}$ può essere approssimata a meno di $\varepsilon$ con una concatenazione di $\mathbf{V}_i$ di lunghezza crescente in modo polinomiale con $1/\varepsilon$.

Quest'ultima proprietà può essere resa intuitivamente evidente con la seguente analogia. Consideriamo per esempio le rotazioni reali del piano, e sia $\mathbf{R}_\alpha$ la rotazione di un angolo $\alpha$ fissato multiplo irrazionale di $\pi$. Se accettiamo che le

[1] Questa scelta dei generatori è quella originariamente considerata da Deutsch. Essa può essere considerevolmente semplificata. Bernstein e Vazirani hanno dimostrato che sono sufficienti una singola rotazione irrazionale insieme alla trasformazione definita da $|0\rangle \rightarrow |0\rangle$ e $|1\rangle \rightarrow |1\rangle$. Questo problema è strettamente connesso con la determinazione di famiglie di gate universali nel modello circuitale della computazione quantistica.

prime $k$ potenze di $\mathbf{R}_\alpha$ trasformino un punto di una circonferenza negli altri punti praticamente a caso, allora, siccome la circonferenza ha lunghezza finita, raddoppiando $k$ raddoppieremo il numero di punti sulla circonferenza, e questo ci permetterà di approssimare qualsiasi rotazione fissata con precisione doppia. Per un'accuratezza fissata $\varepsilon$ avremo perciò bisogno di $O(1/\varepsilon)$ applicazioni di $\mathbf{R}_\alpha$, un polinomio lineare in $1/\varepsilon$. Un ragionamento simile si può ragionevolmente estendere ad ogni gruppo compatto come U(2).

E' analogamente possibile approssimare qualunque trasformazione unitaria nello spazio $2^n$ dimensionale di $n$ bit. Questa considerazione non è banale, in quanto lo stato complessivo degli $n$ bit non è in generale il prodotto di stati individuali dei singoli bit, bensì una combinazione lineare di stati fattorizzabili. Si dimostra però che qualunque matrice unitaria d-dimensionale **U** può essere scritta come il prodotto di un numero polinomiale di trasformazioni unitarie, esattamente $2d^2-d$, ciascuna delle quali agisce soltanto su un paio di stati della base computazionale dei bit coinvolti [Ekert].

Indichiamo gli stati di tale base con $|e_1\rangle = |000\ldots00\rangle$, $|e_2\rangle = |000\ldots01\rangle$, ..., $|e_d\rangle = |111\ldots11\rangle$. Dato un vettore di componenti $(a_1, a_2, \ldots a_d)$ nella base computazionale possiamo trasformarlo in $(1,0,\ldots,0)$ utilizzando una serie di $d-1$ trasformazioni bidimensionali. Per prima cosa applichiamo

$$\mathbf{A}_2 = \frac{1}{\sqrt{|a_1|^2 + |a_2|^2}} \begin{pmatrix} a_1^* & a_2^* \\ a_2 & -a_1 \end{pmatrix}$$

nello spazio generato da $\{|e_1\rangle, |e_2\rangle\}$, col che la seconda componente è ridotta a zero. Similmente, usiamo $\mathbf{A}_3$, ..., $\mathbf{A}_d$ rispettivamente negli spazi generati da $\{|e_1\rangle, |e_3\rangle\}$, ..., $\{|e_1\rangle, |e_d\rangle\}$ per ridurre a zero anche $a_3$, ..., $a_d$. Abbiamo così che $\mathbf{A}_d \ldots \mathbf{A}_3 \mathbf{A}_2$ mappa $(a_1, a_2, \ldots a_d)$ in $(1,0,\ldots,0)$, mentre $\mathbf{A}_2^{-1}\mathbf{A}_3^{-1}\ldots\mathbf{A}_d^{-1}$ compie l'azione inversa.

Siano ora $|\varphi_1\rangle,...|\varphi_d\rangle$ gli autovettori di **U** ed $e^{i\varphi_1},...,e^{i\varphi_d}$ i rispettivi autovalori. Usando la procedura sopra descritta trasformiamo $|\varphi_1\rangle$ in $|e_1\rangle$, moltiplichiamo $|e_1\rangle$ per $e^{i\varphi_1}$ ed applichiamo la trasformazione inversa per ottenere il multiplo richiesto di $|\varphi_1\rangle$. Questo comporta un totale di (d−1)+1+(d−1)=2d−1 trasformazioni unitarie bidimensionali. Ripetendo per ognuno dei d autovettori, avremo espresso la matrice **U** come prodotto di d(2d−1) matrici unitarie bidimensionali.

Il risultato suesposto, unitamente al fatto che le permutazioni degli stati della base computazionale rappresentano funzioni ricorsive bigettive e come tali sono computabili da $\mathcal{T}$ e quindi da $\mathcal{Q}$, permette di dimostrare che *per ogni matrice unitaria* d-*dimensionale* **U** *esiste un programma di lunghezza polinomiale in* d/ε *che la approssima a meno di* ε. Se abbiamo un algoritmo $\mathcal{A}$ che impiega *k* passi per la sua esecuzione, e quindi coinvolge al più *k* trasformazioni unitarie $\mathbf{U}_i$, possiamo usare $\mathcal{Q}$ per approssimarlo con precisione arbitraria ε approssimando ogni $\mathbf{U}_i$ a meno di ε/*k*. Questo richiede al più un numero di passi pari a *k* poly(d*k*/ε), dove d è la dimensione massima delle $\mathbf{U}_i$. Inoltre se $\mathcal{A}$ è un algoritmo efficiente, ovvero esegue in *k*=poly(log *n*) passi se l'ingresso è *n*, ed ogni $\mathbf{U}_i$ ha dimensione polinomiale nel logaritmo di *n* allora *per ogni accuratezza fissata* ε *l'algoritmo approssimante è efficiente*.

E' importante notare che una trasformazione unitaria generica di dimensione d richiede un programma di poly(d) passi per essere implementata. Consideriamo ora il già menzionato algoritmo di Shor per la fattorizzazione dei numeri naturali. Un ingresso N, codificato in $\log_2$ N sistemi a 2 stati, occupa uno spazio di Hilbert N-dimensionale; una generica trasformazione dell'ingresso implementata con il criterio suesposto richiederebbe quindi poly(N) passi, che è esponenziale nelle dimensioni dell'ingresso $\log_2$ N. L'efficienza dell'algorimo dipende così dall'uso di una particolare trasformazione unitaria - in effetti la

trasformata di Fourier discreta - implementabile in poly(log N) anziché in poly(N) passi.

Siamo ora in grado di affermare che la QTM $Q$ sopra introdotta costituisce il ***calcolatore universale quantistico.*** Essa può infatti simulare con precisione arbitraria una qualunque altra QTM $C[U^+,U^-]$ poiché, pur essendo lo spazio delle fasi di $C$ infinito-dimensionale, per simularne l'evoluzione è necessario effettuare ad ogni passo solo una trasformazione unitaria in un sottospazio a dimensione finita[1]

## *Simulazione dei sistemi fisici*

*"What, in other words, is the universal quantum simulator?*
*[...] I believe it's rather simple to answer that question [...],*
*but I just haven't done it."*

*Richard P. Feynman*

Come accennato in precedenza, una delle motivazioni principali per lo studio dei calcolatori quantistici è la ricerca di un dispositivo che permetta di simulare efficientemente l'evoluzione di un sistema quantistico. Nel suo articolo del 1982, dal quale è tratta la citazione, Feynman notava che le analogie presenti nel comportamento di sistemi quantistici apparentemente molto diversi suggeriscono l'idea dell'esistenza di classi di fenomeni intersimulabili, e si

[1] Nel paragrafo precedente abbiamo considerato **U** come una trasformazione sui bit del nastro, mentre la dinamica di una QTM comporta evidentemente una correlazione fra gli stati interni della testina e lo stato del nastro. Bisogna però ricordare che, analogamente a quanto visto per il caso classico, la descrizione completa di $C$, inclusiva degli stati interni, è riportata per la simulazione sul nastro di $Q$. così facendo la più generica evoluzione di $C$ è comunque riproducibile con una trasformazione **U** limitata ad un sottoinsieme del nastro di $Q$. Gli stati interni di $Q$ intervengono solo nell'implementazione di **U**, e non codificano quindi in alcun modo la dinamica di $C$.

domandava se fosse possibile caratterizzare tali classi giungendo ad un risultato analogo alla tesi di Church nella teoria classica della computazione.

In risposta a questo quesito, Deutsch propone il calcolatore quantistico introdotto nella sezione precedente come "simulatore universale", in grado di riprodurre l'evoluzione di qualunque sistema fisico con spazio delle fasi finito-dimensionale [Deutsch]. Sebbene ricalchi da vicino la costruzione delle macchine di Turing tradizionali, la QTM risulta infatti costituita da bit quantistici, ciascuno dei quali può accedere ad un continuo di stati (e può quindi, ad esempio, avere una probabilità $p$ anche non computabile di trovarsi nello stato $|0\rangle$ piuttosto che nello stato $|1\rangle$). Questo suggerisce che $Q$ possa essere opportunamente programmata per simulare esattamente, e non solo entro una certa approssimazione, l'evoluzione di un sistema fisico dato.