APPUNTI SU

ALGORITMI, MACCHINE DI TURING, COMPUTABILITÀ[†]

MARCELLO FRIXIONE

[†] *Una versione di questi appunti è apparsa come §1.7 del volume* Informatica *di Gianni Vercelli e Renato Zaccaria, casa editrice Editoriale Scientifica, Napoli, 1998.*

Algoritmi, macchine di Turing, computabilità

1. Verso una caratterizzazione rigorosa del concetto di algoritmo

Durante tutta la storia delle matematiche sono stati sviluppati algoritmi per risolvere classi sempre più estese di problemi. Tuttavia è soltanto in anni recenti che il concetto stesso di algoritmo è stato problematizzato in maniera esplicita, e fatto oggetto diretto di ricerca matematica. Ciò è avvenuto attorno agli anni '30 del '900, nel contesto delle ricerche sui fondamenti della matematica. Le ricerche precedenti si erano basate su di una nozione di algoritmo del tutto intuitiva, non specificata in modo rigoroso. Tale nozione intuitiva fu del tutto sufficiente fin tanto che lo scopo che ci si proponeva era quello di individuare algoritmi che risolvessero problemi, o classi di problemi determinate. Ma con le ricerche sui fondamenti della matematica avvenne un radicale cambiamento di prospettiva. Furono poste domande di tipo nuovo, che non riguardavano più la possibilità di individuare algoritmi specifici, ma che concernevano l'intera classe dei procedimenti di tipo algoritmico. Soprattutto nel contesto del progetto fondazionalista proposto da David Hilbert, diventava fondamentale rispondere alla domanda se esistessero problemi matematici che non ammettono neppure in linea di principio di essere risolti mediante alcun algoritmo. Tutto ciò era a sua volta strettamente legato allo studio delle proprietà dei sistemi formali della logica matematica. Nel momento in cui la ricerca si indirizzò allo studio delle proprietà della classe di *tutti* i procedimenti algoritmici, tale classe dovette essere caratterizzata in maniera rigorosa, e non fu più sufficiente la tradizionale definizione informale ed intuitiva. Nacque così quel settore della logica matematica che è stato detto in seguito *teoria della computabilità* (o della *calcolabilità*) *effettiva* (oppure anche *teoria della ricorsività*), in cui vengono indagati concetti quali quello di algoritmo e di funzione computabile in modo algoritmico.

Durante gli anni '30 numerosi tra i maggiori logici del periodo, tra i quali Gödel, Church, Post, Kleene e Turing, affrontarono, da punti di vista differenti, il problema di individuare una definizione rigorosa della nozione di algoritmo. In queste pagine verrà presentato uno di questi approcci, quello seguito dal logico inglese Alan Turing, che, rispetto agli studi coevi sulla computabilità, presenta il vantaggio di affrontare il problema in maniera diretta, analizzando il comportamento di un soggetto umano computante, senza presupporre altre nozioni o strumenti formali elaborati nell'ambito della ricerca logico-matematica. Inoltre, Turing ha formulato la sua proposta nei termini di una particolare classe di macchine astratte, che possono essere considerate modelli idealizzati dei calcolatori reali. Infine, parte dell'interesse per il lavoro di Turing risiede nelle implicazioni avute in altri ambiti disciplinari, quali la filosofia della mente, l'intelligenza artificiale e le scienze cognitive.

Prima di esaminare la proposta di Turing, riprendiamo brevemente la nozione informale di algoritmo. Intuitivamente, si dispone di un *algoritmo* (o di un *metodo effettivo*) per risolvere un problema se si dispone di un elenco *finito* di istruzioni tali che:

1) a partire dai dati iniziali, le istruzioni sono applicabili in maniera rigorosamente deterministica, in maniera cioè che ad ogni passo sia sempre possibile stabilire univocamente qual è l'istruzione che deve essere applicata al passo successivo;

2) si dispone di un criterio univoco per stabilire quando si è raggiunto uno *stato finale*, quando cioè il processo deve considerarsi terminato e il risultato, se esiste, è stato ottenuto;

3) uno stato finale deve sempre essere raggiungibile in un numero finito di passi.

Chiameremo *input* i dati di partenza del calcolo; *output* il suo risultato.

Una *funzione* si dice *calcolabile in modo algoritmico* (o *calcolabile in modo effettivo*, o *effettivamente calcolabile*) se esiste un algoritmo che consente di calcolarne i valori per tutti gli argomenti.

Esempi di algoritmi possono essere tratti dalle matematiche elementari: sono algoritmi, ad esempio, gli insiemi di regole che consentono di eseguire le quattro operazioni, come pure è un algoritmo il

procedimento euclideo per la ricerca del massimo comun divisore di due numeri naturali. In logica, il metodo delle tavole di verità è un algoritmo che permette di stabilire se una formula del calcolo proposizionale è o meno una tautologia.

Per le caratteristiche di determinismo e di finitezza che abbiamo enunciato, ogni algoritmo si presta, almeno in linea di principio, ad essere automatizzato, ad essere eseguito cioè da una macchina opportunamente progettata. Con lo sviluppo dell'informatica, la teoria della computabilità ha dunque assunto, in un certo senso, il ruolo di "teoria dei fondamenti" per questa disciplina.

2. Le macchine di Turing

Turing affrontò il problema di fornire un equivalente rigoroso del concetto intuitivo di algoritmo definendo un modello dell'attività di un essere umano che stia eseguendo un calcolo di tipo algoritmico. Egli elaborò tale modello nella forma di una classe di dispositivi computazionali, di macchine calcolatrici astratte, che in seguito furono dette appunto *macchine di Turing* (d'ora in avanti MT). Le MT sono macchine astratte nel senso che, nel caratterizzarle, non vengono presi in considerazione quei vincoli che sono fondamentali se si intende progettare una macchina calcolatrice reale (ad esempio, le dimensioni della memoria, i tempi del calcolo, e così via), e soprattutto nel senso che esse sono definite a prescindere dalla loro realizzazione fisica (cioè, dal tipo di *hardware* utilizzato). Vale a dire, che cosa sia una MT dipende esclusivamente dalle relazioni funzionali che sussistono tra le sue parti, e non dal fatto di poter essere costruita con particolari dispositivi materiali.

Seguiamo l'analisi del processo di calcolo come viene condotta da Turing stesso nell'articolo "On computable numbers, with an application to the Entscheidungsproblem" (Turing 1936-37), dove il concetto di MT viene formulato per la prima volta. Un calcolo, osserva Turing, consiste nell'operare su di un certo insieme di simboli scritti su di un supporto fisico, ad esempio un foglio di carta. Turing argomenta che il fatto che abitualmente venga usato un supporto bidimensionale è inessenziale, e che si può quindi assumere, senza nulla perdere in generalità, che la nostra macchina calcolatrice utilizzi per la "scrittura" un *nastro* monodimensionale di lunghezza virtualmente illimitata in entrambe le direzioni (tuttavia, come vedremo, in ogni fase del calcolo la macchina potrà disporre soltanto di una porzione finita di esso). Tale nastro sia inoltre suddiviso in celle, in "quadretti", "come un quaderno di aritmetica per bambini", ciascuna delle quali potrà ospitare un solo simbolo alla volta (fig. 1).

Quanto ai simboli da utilizzare per il calcolo, ogni macchina potrà disporre soltanto di un insieme finito di essi, che chiameremo l'*alfabeto* di quella macchina. Il fatto che l'alfabeto di cui si può disporre sia finito non costituisce comunque una grave limitazione. È infatti sempre possibile rappresentare un nuovo simbolo mediante una sequenza finita di simboli dell'alfabeto, ed avere così la possibilità di esprimere un numero virtualmente infinito di simboli (come avviene usualmente nella numerazione decimale mediante cifre arabe). Sia dunque $\Sigma \equiv \{s_1, s_2, \ldots, s_n\}$ l'alfabeto di una generica MT. Ogni cella del nastro potrà contenere uno di tali simboli, oppure, in alternativa, restare vuota (indicheremo con s_0 la cella vuota).

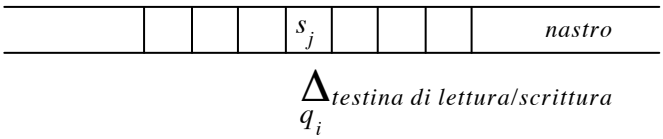

Fig. 1

Vi è senza dubbio un limite al numero di simboli che un essere umano può osservare senza spostare lo sguardo sul foglio su cui sta lavorando. Secondo Turing si può quindi assumere senza perdita di generalità che la macchina possa esaminare soltanto una cella alla volta, ed "osservare" ad ogni passo al più un

singolo simbolo. A tal fine la macchina sarà dotata di una *testina di lettura*, che sarà collocata, in ogni fase del calcolo, su di una singola cella (fig. 1). Essa, per poter accedere alle altre celle del nastro, dovrà spostarsi verso destra o verso sinistra. Chi sta eseguendo un calcolo ha poi la possibilità di scrivere nuovi simboli, di cancellare quelli già scritti o di sostituirli con altri. La testina eseguirà anche tali compiti di cancellazione e di scrittura. Anche in questo caso però essa potrà agire soltanto sulla cella "osservata", e, per accedere ad altre celle, dovrà prima spostarsi lungo il nastro. Poiché ogni cella può contenere un solo simbolo, scrivendo un nuovo simbolo in una cella il simbolo eventualmente presente in essa si deve ritenere cancellato.

Nell'eseguire un calcolo, un essere umano tiene conto delle operazioni già eseguite e dei simboli osservati in precedenza mediante la propria memoria, cambiando cioè il proprio "stato mentale". Al fine di simulare ciò, supporremo che una macchina possa assumere, in dipendenza dagli eventi precedenti del processo di calcolo, un certo numero di *stati interni* (uno e non più di uno alla volta), che corrispondano agli "stati mentali" dell'essere umano. Tali stati saranno in numero finito, poiché (usando la parole dello stesso Turing) "se ammettessimo un'infinità di stati mentali, alcuni di essi sarebbero 'arbitrariamente prossimi', e sarebbero quindi confusi". Il limitarsi ad un numero finito di stati non costituisce tuttavia un vincolo, in quanto "l'uso di stati mentali più complicati può essere evitato scrivendo più simboli sul nastro". Siano allora q_0, q_1, ... , q_m gli stati che una generica MT può assumere. Nella rappresentazione grafica indicheremo sotto la testina di lettura/scrittura lo stato della macchina nella fase di calcolo rappresentata. Definiamo *configurazione* di una MT in una data fase di calcolo la coppia costituita dallo stato interno che essa presenta in quel momento e dal simbolo osservato dalla testina (la configurazione della macchina raffigurata nella fig. 1 è dunque (q_i, s_j)).

Una MT può dunque eseguire operazioni consistenti in spostamenti della testina lungo il nastro, scrittura e cancellazione di simboli, mutamenti dello stato interno. Scomponiamo tali operazioni in un numero di *operazioni atomiche*, tali da non poter essere ulteriormente scomposte in operazioni più semplici. Nel tipo di macchina descritto ogni operazione può essere scomposta in un numero finito delle operazioni seguenti: (1) sostituzione del simbolo osservato con un altro simbolo (eventualmente con s_0; in tal caso si ha la cancellazione del simbolo osservato), e/o (2) spostamento della testina su di una delle celle immediatamente attigue del nastro. Ognuno di tali atti può inoltre comportare (3) un cambiamento dello stato interno della macchina. Nella sua forma più generale, ogni operazione atomica dovrà quindi consistere di un operazione di scrittura e/o di uno spostamento atomico, ed eventualmente di un mutamento di stato. Indicheremo d'ora in avanti rispettivamente con le lettere S, D e C il fatto che una macchina debba eseguire uno spostamento di una cella verso sinistra, di una cella verso destra, oppure non debba eseguire alcuno spostamento (dove C sta per "centro"). Grazie a ciò potremo rappresentare ogni operazione atomica mediante una terna, il primo elemento della quale starà ad indicare il simbolo che deve essere scritto sulla cella osservata, il secondo quale spostamento deve essere eseguito (S, D o C), il terzo infine lo stato che la macchina deve assumere alla fine dell'operazione. Ad esempio, la terna:

$$s_i \ S \ q_j$$

significa che la macchina deve scrivere il simbolo s_i sulla cella osservata, spostarsi di una cella a sinistra, ed assumere infine lo stato q_j. Invece la terna:

$$s_0 \ C \ q_p$$

significa che la macchina deve cancellare il simbolo osservato, non eseguire alcun movimento ed assumere lo stato q_p.

Ogni singola MT è "attrezzata" per eseguire un tipo di calcolo specifico, dispone cioè di una serie di regole, di istruzioni, che le permettano di eseguire il compito per il quale è stata progettata. In un calcolo algoritmico ogni passo deve essere completamente determinato dalla situazione precedente. Nel caso di un calcolatore umano, ogni sua mossa deve dipendere esclusivamente dal ricordo delle operazioni già eseguite e dai simboli che egli può osservare. Analogamente, in una MT, poiché in ogni fase del calcolo la macchina "sa" soltanto in quale stato si trova e quale è il simbolo sulla cella osservata del nastro (cioè sa quale è la sua configurazione corrente), e poiché ogni operazione può essere scomposta in operazioni atomiche, allora ogni generica istruzione avrà la forma seguente:

<configurazione> → <azione atomica>.

In altri termini, ogni istruzione deve specificare quale operazione atomica deve essere eseguita a partire da una determinata configurazione. Un esempio di istruzione è:

$$q_i \ s_j \ \rightarrow s_{j'} \ D \ q_{i'},$$

che deve essere interpretata come segue: qualora la macchina si trovi nello stato q_i ed il simbolo osservato sia s_j, allora il simbolo $s_{j'}$ dovrà essere scritto sul nastro al posto di s_j, la testina dovrà spostarsi di una cella verso destra e la macchina dovrà assumere lo stato $q_{i'}$. In generale, poiché ogni configurazione è rappresentabile mediante una coppia, ed ogni operazione atomica mediante una terna, un'istruzione (in cui di solito il simbolo "→ " viene omesso e considerato sottinteso) avrà la forma di una *quintupla*, i primi due elementi della quale (uno stato interno ed un simbolo dell'alfabeto) indicano la configurazione di partenza, mentre gli ultimi tre elementi specificano l'operazione che deve essere eseguita. Le istruzioni di cui dispone ogni singola MT per eseguire il calcolo per il quale è stata progettata avranno quindi la forma di un opportuno insieme finito di quintuple (che verrà detto la *tavola* di quella MT). Una volta fissato l'alfabeto, ciò che caratterizza ogni singola MT rispetto a tutte le altre è appunto la tavola delle sue quintuple. Affinché un insieme di quintuple costituisca la tavola di una MT è indispensabile che venga rispettata la seguente condizione: poiché il calcolo deve essere deterministico, a partire da una singola configurazione non devono essere applicabili istruzioni diverse. Ciò corrisponde alla condizione che, nella tavola di una macchina, non possano comparire più quintuple con i primi due elementi uguali.

Affinché il calcolo possa terminare, è necessario che ad alcune delle configurazioni possibili non corrisponda alcuna quintupla, altrimenti, qualunque fosse il risultato di una mossa, esisterebbe sempre un'altra mossa che ad essa dovrebbe far seguito. Chiameremo tali configurazioni *configurazioni finali*. Data una MT, è sempre possibile costruirne un'altra che esegua lo stesso calcolo, per la quale esista uno specifico stato interno che compare in tutte e sole le configurazioni finali. Chiameremo tale stato *stato finale*, e stabiliremo convenzionalmente di riservare ad esso il simbolo q_0. Data una MT generica, per trasformarla in una che abbia q_0 come stato finale si proceda nel modo seguente. Sia (q_n, s_m) una generica configurazione finale della macchina di partenza. La tavola della nuova macchina si ottiene aggiungendo tutte le quintuple del tipo:

$$q_n \ s_m \ s_m \ C \ q_0.$$

In tutti i casi in cui la macchina di partenza giungeva in uno stato finale, la nuova macchina farà un'ulteriore mossa, assumendo lo stato q_0 (e lasciando inalterato tutto il resto).

I dati vengono forniti a una MT sotto forma di una sequenza finita di simboli dell'alfabeto scritti sul nastro prima dell'inizio del calcolo. Chiameremo *input* tale sequenza di simboli. Il risultato è costituito da ciò che è scritto sul nastro al momento della fermata, e ciò costituisce l'*output* del calcolo. Stabiliamo

convenzionalmente che all'inizio del calcolo la testina debba essere collocata in *posizione standard*, vale a dire in corrispondenza del primo simbolo a sinistra dell'input; inoltre lo stato interno della macchina debba essere q_1.

Vediamo un semplice esempio di MT. Si consideri l'alfabeto $\Sigma \equiv \{|\}$, composto come unico simbolo da una barra verticale. Definiamo una macchina che, presa come input una successione di barre consecutive, restituisca come output tale successione aumentata di un elemento. A tal fine è sufficiente disporre del solo stato interno q_1 (oltre allo stato finale q_0); la tavola della macchina sarà la seguente:

$$\begin{array}{cccc} q_1 & | & D & q_1 \\ q_1 & s_0 & C & q_0. \end{array}$$

Secondo le convenzioni stabilite, alla partenza la testina deve essere collocata sul primo simbolo a sinistra dell'input, e lo stato di partenza deve essere q_1 (fig. 2).

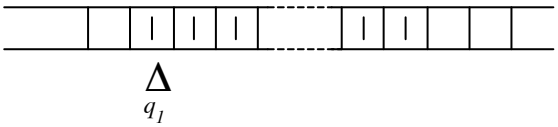

Fig. 2

Fintanto che la testina trova celle segnate con | allora, in virtù della prima quintupla, viene riscritto | sulla cella osservata (cioè, vengono lasciate le cose come stanno), e la testina si sposta a destra di una cella mantenendo lo stato q_1 (fig. 3).

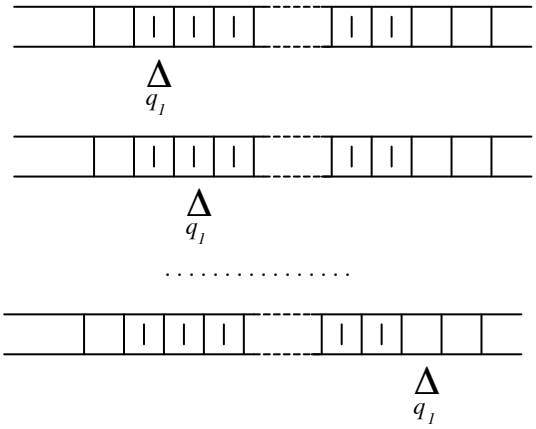

Fig. 3

Quando la testina incontra una cella vuota viene attivata la seconda quintupla, in virtù della quale la macchina deve segnare con una barra la cella osservata, non eseguire alcuno spostamento, ed assumere lo stato finale q_0 (fig. 4).

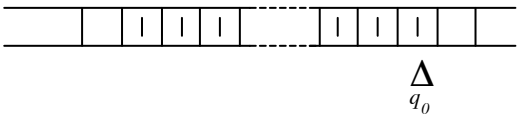

Fig. 4

Sin qui abbiamo considerato MT che eseguono calcoli su dati generici. Vediamo ora come si possano codificare i numeri naturali in modo da definire MT che calcolino funzioni aritmetiche. Utilizziamo come alfabeto $\Sigma \equiv \{|\}$. I numeri naturali vengono codificati come segue. Al numero *0* viene fatta corrispondere la

sequenza composta da una sola barra. In generale, ogni numero n viene codificato da una sequenza di $n + 1$ barre. Una n-pla di numeri naturali $(k_1, \ldots, k_n)$ viene codificata sul nastro scrivendo la sequenza di barre corrispondente ad ogni k_i (con $1 \leq i \leq n$), e lasciando una cella vuota come separatore tra ognuna di tali sequenze. Ad esempio, la terna (4, 1, 0) viene codificata come in fig. 5.

Diremo che una macchina M_φ *computa* una funzione aritmetica φ ad n argomenti (con $n \geq 1$) *sse* quanto segue vale per ogni n-pla $(x_1, \ldots, x_n)$ di numeri naturali. Sia $(x_1, \ldots, x_n)$ codificata nel modo sopra descritto e collocata in posizione standard rispetto alla testina (essendo vuota ogni altra cella del nastro). Allora $\varphi(x_1, \ldots, x_n) = y$ *sse*, al termine del calcolo, l'output di M_φ è costituito dalla codifica di y.

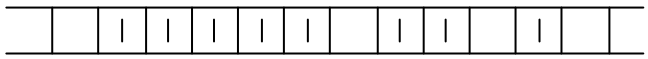

Fig. 5

Diremo che una funzione φ è *T-computabile sse* esiste una MT M_φ che la computa.

3. Esempi di macchine di Turing

In questo paragrafo presentiamo alcuni esempi di MT che eseguono semplici calcoli. Eccetto i casi in cui sia indicato esplicitamente, ognuna delle macchine ha $\Sigma \equiv \{|\}$, e, al momento dell'avvio, la testina deve essere collocata sulla prima cella a sinistra dell'input. Lo stato iniziale è q_1.

1. MT che esegue l'addizione di due numeri. Input: codifica di una coppia di numeri naturali.

q_1	|	|	D	q_1		q_1	s_0	|	D	q_2
q_2	|	|	D	q_2		q_2	s_0	s_0	S	q_3
q_3	|	s_0	S	q_4		q_4	|	s_0	C	q_0

La macchina riempie con una barra la cella vuota che separa i due numeri dell'input, dopo di che cancella le due barre finali del secondo numero.

2. MT che raddoppia il numero di | consecutive che le viene dato in input. Input: sequenza di |.

q_1	|	s_0	D	q_2		q_2	|	|	D	q_2	
q_2	s_0	s_0	D	q_3		q_3	|	|	D	q_3	
q_3	s_0	|	D	q_4		q_4	s_0	|	S	q_5	
q_5	|	|	S	q_5		q_5	s_0	s_0	S	q_6	
q_6	|	|	S	q_7		q_6	s_0	s_0	C	q_0	(*)
q_7	|	|	S	q_7		q_7	s_0	s_0	D	q_1	

La macchina cancella la prima barra a destra dell'input, dopo di che si colloca a destra dell'input (lasciando una cella vuota come separatore), e stampa due barre. Torna quindi indietro, e ripete l'operazione sino a che tutte le barre dell'input sono state cancellate.

3. MT che calcola la funzione $\varphi(x)=2x$. Input: codifica di un numero naturale. La tavola è la stessa della macchina precedente, in cui la quintupla (*) è stata sostituita dalle tre quintuple seguenti:

q_6	s_0	s_0	D	q_8		q_8	s_0	s_0	D	q_8

q_8	|	s_0	C	q_0

Poiché la codifica di un numero n è costituita da $n+1$ barre, la codifica di $2n$ è costituita da $2n+1 = 2(n+1)-1$ barre. Quindi questa macchina, dopo avere raddoppiato il numero delle barre in input, cancella una barra e si ferma.

4. MT che calcola la funzione il cui valore è 1 se l'argomento è un numero pari, 0 se l'argomento è dispari. Input: codifica di un numero naturale.

q_1	|	s_0	D	q_2		q_2	|	s_0	D	q_1
q_1	s_0	|	C	q_0		q_2	s_0	|	D	q_3
q_3	s_0	|	C	q_0						

La macchina cancella successivamente tutte le barre dell'input, assumendo alternativamente gli stati q_1 e q_2. Se, quando l'intero input è stato cancellato, lo stato è q_1 (il che accade se l'input era la codifica di un numero dispari), la macchina stampa una barra (la codifica di 0). Altrimenti, se lo stato è q_2 (se cioè l'input era pari), stampa due barre (la codifica di 1). Dopo di che si ferma.

5. MT che calcola la differenza tra due numeri naturali. Input: codifica di una coppia di numeri naturali, di cui il primo maggiore o uguale del secondo. All'inizio del calcolo la testina deve essere collocata sulla prima cella *a destra* dell'input.

q_1	|	s_0	S	q_2		q_2	s_0	s_0	C	q_0
q_2	|	|	S	q_3		q_3	|	|	S	q_3
q_3	s_0	s_0	S	q_4		q_4	s_0	s_0	S	q_4
q_4	|	s_0	D	q_5		q_5	s_0	s_0	D	q_5
q_5	|	|	D	q_6		q_6	|	|	D	q_6
q_6	s_0	s_0	S	q_1						

La macchina cancella una barra dalla codifica del secondo numero in input. Dopo di che cancella alternativamente una barra dalla codifica del secondo e del primo numero in input, sino a che la codifica del secondo numero non è stata cancellata completamente.

6. MT che controlla se una sequenza di parentesi è bilanciata, se cioè, per ogni parentesi aperta, esiste una parentesi chiusa corrispondente. $\Sigma \equiv \{ \, (\, , \,) \, , X \, \}$. Input: una sequenza di *(* e *)* (senza celle vuote in mezzo). Output: una sequenza di sole X se le parentesi dell'input erano bene accoppiate; altrimenti, una sequenza di simboli di Σ comprendente *(* oppure *)*.

q_1	(	(	D	q_1		q_1	X	X	D	q_1
q_1	)	)	S	q_2		q_2	X	X	S	q_2
q_2	(	X	D	q_3		q_3	X	X	D	q_3
q_3	)	X	D	q_1		q_1	s_0	s_0	C	q_0
q_2	s_0	s_0	C	q_0						

La macchina percorre l'input da sinistra verso destra mantenendosi nello stato q_1 finché non trova una *)*; allora passa in q_2 e torna in dietro fino alla prima *(* che incontra, che sostituisce con una X; assume quindi lo

stato q_3 e torna a destra, a sostituire con *X* anche la *)* precedentemente individuata; dopo di che, torna in q_1 e ripete da capo l'operazione; si ferma non appena la testina incontra una cella vuota.

4. La tesi di Church

Nel 1936 il logico americano Alonzo Church, in seguito alle sue ricerche sulla computabilità effettiva, propose di identificare la classe delle funzioni calcolabili mediante un algoritmo (o funzioni effettivamente calcolabili) con una particolare classe di funzioni aritmetiche, detta in seguito classe delle *funzioni ricorsive generali* (Church 1936). Tale identificazione divenne nota col nome di *Tesi di Church*. È possibile dimostrare l'equivalenza tra la classe delle funzioni ricorsive generali e la classe delle funzioni T-computabili, in quanto ogni funzione T-computabile è ricorsiva generale, e viceversa (Turing 1937). La Tesi di Church può quindi essere formulata come segue:

una funzione è effettivamente calcolabile sse è T-computabile.

Che ogni funzione ricorsiva generale (o T-computabile) sia effettivamente computabile segue direttamente e in modo ovvio dalla definizione di T-computabilità e di MT. Ciò che invece è interessante nella Tesi di Church è l'implicazione inversa, secondo la quale ogni procedimento algoritmico è riconducibile alla ricorsività generale. Algoritmo e funzione computabile in modo effettivo sono concetti intuitivi, non specificati in modo rigoroso, per cui non è possibile una dimostrazione formale di equivalenza con il concetto di funzione ricorsiva generale. La Tesi di Church non è dunque una congettura che, in linea di principio, potrebbe un giorno diventare un teorema. Tuttavia, la nozione intuitiva di funzione computabile in modo effettivo è contraddistinta da un insieme di caratteristiche (quali determinismo, finitezza di calcolo, eccetera) che possiamo considerare in larga misura "oggettive". Questo fa sì che sia praticamente sempre possibile una valutazione concorde nel decidere se un dato procedimento di calcolo debba essere considerato algoritmico o meno. Quindi, almeno in linea di principio, è ammissibile che venga "scoperto" un controesempio alla Tesi di Church; è ammissibile cioè che venga individuata una funzione effettivamente calcolabile secondo questi parametri intuitivi, la quale non sia allo stesso tempo ricorsiva generale. In questo paragrafo esporremo le ragioni per cui si ritiene improbabile che un evento del genere si verifichi.

Prima di procedere, è opportuno un chiarimento. Le funzioni ricorsive generali (o T-computabili) sono esclusivamente funzioni aritmetiche. Ciò potrebbe sembrare troppo restrittivo, in quanto esistono algoritmi definiti su oggetti diversi dai numeri naturali. Vi sono algoritmi che stabiliscono se un certo oggetto matematico appartiene a un dato insieme o meno. Ve ne sono altri che eseguono operazioni simboliche sulle espressioni di un sistema formale, stabilendo ad esempio se una data formula gode o meno di una certe proprietà. In logica matematica tuttavia sono state sviluppate tecniche mediante le quali algoritmi di tipo diverso, quali quelli sopra citati, possono essere ricondotti a funzioni aritmetiche. I numeri naturali infatti possono essere utilizzati per rappresentare, mediante opportune codifiche, dati o informazioni di varia natura, purché di tipo discreto. Nel caso delle MT, si può dimostrare che ogni macchina con un alfabeto Σ di simboli finito può essere trasformata in una macchina equivalente che calcola una funzione aritmetica T-computabile come definita nel par. 2.

Seguendo in parte l'analisi del logico S.C. Kleene (1952, §62), raccoglieremo gli argomenti a favore della Tesi di Church in due gruppi, (a) e (b).

(a) Il primo gruppo di argomenti poggia su quella che si può chiamare *evidenza euristica*. Rientra in questo gruppo la constatazione che, per ogni singola funzione calcolabile che sia stata esaminata, è sempre stato possibile dimostrare la sua appartenenza alla classe delle funzioni ricorsive generali. Analogamente, si è dimostrato che le operazioni note per definire funzioni effettivamente calcolabili a partire da altre funzioni effettivamente calcolabili conservano la ricorsività generale. Tale indagine è stata condotta per un grande

numero di funzioni, di classi di funzioni e di operazioni. Infine, i vari metodi tentati per costruire funzioni effettivamente calcolabili che non fossero ricorsive generali hanno condotto tutti al fallimento, nel senso che le funzioni ottenute erano tutte a loro volta ricorsive generali, oppure non erano calcolabili in modo effettivo.

(b) Nel secondo gruppo di argomenti viene considerata l'*equivalenza delle diverse formulazioni* proposte. Abbiamo accennato al fatto che numerosi studiosi hanno lavorato ad una definizione rigorosa del concetto di algoritmo. Ebbene, tutti i tentativi che furono elaborati per caratterizzare in modo rigoroso la classe di tutte le funzioni effettivamente computabili si rivelarono equivalenti, nel senso che la classe di funzioni ottenuta era sempre la classe delle funzioni ricorsive generali. Ciò che è particolarmente rilevante ai fini di una "corroborazione" della Tesi di Church è la diversità degli strumenti e dei concetti impiegati nelle diverse formulazioni. In molti casi tali formulazioni traggono la loro origine da concetti matematici preesistenti. Nel caso della *ricorsività generale di Herbrand-Gödel* si prendono le mosse dal concetto di sistema di equazioni, nella λ-*ricorsività* di Church (1936) si parte dall'idea di un calcolo di sole funzioni, il λ-*calcolo*. Schönfinkel (1924) e Curry (1929, 1930, 1932) elaborarono il cosiddetto *calcolo dei combinatori*. Ad E. Post (1943, 1946) è dovuto l'approccio basato sui *sistemi normali* o *canonici*. Negli anni cinquanta, il logico sovietico A. A. Markov (1951, 1954) propose un'ulteriore formulazione tramite quelli che vennero poi detti appunto *algoritmi di Markov*. Tale indipendenza dalla formulazione utilizzata è ovviamente un forte elemento a favore della Tesi di Church.

Un posto a sé merita l'apporto all'evidenza della Tesi di Church fornito dall'analisi del concetto di calcolo algoritmico compiuta da Turing. Il concetto di *macchina di Turing* si distingue dalla maggior parte degli approcci sopra elencati in quanto non si tratta di un concetto matematico elaborato per ragioni diverse e proposto in un secondo tempo come formulazione rigorosa del concetto di algoritmo, quanto piuttosto di un tentativo diretto di costruire un modello dell'attività di un essere umano che esegue un calcolo di tipo deterministico. Storicamente, fu proprio l'analisi di Turing ad aumentare notevolmente il convincimento della correttezza della Tesi di Church.

Questo tipo di approccio al problema della computabilità effettiva ha condotto alcuni studiosi a considerare la Tesi di Church come una sorta di "legge empirica" piuttosto che come un enunciato a carattere logico-formale". Il logico Emil Post, il quale, nel 1936, propose un concetto di macchina calcolatrice in parte analogo a quello sviluppato da Turing, sottolineava il suo disaccordo da chi tendeva ad identificare la Tesi di Church con un assioma o una mera definizione. Essa dovrebbe piuttosto essere considerata, afferma Post, una ipotesi di lavoro, che, se opportunamente corroborata, dovrebbe assumere il ruolo di una "legge naturale", una "fondamentale scoperta circa le limitazioni del potere matematizzante dell'*Homo sapiens*" (Post 1936, pag. 105).

5. La macchina di Turing universale e il calcolatore di von Neumann

L'interesse delle MT per la teoria delle macchine calcolatrici e per l'informatica risiede innanzi tutto nel fatto che le MT sono un modello del calcolo algoritmico, di un tipo di calcolo quindi che è, in linea di principio, automatizzabile, eseguibile cioè da un dispositivo meccanico. Ogni MT è il modello astratto di un calcolatore - astratto in quanto prescinde da alcuni vincoli di limitatezza cui i calcolatori reali devono sottostare; ad esempio, la memoria di una MT (vale a dire il suo nastro) è potenzialmente estendibile all'infinito (anche se, in ogni fase del calcolo, una MT può sempre utilizzarne solo una porzione finita), mentre un calcolatore reale ha sempre limiti ben definiti di memoria.

Vi sono altre ragioni che giustificano l'analogia tra MT e moderni calcolatori digitali. Sino ad ora abbiamo considerato MT che sono in grado di effettuare un solo tipo di calcolo, sono cioè dotate di un insieme di quintuple che consente loro di calcolare una singola funzione (ad esempio la somma, o il prodotto). Esiste tuttavia la possibilità di definire una MT, detta *Macchina di Turing Universale* (d'ora in poi MTU), che è in grado di simulare il comportamento di ogni altra MT. Ciò è reso possibile dal fatto che

le quintuple di ogni MT possono essere rappresentate in maniera tale da poter essere scritte sul nastro di una MT. Abbiamo accennato al fatto (par. 4) che i numeri naturali possono essere utilizzati per codificare informazioni di tipo discreto di diverso genere. In particolare, è possibile sviluppare un metodo per codificare mediante numeri naturali la tavola di una qualsiasi MT. In questo modo, il codice di una MT può essere scritto sul nastro e dato in input a un'altra MT. Inoltre, tale codifica può essere definita in maniera tale che, dato un codice, si possa ottenere la tavola corrispondente e viceversa mediante un procedimento algoritmico (una codifica che goda di questa proprietà è detta una *codifica effettiva*).

Si può dimostrare che esiste un MT (la MTU appunto) che, preso in input un opportuno codice effettivo delle quintuple di un'altra macchina, ne simula il comportamento. In altre parole, la MTU è una macchina il cui input è composto da due elementi (si veda la parte superiore di fig. 6): 1. la codifica della tavola di una MT (chiamiamola *M*), 2. un input per *M* (chiamiamolo *I*). Per ogni *M* e per ogni *I*, la MTU "decodifica" le quintuple di *M*, e le applica ad *I*, ottenendo lo stesso output che *M* avrebbe ottenuto a partire da *I* (come schematizzato nella parte inferiore di fig. 6).

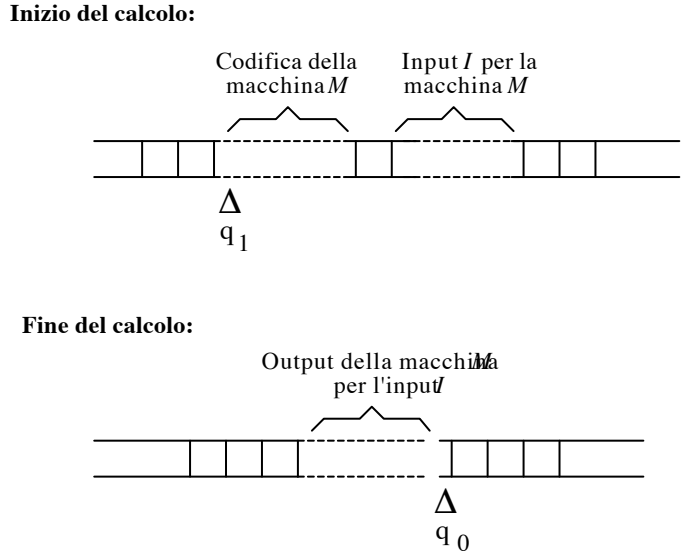

Fig. 6

Poiché la MTU è in grado di simulare il comportamento di qualsiasi MT, allora essa, in virtù della Tesi di Church, è in grado di calcolare qualsiasi funzione che sia calcolabile mediante un algoritmo. Ciò che caratterizza la MTU rispetto alle MT usuali è costituito dal fatto di essere una macchina calcolatrice *programmabile*. Mentre infatti le normali macchine di Turing eseguono un solo programma, che è "incorporato" nella tavola delle loro quintuple, la MTU assume in input il programma che deve eseguire (cioè, la codifica delle quintuple della MT che deve simulare), e le quintuple che compongono la sua tavola hanno esclusivamente la funzione di consentirle di interpretare e di eseguire il programma ricevuto in input.

Un'altra caratteristica fondamentale della MTU è dato dal tipo di trattamento riservato ai programmi. La MTU tratta i programmi (cioè la codifica delle quintuple della MT da simulare) e i dati (l'input della MT da simulare) in maniera sostanzialmente analoga: essi vengono memorizzati sullo stesso supporto (il nastro), rappresentati utilizzando lo stesso alfabeto di simboli ed elaborati in modo simile. Queste caratteristiche sono condivise dagli attuali calcolatori, che presentano la struttura nota come *architettura di von Neumann* (dal nome dello scienziato di origine ungherese John von Neumann che la ideò). La struttura di un calcolatore di von Neumann è raffigurata, molto schematicamente, nella fig. 7. Un dispositivo di input e un dispositivo di output permettono di accedere dall'esterno alla memoria del calcolatore, consentendo, rispettivamente, di inserirvi e di estrarne dei dati. Le informazioni contenute in memoria vengono elaborate

da una singola unità di calcolo (detta CPU - *Central Processing Unit*), che opera sequenzialmente su di essi. La caratteristica più importante della macchina di von Neumann è costituita dal fatto che sia dati che programmi vengono trattati in modo sostanzialmente omogeneo, ed immagazzinati nella stessa unità di memoria. Così, quando un programma deve essere eseguito, l'unità di calcolo lo reperisce in memoria, e lo applica quindi ai dati, anch'essi conservati in memoria. Questo consente una grande flessibilità al sistema. Ad esempio, poiché dati e programmi sono oggetti di natura omogenea, è possibile costruire programmi che prendano in input altri programmi e li elaborino, e che producano programmi in output. Queste possibilità sono ampiamente sfruttate negli attuali calcolatori digitali, e da esse deriva gran parte della loro potenza e della loro facilità d'uso (ad esempio, un compilatore o un sistema operativo sono essenzialmente programmi che operano su altri programmi). In questo senso limitato, un calcolatore di von Neumann costituisce una realizzazione concreta della MTU (e la memoria dati/programmi può essere considerata l'equivalente del nastro della MTU). Anche la potenza computazionale è la stessa, nel senso che, se lo si suppone dotato di una memoria e di tempi di calcolo virtualmente illimitati, un calcolatore di von Neumann è in grado di calcolare tutte le funzioni computabili secondo la Tesi di Church (per questo si dice che una macchina di von Neumann è un *calcolatore universale*). La MTU costituisce quindi un modello astratto degli attuali calcolatori digitali (elaborato prima della loro realizzazione fisica).

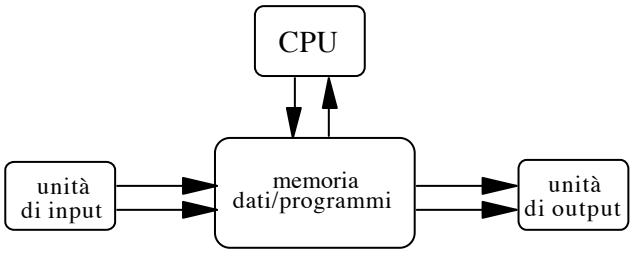

Fig. 7

Si noti che anche i vari linguaggi di programmazione sviluppati in informatica consentono di definire tutte e sole le funzioni ricorsive generali (purché, ovviamente, si supponga che tali linguaggi "girino" su calcolatori ideali con memoria e tempi di calcolo illimitati). Questo vale sia per i linguaggi di programmazione di alto livello (come PASCAL, FORTRAN, BASIC, VISUAL BASIC, C, C++, JAVA, LISP, PROLOG, eccetera), sia per i vari tipi di *codice assembler*. In questo senso, tali linguaggi possono essere considerati analoghi ai vari formalismi citati al punto (b) del par. 4.

6. Il problema della fermata

La tesi di Church ha molte importanti conseguenze dal punto di vista teorico. Dalla sua validità consegue l'esistenza di problemi che non sono risolvibili mediante un algoritmo (come si ricorderà, stabilire se tutti i problemi matematici possono essere in linea di principio risolti con un algoritmo era stata una delle motivazioni principali per lo studio rigoroso del concetto di algoritmo). In particolare, si può dimostrare che non è effettivamente decidibile il *problema della fermata* (*halting problem*) per le MT, cioè il problema di stabilire se, per ogni MT M e per ogni input I, M con input I termina il suo calcolo o meno. Va precisato innanzi tutto che il fatto che la tavola di una MT comprenda almeno una configurazione finale è una condizione necessaria ma non sufficiente perché la macchina termini il calcolo. Si consideri ad esempio la MT seguente:

q_1	s_0	s_0	D	q_1
q_1	|	s_0	C	q_2

La coppia (q_2, s_0) costituisce una configurazione finale; se tuttavia questa macchina viene attivata col nastro completamente vuoto, il suo calcolo andrà avanti all'infinito.

L'indecidibilità del problema della fermata comporta che non esista alcun algoritmo che, data una generica MT (o il suo codice secondo una opportuna codifica effettiva) e dato un generico input per essa, consenta di stabilire se il calcolo di quella macchina con quell'input termina o meno. Diamo qui di seguito una breve traccia intuitiva di come, partendo dalla tesi di Church, si possa giungere a questo risultato ragionando per assurdo. Data una generica macchina M, sia C_M il suo codice in base a una codifica effettiva (scritta nell'alfabeto $\Sigma \equiv \{|\}$). Supponiamo, per assurdo, che il problema della fermata per le MT sia decidibile. Per la tesi di Church, questo comporta che deve esistere una certa macchina di Turing H si comporti nella maniera seguente. Per ogni macchina di Turing M e per ogni input I di M,

$$H \text{ con input } C_M \text{ e } I \begin{cases} \text{dà come output 1 se il calcolo di } M \text{ per l'input } I \text{ termina} \\ \\ \text{dà come output 0 se il calcolo di } M \text{ per l'input } I \text{ non termina.} \end{cases}$$

Qualora esistesse la macchina H, allora sarebbe banale costruire un'altra macchina H' che si comporti come segue:

$$H' \text{ con input } C_M \begin{cases} \text{dà come output 1 se il calcolo di } M \text{ per l'input } C_M \text{ termina} \\ \\ \text{dà come output 0 se il calcolo di } M \text{ per l'input } C_M \text{ non termina.} \end{cases}$$

H' infatti calcola una funzione che è un "caso particolare" della funzione calcolata da H (in quanto C_M è un valore particolare di I). Se tuttavia esistesse H', allora si potrebbe a sua volta costruire una macchina Z così definita:

$$Z \text{ con input } C_M \begin{cases} \text{genera un calcolo che non termina se } H' \text{ con input } C_M \text{ dà come output 1} \\ \text{(cioè, se il calcolo di } M \text{ per l'input } C_M \text{ termina)} \\ \\ \text{dà come output 0 se } H' \text{ con input } C_M \text{ dà come output 0} \\ \text{(cioè, se il calcolo di } M \text{ per l'input } C_M \text{ non termina).} \end{cases}$$

Per ottenere Z a partire da H' sarebbe sufficiente aggiungere alla tavola di H' alcune quintuple che facciano in modo che, se l'output di H' è 1, allora abbia origine un calcolo che non termina (ad esempio, la testina potrebbe iniziare a spostarsi a destra sul nastro qualunque sia il simbolo osservato).

Ora, si immagini di dare in input a Z il suo stesso codice C_Z. E' facile constatare che, in base alla definizione di Z, Z con input C_Z darebbe origine a un calcolo che termina se e soltanto se il calcolo di Z per l'input C_Z non termina, il che è palesemente assurdo. Ne consegue quindi che una macchina che si comporti come H non può esistere, e che quindi, se è vera la tesi di Church, non può esistere un algoritmo che decida il problema della fermata.

Da questo risultato consegue che esistono problemi i quali, neppure in linea di principio, possono essere risolti da un calcolatore. Ad esempio, non può esistere alcun programma che sia grado di stabilire in generale se un programma qualsiasi con un certo input terminerà il suo calcolo o meno. E' importante ricordare che l'indecidibilità del problema della fermata è strettamente collegata ai risultati di limitazione della logica matematica, in primo luogo i teoremi di Gödel.

7. Le macchine di Turing e la mente: il test di Turing

Abbiamo accennato alla tendenza ad interpretare la Tesi di Church come un'ipotesi empirica sulle capacità computazionali degli esseri umani. Su questa linea procedono alcuni sviluppi successivi del pensiero dello stesso Turing. Nel suo saggio "Macchine calcolatrici ed intelligenza" (Turing 1950) assistiamo ad una sorta di "radicalizzazione" di questo modo di intendere la Tesi di Church. Facendo riferimento a calcolatori reali, che tuttavia vengono caratterizzati in maniera analoga a macchine di Turing, Turing si dichiara fiducioso che macchine di questo tipo possano giungere a simulare, nel volgere di pochi decenni, non soltanto il "comportamento computazionale" di un essere umano, ma anche qualsiasi altra attività cognitiva umana. Turing propone di riformulare la domanda "possono pensare le macchine?" nei termini del cosiddetto *gioco dell'imitazione*. Il gioco viene giocato da tre "attori": a) un essere umano, b) una macchina calcolatrice e c) un altro essere umano, l'interrogante. L'interrogante non può vedere a) e b), non sa chi dei due sia l'essere umano, e può comunicare con loro solo in maniera indiretta (ad esempio, attraverso un terminale video e una tastiera). L'interrogante deve sottoporre ad a) e a b) delle domande, in maniera tale da scoprire, nel più breve tempo possibile, quale dei due sia l'uomo e quale la macchina. a) si comporterà in modo da agevolare c), mentre b) dovrà rispondere in modo da ingannare c) il più a lungo possibile. Invece di chiedersi se le macchine possono pensare, dice Turing, è più corretto chiedersi se una macchina possa battere un uomo nel gioco dell'imitazione, o, comunque, quanto a lungo possa resistergli. Questo "esperimento mentale" viene oggi abitualmente indicato col nome di *Test di Turing*.

Turing era decisamente troppo ottimista circa le possibili prestazioni delle macchine calcolatrici: "Credo che entro circa 50 anni sarà possibile programmare calcolatori ... per far giocare loro il gioco dell'imitazione così bene che un esaminatore medio non avrà più del 70 per cento di probabilità di compiere l'identificazione esatta dopo cinque minuti di interrogazione. Credo che la domanda iniziale, 'possono pensare le macchine?', sia troppo priva di senso per meritare una discussione. Ciò nonostante credo che alla fine del secolo l'uso delle parole e l'opinione corrente si saranno talmente mutate che chiunque potrà parlare di macchine pensanti senza aspettarsi di essere contraddetto". Ci troviamo qui di fronte ad una sorta di versione "estremista", o "radicale", della Tesi di Church, che, grosso modo, potrebbe essere formulata come segue: *ogni attività cognitiva è T-computabile* (il che non vuol dire, ovviamente, che la nostra mente funziona *come* una macchina di Turing, ma che ogni attività mentale è simulabile da un dispositivo che abbia la stessa potenza computazionale delle macchine di Turing). Seppure modificata e raffinata rispetto alla formulazione di Turing, una assunzione di questo genere è a fondamento di numerose teorie e ricerche svolte nell'ambito di quel settore di ricerca che va sotto il nome di *scienze cognitive*. Si tratta di un ambito di ricerca interdisciplinare che ha per oggetto lo studio della mente, e che raccoglie i contributi di diverse discipline quali la psicologia cognitiva, la linguistica, la filosofia, l'informatica e le neuroscienze. Ciò che accomuna le ricerche svolte nelle scienze cognitive è appunto l'ipotesi che gli strumenti di tipo computazionale possano essere in qualche misura adeguati come modelli per lo studio delle facoltà mentali. In particolare, tra le scienze cognitive, l'*intelligenza artificiale* è quel settore dell'informatica che si prefigge di elaborare programmi di calcolatore che simulino specifiche attività cognitive umane, sia allo scopo di meglio comprendere queste ultime, sia allo scopo di costruire manufatti tecnologicamente rilevanti.

8. Oltre von Neumann: reti neurali e calcolo parallelo

Nel corso della storia dell'informatica, sono stati proposti vari modelli alternativi al calcolatore di von Neumann. Infatti, benché siano estremamente versatili, alle macchine con architettura di von Neumann sono stati imputati dei limiti dal punto di vista informatico. In particolare, è stata criticata la netta separazione tra immagazzinamento ed elaborazione dei dati che questo tipo di architettura comporta. In un calcolatore di von Neumann memoria e unità centrale di calcolo (CPU) sono due componenti rigidamente distinte. L'unità

di calcolo attinge di volta in volta ai dati contenuti nella memoria, ma quest'ultima rimane sostanzialmente passiva durante la maggior parte della durata del calcolo. Si tratta del cosiddetto problema del "collo di bottiglia" dell'architettura di von Neumann: le informazioni vengono elaborate solo quando vengono richiamate dalla CPU del sistema. Ciò comporta problemi di efficienza nello sfruttamento delle risorse computazionali.

Queste limitazioni hanno un corrispettivo anche dal punto di vista dello studio computazionale della mente. Una distinzione netta tra memorizzazione delle informazioni e loro elaborazione è difficilmente giustificabile sulla base delle conoscenze disponibili sul sistema nervoso. Nel cervello non esiste alcun dispositivo centralizzato per il controllo dell'elaborazione. Le operazioni computazionali nel sistema nervoso sembrano demandate ad un meccanismo di controllo altamente distribuito. Inoltre, non esiste una separazione netta tra dispositivi per la memorizzazione e per l'elaborazione delle informazioni. Ciò pone problemi ai modelli computazionali dell'intelligenza artificiale e delle scienze cognitive tradizionali, relativi alla mancanza di plausibilità dal punto di vista anatomico e neurofisiologico del paradigma computazionale di Turing e di von Neumann. Il punto centrale è che il cervello è un dispositivo di calcolo *altamente parallelo*. Il numero dei neuroni è di un ordine stimabile tra 10^{10} e 10^{11}, e ciascuno di essi si comporta come una singola unità di calcolo, che lavora contemporaneamente a tutte le altre. I neuroni sono altamente interconnessi: ogni neurone ha moltissime sinapsi in entrata e in uscita, mediante le quali scambia i propri input e i propri output con gli altri neuroni. Ogni neurone esegue operazioni relativamente semplici. La complessità dei meccanismi cognitivi viene determinata dall'interazione di un grande numero di neuroni.

Su considerazioni di questo genere si è basato lo sviluppo delle cosiddette *reti neurali*, una classe di dispositivi di calcolo in parte motivati dall'intento di superare i limiti del modello di von Neumann. Si tratta di sistemi distribuiti ad alto parallelismo, ispirati, in senso lato, alle proprietà del sistema nervoso. Una rete neurale è costituita da un insieme di *unità* (che sono il corrispettivo dei neuroni), collegate tra loro da *connessioni*, che costituiscono l'analogo delle sinapsi. Ogni unità ha un certo numero di connessioni in ingresso e/o un certo numero di connessioni in uscita. Ciascuna unità costituisce un semplice processore, un singolo dispositivo di calcolo che, ad ogni fase del calcolo, riceve i propri input attraverso le connessioni in ingresso, li elabora, e invia l'output alle altre unità connesse per mezzo delle sue connessioni in uscita. In una rete tutte le unità operano in parallelo, e non esiste alcun processo di ordine "più alto", nessuna CPU che ne coordini l'attività. Il calcolo che ciascuna unità esegue è di norma molto semplice; la potenza computazionale del sistema deriva dal grande numero delle unità e delle connessioni. Nell'ambito delle scienze cognitive, sulle reti neurali si basano le teorie e i modelli di tipo connessionista. Il *connessionismo* è una tendenza nello studio computazionale della mente che ha avuto un grande sviluppo nel corso degli ultimi quindici anni, e che si è in parte contrapposta agli approcci dell'intelligenza artificiale e delle scienze cognitive tradizionali. Rispetto a queste ultime, il connessionismo è caratterizzato appunto da una maggiore attenzione per i rapporti tra attività cognitive e struttura del sistema nervoso.

E' opportuno notare tuttavia che questi sviluppi non hanno comportato un superamento dei risultati della teoria della computabilità effettiva, o una qualche forma di "falsificazione" della tesi di Church. Di fatto, tutti i modelli computazionali basati sulle reti neurali che siano stati effettivamente realizzati sono risultati riconducibili entro i limiti della ricorsività generale, nel senso che le funzioni computate da tali modelli risultano essere funzioni ricorsive generali.

Più in generale, benché le MT e i calcolatori con architettura di von Neumann siano dispositivi di calcolo di tipo strettamente sequenziale, è possibile estendere la validità della tesi di Church anche a calcoli di tipo parallelo. Rilevanti in questa direzione sono state ad esempio le ricerche di Robin Gandy (1980), che è partito dalla constatazione che il concetto di MT corrisponde ad una nozione di calcolo troppo specifica e particolare perché le possa essere ricondotto ogni tipo di dispositivo di calcolo concepibile. Ad esempio, nelle MT si assume che il calcolo proceda secondo una sequenza di passi elementari, elaborando un solo simbolo alla volta, mentre un calcolatore artificiale può procedere in parallelo, elaborando

contemporaneamente un numero arbitrario di simboli. Per superare tali limitazioni, Gandy ha formulato, utilizzando strumenti di tipo insiemistico, una caratterizzazione estremamente generale del concetto di macchina calcolatrice, in cui le MT rientrano come caso particolare. Egli ha dimostrato quindi che ogni funzione calcolabile da tali macchine è ricorsiva generale, a patto che vengano rispettate alcune condizioni molto generali di finitezza (determinismo, possibilità di descrivere il calcolo in termini discreti, e così via).

Va ricordato tuttavia che, dal punto di vista applicativo, l'architettura di von Neumann (o sue varianti che non ne differiscono in maniera sostanziale) resta il modello di calcolatore di gran lunga più diffuso. Attualmente calcolatori con architettura non di von Neumann vengono progettati e utilizzati per tipi di applicazioni specifiche.

Ulteriori letture

Per chi volesse affrontare gli aspetti tecnici della teoria della computabilità, alcune trattazioni approfondite sono (Kleene 1952 [parte III], Hermes 1961; Rogers 1967; Minsky 1967; Lewis e Papadimitriou 1981; Davis e Weyuker 1983; Odifreddi 1989). (Davis 1965) è una raccolta di articoli storici sulla teoria della computabilità. Una raccolta di articoli classici di logica e filosofia della matematica, con molti punti di contatto con i temi qui trattati è (van Heijenhoort 1967). Sui problemi di filosofia della matematica che hanno portato alla formulazione della teoria della computabilità si veda (Borga e Palladino 1997). (Turing 1994) è una raccolta di scritti di Alan Turing. Una biografia di Turing è stata scritta da Andrew Hodges (1983). L'articolo dove viene proposto il test di Turing è compreso in (Somenzi e Cordeschi 1994). Infine, un libro che tratta in modo affascinante e molto particolare i temi della computabilità e dei teoremi di limitazione della logica in relazione alla teoria della mente è (Hofstadter 1979).

Bibliografia

Borga, M. e Palladino, D. (1997). *Oltre il mito della crisi. Fondamenti e filosofia della matematica nel ventesimo secolo*. La Scuola, Brescia.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345-363.

Curry, H.B. (1929). An analysis of logical substitution. *American Journal of Mathematics*, 51: 363-384.

Curry, H.B. (1930). Grundlagen der Kombinatorischen Logik. *American Journal of Mathematics*, 52:509-536, 789-834.

Curry, H.B. (1932). Some additions to the theory of combinators. *American Journal of Mathematics*, 54:551-558.

Davis, M. (a cura di) (1965). *The Undecidable*. Raven Press, Hewlett, New york.

Davis, M. e Weyuker, E. (1983). *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. Academic Press, New York.

Gandy, R. (1980). Church's thesis and principles for mechanisms. In *The Kleene Symposium*, 123-148, North Holland, Amsterdam.

van Heijenhoort, J. (a cura di) (1967). *From Frege to Gödel*. Harvard University Press, Harvard.

Hermes, H. (1961). *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit*. Springer, Berlin, Heidelberg. Tr. it. *Numerabilità, decidibilità, computabilità*, Boringhieri, Torino, 1973.

Hodges, A. (1983). *Alan Turing: The Enigma*. New York, Simon and Shuster. Tr. it. *Storia di un enigma. Vita di Alan Turing (1912-1954)*. Bollati, Torino, 1991.

Hofstadter, D. (1979).*Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books. Tr. it. *Gödel, Escher, Bach: un'eterna ghirlanda brillante*, Adelphi, Milano, 1984.

Kleene, S.C. (1952). *Introduction to metamathematics*. North Holland, Amsterdam.

Lewis, H.R. e Papadimitriou, C.H. (1981). *Elements of the Theory of Computation*. Prentice-Hall, Englewood Cliffs, NJ.

Markov, A.A. (1951). Theory of algorithms. *American Mathematical Society Translations*, seconda serie, 15(1960):1-14 (trad. ingl. dell'originale russo).

Markov, A.A. (1954). *Theory of algorithms*. National Science Foundation and Israel Program for Scientific Translation (1961) (trad. ingl. dell'originale russo).

Minsky, M. (1967). *Computation. Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs.

Odifreddi, P. (1989). *Classical Recursion Theory : The Theory of Functions and Sets of Natural Numbers*. Elsevier, Amsterdam.

Post, E. (1936). Finite combinatory processes - formulation 1. *Journal of Symbolic Logic*, 1:103-105.

Post, E. (1943). Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197-215.

Post, E. (1946). A variant of a recursively unsolvable problem. *Bullettin of the American Mathematical Society*, 52:284-316.

Rogers H. (1967). *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York. Tr. it. *Teoria delle funzioni ricorsive e della computabilità effettiva*, Tecniche Nuove, Milano, 1992.

Schönfinkel, M. (1924). Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305-316.

Somenzi, V. e Cordeschi, R. (a cura di) (1994). *La filosofia degli automi. Origini dell'intelligenza artificiale*. Bollati, Torino.

Turing, A. (1936-7). On computable number, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, serie 2, 42:230-365; A correction, ibid., 43:544-546 (ristampato in Davis 1965).

Turing, A. (1937). Computability and λ-definability. *Journal of Symbolic Logic*, 2:153-163.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59:433-460.

Turing, A. (1994). *Intelligenza meccanica*. A cura di G. Lolli, Bollati, Torino.