La Matematica nella Società e nella Cultura
Rivista dell'Unione Matematica Italiana
Serie I, Vol. V, Agosto 2012, 105-148

Turing: la vita, l'opera, l'impatto

Carlo A. Furia - Dino Mandrioli

In occasione del centenario della nascita di Alan Turing ben volentieri abbiamo aderito alla richiesta della redazione di partecipare alle numerose altre celebrazioni, raramente tuttavia destinate al mondo accademico italiano.

Siamo entrambi ingegneri informatici e da sempre attribuiamo grande valore agli aspetti teorici e fondanti della nostra disciplina; ci risulta perciò del tutto naturale riconoscere alla figura di Turing, certo uno tra non pochi che nell'arco dei millenni – ma soprattutto nella prima metà del secolo scorso – hanno costruito le fondamenta teoriche dell'informatica, un ruolo assolutamente centrale – oseremmo dire dominante – nella storia di questa disciplina. Attribuiamo questo merito soprattutto all'innata capacità di coniugare la speculazione filosofica con il rigore e l'approfondimento matematico ma anche con la sintesi e l'applicazione a problemi pratici tipiche della mentalità ingegneristica; dote riscontrabile in pochissimi altri geni della nostra umanità e che potremmo definire "leonardesca".

Ben a ragione dunque, il più prestigioso premio internazionale dell'informatica, annualmente assegnato a chi abbia prodotto contributi scientifici di massimo valore e impatto e unanimemente riconosciuto come il Premio Nobel dell'informatica, è intitolato ad Alan Turing.

Tra i tanti modi per rendere omaggio alla figura di Turing abbiamo cercato un'integrazione degli aspetti più significativi della sua intensissima sia pur relativamente breve biografia con la presentazione, congiunta a una limitata esegesi, del suo contributo scientifico probabilmente di maggior rilievo ed impatto, oltre che forse il più noto. La prossima sezione, dunque, descrive le tappe fondamentali della sua vita mettendole in relazione con i risultati scientifici, e non solo, che

Date fondamentali nella vita di Turing.

1912:

Nasce a Londra, figlio di un funzionario dell'Indian Civil Service.

1912–1930:

Trascorre infanzia e adolescenza in Inghilterra; studia con discreto profitto presso una Public School (cioè scuola privata) e coltiva privatamente interessi scientifici.

1931–1934:

Studente di matematica a Cambridge; viene a conoscenza dei lavori fondamentali di logica matematica di Russell e Whitehead e di Gödel.

1935–1936:

Pubblica "On computable numbers, with an application to the *Entscheidungsproblem*" [Tur37], quadro teorico fondamentale all'informatica moderna.

1936–1938:

Consegue un PhD a Princeton sotto la supervisione di Alonzo Church [Tur39]; conosce von Neumann.

1939–1944:

Lavora a Bletchley Park, sede del quartier generale del dipartimento di crittanalisi, alla decrittazione dei messaggi codificati con la macchina tedesca "Enigma".

1945–1947:

Lancia il progetto ACE (Automatic Computing Engine) [Tur45] di costruzione di un calcolatore elettronico, finanziato dal National Physical Laboratory inglese.

1948–1949:

Supervisiona la realizzazione del calcolatore Mark 1 a Manchester. Pubblica "Rounding-off errors in matrix processes" [Tur48], fondazione dell'analisi numerica moderna, e alcune idee pionieristiche sulla verifica di correttezza di software.

1950–1951:

Pubblica "Computing machinery and intelligence" [Tur50b], che introduce il "test di Turing" per valutare l'intelligenza di macchine calcolatrici, e "The chemical basis of morphogenesis" [Tur52], che sviluppa uno dei primissimi modelli matematici di fenomeni biologici.

1952–1954:

Viene condannato a castrazione chimica per una relazione omosessuale; muore suicida a 42 anni.

Turing ha via via ottenuto; la sezione successiva presenta il principale risultato relativo alle potenzialità e limiti del calcolo automatico; nella sua costruzione abbiamo cercato di coniugare gli aspetti più specifici delle tecniche matematiche utilizzate con i risvolti a carattere più filosofico e con le relazioni con i contributi principali di altri scienziati e filosofi precedenti e contemporanei a Turing.

Infine, l'ultima sezione riprende, sempre guidata dal pensiero e dalle intuizioni di Turing, gli aspetti più squisitamente filosofici della teoria della computazione, affrontando il sempre vivo e presente interrogativo sulle relazioni tra calcolo automatico, ragionamento, intelligenza. Anche in questo caso, il contributo di Turing, con la sua consueta lucidità, può essere visto come l'anello di congiunzione tra gli approcci filosofici tradizionali, la loro più recente "revisione nel linguaggio matematico" (con i contributi di Cantor, Russell, Gödel, e altri) e la moderna intelligenza artificiale, con tutte le accezioni anche di immediato impatto pratico che essa ha assunto ai giorni nostri. È doveroso tuttavia sottolineare che gli aspetti trattati in questa sezione sono per loro natura condizionati da convincimenti soggettivi, spesso neanche del tutto consapevoli, di chi li tratta, nonché dalla sua cultura, che, nel caso degli autori di queste note, è inevitabilmente quella del mondo occidentale.

Biografia

Con la fine della Seconda Guerra Mondiale si chiude uno dei periodi più bui e tremendi della storia moderna. Sulle macerie dell'Europa ancora fumanti, il mondo volta pagina aprendo una lunga stagione di progresso scientifico e tecnologico senza precedenti per rapidità ed estensione. Al centro di questo progresso c'è la rivoluzione dell'informatica: la "scienza dei calcolatori", che mostra le prime promettenti applicazioni pratiche proprio in questi anni, e la cui spinta propulsiva arriverà fino all'epoca contemporanea, ben lungi dall'esaurirsi. In questo momento storico di svolta incipiente, c'è al mondo una persona che, con una combinazione unica di competenze eclettiche e genio straordinario, è in grado di iniziare questa rivoluzione, pas-

sando alla storia come autorevole padre dell'informatica moderna: Alan Mathison Turing.

Nel 1945 il nome di Alan Turing è sconosciuto al grande pubblico, al di fuori della ristretta comunità accademica di matematici di cui fa parte, che ha avuto modo di apprezzare i suoi lavori di logica matematica negli anni trenta. Quasi nessuno dei suoi colleghi matematici è al corrente delle attività di Turing durante la guerra, quando sospende temporaneamente i propri incarichi accademici per lavorare come dipendente del dipartimento di crittanalisi del governo britannico, e si trasferisce nel quartier generale di Bletchley Park, località a 80 chilometri a nord-ovest di Londra. L'obiettivo principale del lavoro a Bletchley Park è quello di decifrare i messaggi codificati con la macchina "Enigma", il cifrario usato dall'esercito tedesco per le comunicazioni strategiche alle unità in guerra. Il lavoro di decrittazione parte da alcune idee di analisti polacchi, che erano riusciti a decodificare un sottoinsieme ristretto dei messaggi codificati con l'Enigma e avevano sviluppato una macchina elettromeccanica, soprannominata "Bomba", di supporto all'analisi combinatorica richiesta dalla decrittazione. A Bletchley Park, questo lavoro di base viene ampliato significativamente, sia nell'aspetto di analisi matematica sia per quanto riguarda l'implementazione delle "Bombe", con l'obiettivo di arrivare a decodificare sistematicamente e in tempo utile i messaggi intercettati dei vertici dell'esercito tedesco codificati con l'Enigma. Turing diventa ben presto una persona chiave nei laboratori di Bletchley Park, dove guida sia la parte teorica del lavoro di analisi combinatorica, sia il progetto e l'implementazione di nuove versioni delle "Bombe" più potenti e veloci. In una alternanza tumultuosa di successi di decrittazione e contromisure tedesche, il contributo del lavoro di Bletchley Park per aumentare il vantaggio delle forze alleate e, in ultima analisi, vincere la guerra, è fondamentale. Anche se, per motivi di segretezza, i dettagli e l'importanza di questo lavoro saranno divulgati solo molti anni più tardi, Turing è unanimemente riconosciuto da chi ha lavorato con lui durante la guerra come un genio eclettico e un po' eccentrico, il cui ruolo è di primissimo piano per quantità e qualità. Dalla prospettiva della nascita dell'informatica moderna, il lavoro di realizzazione delle "Bombe" a Bletchley Park ha fatto comprendere a Turing due aspetti

fondamentali: i vantaggi, in termini di velocità e affidabilità, dell'uso di tecnologie elettroniche invece che elettromeccaniche; e il vantaggio strategico di progettare macchine calcolatrici generiche, indipendenti dal problema concreto al quale sono applicate, con un alto grado di riutilizzabilità e scalabilità su istanze differenti.

Nei suoi lavori di logica matematica degli anni trenta, che descriviamo nel dettaglio nella prossima sezione, Turing aveva già studiato approfonditamente in un contesto astratto il secondo problema, della "genericità" ovvero "universalità" delle macchine calcolatrici. Ma come era giunto un figlio della borghesia inglese, senza parenti o antenati nel mondo accademico e scientifico, ad occuparsi di problemi astratti di logica matematica, proprio negli anni che vedono risultati fondamentali in questa disciplina ad opera di un ristretto numero di figure straordinarie?

Alan Turing nasce nel 1912 nel quartiere di Paddington, a Londra, figlio di Julius – funzionario statale presso l'Indian Civil Service a Madras (Chennai) – e Ethel – di famiglia anglo-irlandese che pure lavora per l'impero britannico. Pur essendo a tutti gli effetti un "figlio dell'impero britannico", trascorre infanzia e adolescenza in Inghilterra insieme al fratello maggiore John, mentre i genitori rimangono in India per lunghi periodi ogni anno. Mentre percorre con profitto presso una "Public School" ([1]) un curriculum tradizionale incentrato sulle materie umanistiche, coltiva privatamente spiccati interessi scientifici (argomenti che vengono considerati di poco valore dal sistema scolastico che frequenta).

Nel 1931 inizia gli studi universitari a Cambridge come studente del King's College, dove trova finalmente un ambiente aperto che incoraggia i suoi interessi culturali. Inizialmente focalizza i propri studi sulla matematica classica, con un interesse particolare per la teoria della probabilità e la nascente meccanica quantistica. Tra i primi risultati degni di nota si ricorda una dimostrazione originale del teorema del limite centrale [Tur34] – che asserisce che la media campionaria di

([1]) Nel sistema scolastico inglese, il termine Public School indica un insieme ristretto di scuole indipendenti qualificate ed esclusive, private ma aperte a chiunque passi gli esami di ammissione e paghi la retta.

un grande numero di variabili casuali converge a una distribuzione normale. Pur trattandosi di un risultato minore nella produzione di Turing, essendo una rielaborazione di un risultato classico, il tema affrontato è indicativo della sua capacità di connessione tra modelli matematici astratti e processi fisici concreti: il teorema ha anche una valenza empirica nel giustificare come fenomeni complessi, risultanti dall'interazione di molte variabili indipendenti, possono essere modellizzati con buona approssimazione da un modello normale, che infatti è ubiquo nel mondo naturale.

Durante gli studi a Cambridge, Turing entra ben presto a contatto con il mondo in fermento della logica matematica, e in particolare ha modo di studiare i *Principia Mathematica* [WR09] di Russel e Whitehead (la cui seconda edizione esce nel 1925), e i teoremi di incompletezza di Gödel (1931, [Göd31]). Durante una lezione del topologo Newman nel 1935, Turing viene a conoscenza dell'*Entscheidungsproblem* (Problema della decisione) formulato da Hilbert nel 1928 come parte fondamentale del suo ambizioso programma di formalizzazione e meccanizzazione della matematica con mezzi "finitisti". Nel giro di un anno, Turing elabora una soluzione brillante che mostra che l'*Entscheidungsproblem* non è risolvibile, ossia non esiste una procedura automatica che possa decidere la validità di una generica formula logica: la dimostrazione di teoremi non può essere completamente automatizzata. Il risultato di Turing appare nei *Proceedings* della *London Mathematical Society* in un articolo dal titolo "On computable numbers, with an application to the *Entscheidungsproblem*" [Tur37]. Senza saperlo, Turing lavora a questo problema e lo risolve in parallelo con Alonzo Church, che anzi pubblica il suo risultato circa un anno in anticipo [Chu36]: anche se il lavoro di Turing è completato nel 1936 apparirà in stampa ufficialmente solo un anno dopo. Ciononostante, la portata storica del contributo di Turing è significativamente più importante del lavoro di Church, e l'articolo "On computable numbers" è unanimenente considerato come un atto di fondazione dell'informatica moderna. Come discuteremo nel dettaglio nella prossima sezione, uno dei contributi fondamentali dell'articolo del 1936 è la nozione di *macchina universale*, capace di emulare il funzionamento di ogni altra

macchina calcolatrice passatale, opportunamente codificata, come input. Nove anni più tardi, a valle dell'esperienza di Bletchley Park, Turing capirà per primo come la tecnologia fosse matura per realizzare concretamente una macchina calcolatrice universale secondo i principi da lui enunciati nel 1936. Questa idea sarà alla base dello sviluppo di calcolatori con programma in memoria, una componente fondamentale dell'architettura, resa popolare da von Neumann, incorporata in ogni calcolatore moderno.

Nel settembre del 1936, in contemporanea alla diffusione dei propri lavori di logica matematica, Turing si trasferisce a Princeton per studiare sotto la supervisione di Alonzo Church. Mentre a Cambridge aveva lavorato in sostanziale isolamento, a Princeton Turing entra in contatto con il gotha della logica matematica. Von Neumann, in particolare, è a Princeton in questi stessi anni, e mostra interesse per i lavori di Turing sulla computabilità. Nel 1938 Turing consegue il titolo di PhD con una tesi di dottorato su un tema tecnico di logica matematica sui numeri ordinali [Tur39], ancora in collegamento con i risultati di incompletezza di Gödel. La tesi introduce anche la nozione di "oracolo" e la tecnica di dimostrazione per riduzione a oracolo, tecnica tuttora usata estensivamente per dimostrare la non decidibilità di un problema e nell'analisi di complessità di problemi computazionali. Al termine del soggiorno a Princeton, Turing declina l'offerta di von Neumann di una posizione temporanea e ritorna a Cambridge.

L'esperienza di Princeton non solo ha contribuito all'ampliamento di un bagaglio teorico di raffinate tecniche matematiche, ma include anche delle esperienze pratiche nel campo della crittanalisi applicata come l'invenzione e la costruzione di congegni elettromeccanici artigianali di supporto all'analisi combinatorica, sviluppati perlopiù come hobby. Sull'onda di un interesse crescente, e con la minaccia sempre più concreta costituita dalla Germania nazista, Turing inizia una collaborazione con il dipartimento governativo di crittanalisi, che porterà all'esperienza di Bletchley Park di cui abbiamo già detto. Queste esperienze eterogenee si combinano in modo che, alla fine della guerra, Turing è in possesso di una combinazione di competenze pressoché unica: i suoi lavori teorici degli anni trenta – e in particolare la nozione di macchina universale – sottostanno alla nozione di calcolatore uni-

versale programmabile; e la sua esperienza pratica con le "Bombe" a Bletchley Park lo ha reso familiare con gli aspetti più ingegneristici della costruzione di macchine calcolatrici.

Mostrando ancora una volta una straordinaria capacità di sintesi di idee disparate, Turing matura l'idea precisa di costruire un "cervello" artificiale e convince il National Physical Laboratory (NPL) a lanciare il progetto ACE: Automatic Computing Engine. L'obiettivo del progetto è lo sviluppo delle idee fondamentali e la realizzazione concreta di un calcolatore elettronico. Lo sviluppo dell'ACE non potrà contare su una logistica altrettanto efficiente (nè su finanziamenti altrettanto generosi) di quella di analoghi progetti americani; come risultato, l'EDVAC di von Neumann rilascerà le prime pubblicazioni nel 1945, in anticipo rispetto a Turing, da cui pure aveva preso in prestito molte idee e intuizioni. Nonostante questi ostacoli imprevisti, Turing rimane al vertice degli sviluppi dell'informatica, e riesce a contribuire svariate idee progettuali innovative, con un'influenza evidente su progetti analoghi (l'EDVAC in primis) che includeranno idee paragonabili solo anni più tardi. In particolare, il progetto ACE [Tur45] sviluppa uno dei primi linguaggi di programmazione dotati del meccanismo del sottoprogramma, e prevede un'architettura incentrata sul software per semplificare e riutilizzare il più possibile la progettazione di componenti hardware fondamentali.

L'inefficiente organizzazione del NPL, che ha significativamente cambiato priorità e flessibilità col termine della guerra, ostacola lo sviluppo dell'ACE e motivi di presunta segretezza ritardano la pubblicazione delle idee fondamentali partorite dal progetto. Max Newman, un collega di Turing a Bletchley Park diventato nel frattempo "chair" all'Università di Manchester, ottiene un altro finanziamento per la costruzione di una macchina calcolatrice e convince Turing a supervisionare il progetto a Manchester. Nel 1948 entra finalmente in funzione il Mark 1, un prototipo di calcolatore costruito secondo le idee di Turing.

Nello stesso anno Turing pubblica altri due articoli pionieristici, sempre caratterizzati da una sintesi formidabile tra idee teoriche e pratiche. Il primo tratta dei "Rounding-off errors in matrix processes" [Tur45], e costituisce la fondazione dell'analisi numerica moderna,

introducendo nozioni fondamentali come quella di accumulazione dell'errore e considerazioni di efficienza e complessità. L'illustre analista numerico James Wilkinson, che ha lavorato con Turing al progetto ACE, riconoscerà pubblicamente l'enorme influenza di questo articolo durante il suo discorso di accettazione del Premio Turing nel 1970 – attribuitogli proprio per i suoi lavori fondamentali di analisi numerica. Il secondo contributo del 1948 è un rapporto tecnico [Tur50a] con alcune idee pionieristiche sul "program checking", ovvero sull'uso della logica matematica per verificare la correttezza di programmi per calcolatori. Anche in questo caso, la portata del contributo verrà apprezzata appieno solo diversi anni più tardi, quando Floyd, Hoare, e Dijkstra negli anni sessanta daranno una fondazione completa alla nozione di verifica formale del software.

Negli anni seguenti, Turing torna ad occuparsi di lavori più spiccatamente teorici, ma sempre guidato dall'ambizione fondamentale di caratterizzare le possibilità e i limiti ultimi della computazione. Nel 1950, pubblica un articolo filosofico dal titolo "Computing machinery and intelligence" [Tur50b], che affronta con rigore e lucidità la domanda "Le macchine sono capaci di pensare?" Ancora una volta, Turing dimostra una capacità eccezionale di sintesi di nozioni informali in una forma rigorosa che le trasforma in domande definite, suscettibili di studio scientifico. L'articolo è passato alla storia per la presentazione del cosiddetto "test di Turing", che è una procedura sperimentale per stabilire il livello di "intelligenza" di una macchina capace di conversazione in linguaggio naturale. L'idea di base del test di Turing (che descriviamo meglio nell'ultima sezione), chiamato "gioco dell'imitazione" nell'articolo del 1950, è di dare una nozione operativa di intelligenza basata sugli stessi criteri coi quali attribuiamo coscienza ad un'altra persona, cioè basandosi sulla capacità di intrattenere una conversazione spontanea e credibile. Nonostante le molte analisi critiche di cui l'idea di Turing è stata oggetto, essa appare ancora come un'idea solida e un'importante base per trasportare l'idea di intelligenza e coscienza dal dominio del discorso puramente filosofico a quello dei problemi trattabili col metodo scientifico. Mostrando una dualità che Turing avrebbe sicuramente apprezzato, l'idea del test di Turing ha anche motivato applicazioni pratiche come i CAPTCHA

(Completely Automated Public Turing Test to tell Computers and Humans Apart) che implementano una versione ristretta del gioco dell'imitazione per impedire l'accesso a spam-bot e consentirlo a utenti umani.

Un altro lavoro importante di questi anni è un articolo dal titolo "The chemical basis of morphogenesis", che sviluppa alcune nozioni innovative di teoria dei sistemi non lineari, e uno dei primissimi modelli matematici di fenomeni biologici. Nello stesso anno 1951, Turing viene eletto membro della Royal Society per i suoi lavori sulla computabilità.

Questa progressione impressionante di risultati scientifici straordinari giungerà ben presto a una brusca e drammatica fine. Per tutta la vita, Turing non ha mai tenuto segreta la propria omosessualità, ma mentre l'ambiente liberale e illuminato di Cambridge non lo aveva mai discriminato, l'atmosfera di Manchester era molto meno tollerante. L'omosessualità era ancora reato nell'Inghilterra degli anni cinquanta, e Turing viene arrestato nel 1952 per una sua relazione con un giovane di Manchester. Messo di fronte all'alternativa tra il carcere e la castrazione chimica, sceglie la seconda per rimanere in libertà e continuare il proprio lavoro. Gli effetti della somministrazione di estrogeni si rivelano però molto onerosi e contribuiscono in maniera determinante ad un periodo di progressivo isolamento e pessimismo. L'8 giugno 1954, poco tempo prima del suo quarantaduesimo compleanno, viene trovato morto nella propria abitazione, suicidatosi ingerendo una mela avvelenata con cianuro. Solo nel 2009, in risposta a una petizione raccolta su Internet, il governo britannico formulerà le proprie scuse ufficiali per il trattamento omofobico che ha condannato Turing a questa fine tragica e prematura. Pochi altri personaggi nella storia dell'umanità hanno avuto un così importante impatto durante una così breve vita.

Algoritmi e macchine: La teoria della computabilità

Tra i numerosi contributi scientifici forniti da Turing, tutti di grandissimo impatto concettuale e pratico, in questa sezione ne esaminiamo forse il più importante e famoso: quello relativo alla teoria della computabilità.

Anche se il termine Informatica – fortunata crasi di "Informazione automatica" – e la relativa tecnologia – l'elettronica digitale – sono relativamente recenti, essendo databili verso la metà del secolo scorso, le radici concettuali di questa disciplina sono ben più lontane nel tempo e non a caso si confondono con quelle di altre fondamentali scienze del pensiero umano come la matematica e la filosofia in particolare.

Il principale concetto dell'informatica, l'algoritmo, è infatti connaturato con la prima astrazione del "ragionar matematico" ossia il numero. Tra le tante formulazioni del concetto di algoritmo facciamo riferimento alla seguente:

> *Un algoritmo è un processo di elaborazione di informazioni, basato su una codifica rigorosa e precisa delle medesime e costituito da una sequenza di passi elementari definita in maniera altrettanto precisa e rigorosa, tanto da non lasciare alcun dubbio all'ente preposto alla sua esecuzione.*

È quindi evidente che le regole che impariamo fin dalle elementari per eseguire le operazioni aritmetiche, sono algoritmi (tutt'altro che banali se rapportati, ad esempio, con la numerazione greco-romana); è anche significativo che ancor oggi molti docenti di informatica – inclusi gli autori – tra i primissimi esempi di algoritmo presentino il famoso algoritmo di Euclide per il calcolo del massimo comun divisore tra due numeri interi positivi. Nella sua bellezza e semplicità questo algoritmo introduce in modo naturale uno dei principali strumenti di analisi e sintesi di algoritmi: l'invariante di ciclo; esso sfrutta infatti la proprietà che, se z è il massimo comun divisore tra x e y, esso lo è anche tra $x - y$ e y (se $x > y$) e da essa sintetizza un ciclo, ossia la ripetizione di una sequenza di operazioni, fino al raggiungimento del valore desiderato, quando cioè $x = y$.

Un po' meno evidenti ma di uguale e, in un certo senso, convergente impatto sono i legami tra informatica e logica matematica, anch'essa radicata nel periodo ellenistico grazie soprattutto al pensiero di Aristotele, ma con origini ancor più antiche nelle speculazioni di Parmenide e Zenone. Su questi legami torneremo tra breve.

Non può certo stupire dunque che, come per ogni altro processo, una volta individuate regole precise per ottenere un "manufatto"

l'uomo abbia desiderato e tentato di costruire un macchina cui affidare il compito di applicare tali regole, sollevando quindi se stesso dalla fatica di eseguirle in prima persona, oltre che con lo scopo di ottenere risultati migliori in termini di efficienza e qualità. Non è quindi il caso di insistere sulle analogie tra i tentativi di costruire macchine per il volo o macchine da guerra e quelli di costruire macchine per il calcolo. A ben pensare, anzi, in una prospettiva storica, la distanza temporale tra la rivoluzione industriale e la rivoluzione informatica, dovute alla nascita e allo sviluppo vertiginoso delle rispettive tecnologie, appare abbastanza limitata.

Senza addentrarci oltre nell'analisi di analogie e differenze tra l'ingegneria industriale tradizionale e quella legata all'informatica, osserviamo che entrambe hanno avuto un'autentica esplosione (con una lieve forzatura matematica, come il ginocchio dell'esponenziale) in pochi decenni, nel caso dell'informatica tra gli anni trenta e settanta del secolo scorso. Ciò che però è peculiare della storia dell'informatica e non può non stupire è che i fondamenti teorici di questa disciplina sono a loro volta maturati, dopo millenni di gestazione, in pochi anni, intorno al quarto decennio del 1900 e hanno preceduto e non seguito la realizzazione pratica dei primi calcolatori, che – è doveroso sottolineare – devono sì molto alla teoria che li ha ispirati, ma sarebbero rimasti ancora a lungo nel mondo delle idee se nel frattempo l'elettronica non avesse messo a disposizione tutta la potenza della sua tecnologia.

Questa sezione è dunque dedicata, anche in prospettiva storica, all'esame della teoria del calcolo automatico, o della computabilità, cui Turing ha fornito un contributo che definiremmo centrale nel contesto di numerosi altrettanto rilevanti risultati prodotti dai diversi pionieri della disciplina.

Il primo a porre in termini espliciti e rigorosi il fondamentale quesito legato al concetto di algoritmo e quindi di elaborazione automatica è stato forse Hilbert: se l'algoritmo è un procedimento meccanico o almeno meccanizzabile per risolvere un problema di elaborazione dell'informazione, una volta definito un problema di tal fatta, come posso ricavare un tale procedimento? O addirittura, come posso sapere se un tale procedimento esiste? O ancora, posso classificare o definire i

problemi risolvibili algoritmicamente? Hilbert ha certamente messo a fuoco queste domande fondamentali con estrema lucidità: famosissimo ad esempio, e rimasto a lungo senza risposta, il suo *decimo problema*. Ancor più generale e di maggior impatto, tanto teorico quanto pratico, ancor più ai nostri giorni, è il suo *Entscheidungsproblem* (Problema della decisione), già menzionato nella sezione precedente, che affonda le sue radici nella visione di Leibniz [Lei85]. Con terminologia moderna l'*Entscheidungsproblem* potrebbe essere definito *problema della validità*: data una formula, ad esempio espressa in logica del prim'ordine, è possibile stabilire in modo algoritmico se essa sia o meno valida ossia vera per ogni assegnamento di valore alle sue variabili libere? Si noti che già alla fine del 1800 Peano aveva utilizzato la logica matematica per formalizzare l'aritmetica [Pea81]. Potremmo dunque convenzionalmente ricondurre alla formulazione dell'*Entscheidungsproblem* il ricongiungimento definitivo tra informatica e logica matematica; integrazione che, invece che affievolirsi, è andata rinforzandosi sempre più con impatto pratico sempre crescente.

Nel 1931 Gödel aveva mostrato un formidabile ostacolo al programma di Hilbert con il suo secondo teorema di incompletezza, che dimostra che in ogni formalizzazione sufficientemente espressiva e coerente (non contraddittoria) della matematica (e in particolare, in ogni assiomatizzazione che include l'aritmetica) esistono delle formule vere, la cui verità non è dimostrabile a partire dagli assiomi del sistema. Questo risultato pone dei limiti invalicabili a quello che è possibile dimostrare meccanicamente con sistemi formali espressivi, ma lascia ancora aperta la possibilità di poter costruire una procedura di decisione automatica che, data una formula del prim'ordine, stabilisca se essa è vera, falsa, o indimostrabile (dagli assiomi della teoria). Anche altri matematici, ad esempio Markov e Church, stavano lavorando in quegli anni al problema della computabilità meccanica nel contesto del programma di Hilbert e con riferimento alle formalizzazioni di Russel e Whitehead.

In questo panorama irrompe prepotentemente Turing con il suo articolo del 1936 "On computable numbers with an application to the *Entscheidungsproblem*" [Tur37], che dimostra in maniera definitiva che una procedura di decisione automatica che risolva

l'*Entscheidungsproblem* non può esistere. In realtà Turing non è il primo a fornire una risposta di tal fatta. Anche le tecniche matematiche utilizzate da Turing, in particolare l'enumerazione a sua volta algoritmica di oggetti formali, non a caso denominata gödelizzazione, e la dimostrazione per assurdo mediante diagonalizzazione sfruttano meccanismi inventati e già proficuamente utilizzati rispettivamente da Gödel e Cantor. Ciononostante non esitiamo a definire centrale e fondamentale il contributo di Turing pur all'interno di un insieme di risultati che nel loro complesso in pochi anni hanno costruito l'asse portante della teoria dell'informatica; ciò che contraddistingue il contributo di Turing, come del resto gli è stato ampiamente riconosciuto dai colleghi contemporanei, del tutto privi di rivalità e invidie reciproche, è l'impostazione ingegneristica o costruttivistica che, da un punto di vista teorico, ha ricondotto problemi e teoremi formulati in puro stile matematico al tema originario del calcolo automatico e, d'altro canto, ha anche gettato le fondamenta per l'effettiva costruzione di macchine di calcolo, che sarebbero successivamente sfociate nello schema – e realizzazione – di von Neumann, ancor oggi riferimento principale per la gran parte della architetture di calcolo. Ben a ragione, dunque, il termine più frequentemente abbinato al nome di Turing è quello della sua *macchina*. Da essa dunque cominceremo a ripercorrere – seguendo un cammino ormai consolidato sia pur con qualche deviazione dall'articolo originale – il ragionamento che ha portato a dimostrare l'esistenza di problemi non risolvibili algoritmicamente.

La macchina di Turing

La macchina di Turing è un modello astratto di calcolatore che, grazie alla sua semplicità e naturalezza ha ispirato la definizione di una ricchissima famiglia di *macchine astratte* o *automi*. Essa consiste di due parti tra loro collegate:

- un *nastro* infinito, suddiviso in celle, ciascuna contenente un *singolo simbolo* (come una lettera dell'alfabeto o una cifra deci-

male) appartenente ad un insieme finito di caratteri detto *alfabeto* della macchina, che qui indicheremo con il simbolo A. Il nastro funge da *supporto per l'inserimento dei dati* (contiene i dati da elaborare), da *supporto di memoria* (è possibile leggere e cambiare il contenuto delle celle), e da *supporto di uscita* (il risultato del calcolo è parte di quello che rimane sul nastro quando l'esecuzione ha termine);
- un'*unità o organo di controllo*, che in ogni istante si trova in uno e un solo *stato*, appartenente ad un prefissato insieme finito che qui indicheremo con il simbolo Q; i suoi elementi saranno indicati dalla lettera q, eventualmente dotata di indice.

Una *testina di lettura-scrittura*, concettualmente analoga a quelle che fisicamente vengono utilizzate per registrare su, e leggere da, nastri magnetici, mette in comunicazione le due unità; in ogni istante essa è posizionata su una cella del nastro; chiameremo l'accoppiamento del contenuto del nastro, ivi inclusa la posizione della testina, con lo stato dell'unità di controllo, lo *stato complessivo* o *configurazione* della macchina.

La macchina evolve modificando la propria configurazione attraverso una serie di operazioni elementari dette *mosse*, *transizioni*, o passi discreti. Una transizione della macchina si articola in due fasi; nella prima la macchina:

- legge un dato dal nastro per mezzo della testina di lettura-scrittura;
- acquisisce lo stato dell'organo di controllo.

In base alle letture effettuate, successivamente la macchina:

- scrive un nuovo simbolo sul nastro al posto di quello letto;
- modifica lo stato dell'organo di controllo;
- sposta la testina di lettura-scrittura di una posizione a destra o a sinistra, o la lascia dove si trova. In alternativa, la macchina viene posta in uno stato di *arresto*, in cui cioè non è in grado di eseguire altri movimenti; a questo punto, e solo a questo punto, l'elaborazione è terminata.

Una *computazione* della macchina consiste dunque nel:

- Prepararla in una configurazione iniziale, che consiste in:
  - Un particolare stato dell'unità di controllo, detto stato iniziale, che indicheremo con q_0;
  - Una stringa finita di caratteri contenuta nel nastro; tutte le restanti, infinite, celle del nastro contengono un simbolo speciale detto *blank*, che qui indicheremo con il simbolo $\not b$;
  - La testina posizionata in corrispondenza del primo carattere non blank della stringa.

- Eseguire una sequenza di mosse a partire dalla configurazione iniziale secondo quanto specificato da una funzione $\delta : Q \times A \to (Q \times A \times \{R, L, S\}) \cup \{\perp\}$, dove R, L, S significano rispettivamente Destra, Sinistra, Nessun movimento, e $\perp$ significa *indefinito*; ad esempio:
  - $\delta(q_i, a) = \langle q_j, b, R\rangle$ significa che se la macchina si trova nello stato q_i e legge il simbolo a in corrispondenza della testina, allora mediante la prossima transizione si porta nello stato q_j, scrive il carattere b al posto di a e sposta la testina di una posizione a destra della posizione corrente;
  - $\delta(q_i, a) = \perp$ significa che la macchina si ferma definitivamente.

La sequenza di mosse, o computazione della macchina termina dunque quando (e *se*!) a un certo momento la macchina esegue la "non mossa" $\delta(q_i, a) = \perp$. A questo punto il contenuto del nastro – o una sua porzione opportunamente definita – rappresenta il risultato o output fornito dalla macchina.

La macchina quindi realizza, ossia *calcola*, una funzione che ha come dominio e codominio l'insieme delle stringhe finite di caratteri in A e che considereremo indefinita in corrispondenza di quelle stringhe per cui la computazione non terminasse mai, ossia non giungesse mai in una configurazione per cui $\delta(q_i, a) = \perp$.

ESEMPIO 1 – Descriviamo ora un primo esempio di macchina che calcola la funzione *successore* di un numero naturale assumendo che

argomento e valore della funzione siano codificati in numerazione binaria. La configurazione iniziale della macchina prevede dunque che il contenuto del nastro sia una successione di *bit* delimitata a destra e a sinistra dai blank che ricoprono la parte restante del nastro.

A grandi linee la macchina opera come segue:

- Sposta la testina fino a raggiungere il bit più a destra – il meno significativo della stringa di ingresso (per "accorgersi" di averlo raggiunto dovrà verificare che nella posizione successiva si trovi un blank).
- A questo punto, riportando la testina in corrispondenza del bit più a destra:
  - se esso è uno 0, al suo posto scrive un 1 e termina la computazione;
  - se invece è un 1, al suo posto scrive uno 0 e sposta la testina di un'altra posizione a sinistra; indi ripete l'operazione precedente; in altre parole:
    - continua a spostare la testina di una posizione per volta a sinistra e a riscrivere il carattere 0 al posto di 1 finché non trova il primo 0, se esiste: non appena lo trova riscrive un 1 al suo posto e termina la computazione.
    - Come caso particolare, se la stringa originaria consisteva di soli 1, la macchina procede a riscrivere gli 1 in 0 fino a trovare il blank che delimita la stringa a sinistra e scrive un 1 al suo posto prima di fermarsi definitivamente.

Il procedimento tratteggiato qui sopra – a tutti gli effetti un *algoritmo* – può essere facilmente dettagliato nella descrizione completa e formale della macchina, specificando esattamente i dettagli della funzione δ che la caratterizza: ad esempio un opportuno stato dell'unità di controllo potrà tenere traccia o memoria del fatto che la macchina è nella fase di ricerca del bit meno si-

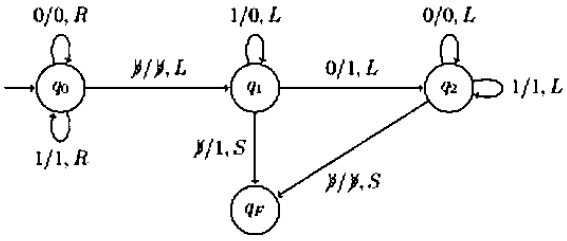

Figura 1. – Una macchina di Turing che calcola il successore di un numero binario.

gnificativo; la lettura del carattere blank in quello stato sarà il segnale dell'individuazione di tale bit e innescherà la nuova fase, caratterizzata dall'uso di un nuovo stato dell'organo di controllo e così via. La Figura 1 fornisce una definizione precisa e completa della funzione δ – e quindi della macchina – facendo uso di un simbolismo grafico ora di uso universale sia nelle notazioni tipiche della teoria degli automi che in quelle adottate come strumento di documentazione di progetto in vari settori dell'ingegneria. Nel grafo di figura gli stati dell'organo di controllo sono rappresentati dai nodi del grafo (si rammenti che q_0 denota lo stato iniziale); un arco etichettato a/b, M, con $M \in \{R, L, S\}$ va da q_i a q_j se e solo se $\delta(q_i, a) = \langle q_i, b, M \rangle$. La δ non è definita per $\langle q_i, a \rangle$ se nessun arco etichettato a (a sinistra del /) esce dal nodo q_i. □

Questo primo pur semplice esempio ci permette già di ricavare alcune fondamentali proprietà della macchina, che d'ora innanzi abbrevieremo con l'acronimo MT, grazie alle quali essa è, dal momento della sua "nascita", assunta come modello più astratto, generale e semplice, di quanto, prima e dopo il suo avvento, si intende per *calcolatore*.

1. Contrariamente a precedenti tentativi di costruzione di apparati meccanici per il calcolo di operazioni aritmetiche (in quegli anni cominciavano a diffondersi le calcolatrici meccaniche, e alcune di esse erano già integrate con motorini elettrici per sostituire l'azione manuale), la MT è un modello astratto e

generale, con lo scopo di poter descrivere *qualsiasi computazione automatica* ([2]).

2. La funzione δ costituisce un modo naturale ma matematicamente preciso di descrivere i singoli passi della computazione: a tutti gli effetti essa rappresenta il programma che codifica l'algoritmo concepito dal progettista, come è già immediatamente evidente confrontando la descrizione a grandi linee dell'Esempio 1 con la corrispondente Figura 1.
3. La macchina dell'Esempio 1 risolve un problema estremamente semplice – il calcolo del successore di un numero; tuttavia, già alla fine del 1800 Peano aveva assiomatizzato – in una logica "quasi" del prim'ordine – l'intera aritmetica sulla base di questa unica operazione elementare.
4. Non è difficile costruire esempi analoghi all'Esempio 1 e sulla base di una semplice esperienza, convincersi che, pur di usare una notevole dose di pazienza e di prendere in considerazione un gran numero di dettagli, sia possibile costruire MT che risolvano problemi anche molto più complicati del precedente.

A questo punto le precedenti considerazioni possono essere integrate con il confronto del modello di Turing con altri meccanismi, concreti o astratti, atti a descrivere una computazione automatica: è sempre risultato possibile verificare che ognuno di essi, precedenti, contemporanei, ma anche susseguenti alla MT, come ad esempio i numerosi *automi* fioriti negli anni 1950 e 1960, non ha una maggior *potenza computazionale*: per ogni problema risolvibile automaticamente mediante un qualsiasi strumento, è sempre possibile trovare una MT che risolva lo stesso problema (ovviamente esistono anche

([2]) È forse opportuno ricordare che in questo contesto facciamo riferimento esclusivamente alla computazione discreta. Macchine analogiche per eseguire calcoli matematici sono state a lungo utilizzate, a cominciare dal regolo calcolatore presente nel taschino di ogni ingegnere fino agli anni 70 del secolo scorso; vari tipi di calcolatori elettronici di tipo analogico sono stati sviluppati e utilizzati soprattutto nel contesto dei sistemi di automazione; essi sono però definitivamente tramontati non essendo in grado di fornire gli stessi livelli di precisione e potenza di calcolo della moderna tecnologia elettronica digitale.

numerosi modelli di calcolo meno potenti delle MT); ad esempio è facile mostrare come una MT possa simulare il comportamento della *Macchina di von Neumann*, il successivo modello, dovuto a von Neumann, che ancora adesso cattura gli aspetti essenziali della gran parte delle architetture di calcolatori. Queste constatazioni sulla riducibilità di altri modelli alle macchine di Turing hanno portato diversi pionieri della teoria della computazione a formulare la seguente, famosa tesi.

Tesi di Church-Turing

Nessun formalismo atto a descrivere un processo di calcolo automatico ha, né può avere, una potenza espressiva maggiore della MT o di altri formalismi ad essa equivalenti ([3]) .Tali formalismi costituiscono quindi il modo più generale possibile per descrivere qualsiasi algoritmo. In breve: ogni processo calcolabile è realizzabile da una macchina di Turing. □

Non a caso questa fondamentale affermazione è chiamata *Tesi* e non *Teorema*: essa è infatti basata su alcuni fatti matematicamente dimostrati – una volta definito un modello di calcolo X si può *dimostrare*, ed è stato dimostrato, che esso non può risolvere problemi che non siano risolvibili anche con una MT; ma è anche inevitabilmente legata al concetto intuitivo di algoritmo e a realistiche ipotesi su quali operazioni elementari siano realizzabili da opportune risorse fisiche che ne supportino l'esecuzione.

Ad esempio, è ragionevole assumere che una macchina esecutrice di algoritmi sia dotata di una capacità illimitata di memoria: ovviamente ogni macchina fisica ne avrà a disposizione una quantità finita ma possiamo anche assumere che, man mano che le dimensioni dei dati sui cui lavorare aumentano, si possa aumentare in parallelo anche la disponibilità di memoria fisica, eventualmente aggiungendo nuovi banchi o dischi; il semplice nastro della MT è quindi una ragionevole astrazione della memoria necessariamente associata all'esecuzione di

([3]) Ad esempio il Lambda-calcolo di Church [Chu56] o gli algoritmi normali di Markov [Mar54].

algoritmi; d'altro canto, una disponibilità illimitata di memoria non può implicare che in un tempo finito ne venga gestita (letta o scritta) una quantità infinita: la testina della MT può muoversi e accedere a una posizione per volta; quindi il suo nastro, in ogni momento, conterrà solo un numero finito di celle non contenenti un blank.

Osserviamo anche, per doverosa precisione storica, che per lungo tempo l'enunciato precedente è stato riferito esclusivamente come Tesi di Church; più recentemente (si veda ad esempio [Soa09]) però si sta affermando il termine più completo e preciso: come lo stesso Church riconobbe, infatti, tra le diverse formulazioni matematicamente equivalenti, il modello di Turing è quello che meglio rappresenta il concetto generale di *macchina* da calcolo. Nell'introdurre il suo modello di macchina, Turing era infatti esplicitamente alla ricerca di un meccanismo che catturasse ogni definizione ragionevole di processo computazionale. Nel suo articolo del 1936, egli propone tre argomenti distinti per cui la sua macchina potesse soddisfare tale requisito, tra cui il fatto che i principi di funzionamento fossero un'astrazione diretta di come opera un calcolatore umano quando "fa di conto" usando carta e penna. La tesi di Church-Turing ci autorizza perciò a identificare la Teoria della computabilità con la "Teoria delle macchine di Turing", che ci accingiamo a presentare, nei suoi aspetti fondamentali, nelle prossime sotto-sezioni.

L'enumerazione delle macchine di Turing

Il primo passo ai fini di investigare potenzialità e limiti delle MT, consiste nella loro enumerazione. Stabilire una corrispondenza biunivoca tra elementi di un insieme e i numeri naturali è infatti una tecnica largamente impiegata per vari scopi. Nel nostro caso definiremo una corrispondenza $\mathcal{E} : \mathcal{MT} \leftrightarrow \mathcal{N}$, dove $\mathcal{MT}$ rappresenta l'insieme delle MT e $\mathcal{N}$ l'insieme dei numeri naturali.

Preliminarmente, senza perdita di generalità, fissiamo l'alfabeto dei simboli che possono essere contenuti nelle celle del nastro: per semplificare la notazione assumiamo che tale alfabeto A contenga esattamente due elementi; un ben noto teorema di Shannon [Sha38]

garantisce che qualsiasi informazione può essere codificata mediante due simboli base e qualsiasi algoritmo per elaborarla può essere espresso con riferimento a tale rappresentazione; in altri termini ogni stringa su un certo alfabeto può essere messa in corrispondenza biunivoca con una stringa su un alfabeto di cardinalità due ([4]). Nel nostro caso quindi $A = \{\not b, |\}$ indicando con '|' l'unico simbolo diverso dal blank.

Indi, raggruppiamo le varie macchine in $\mathcal{MT}$ per numero crescente di stati. Ora, fissato A e il numero di stati (è indifferente il modo con cui identifichiamo gli stati all'interno di Q) esiste un numero finito di diverse macchine con alfabeto A e data cardinalità $|Q|$. Precisamente tale numero coincide con il numero di diversi modi con cui si può definire la funzione δ; ma δ, come si è visto, ha dominio e codominio finiti. Esistono quindi esattamente $(1 + 3 \cdot |Q| \cdot 2)^{|Q| \cdot 2}$ MT aventi A come insieme dei simboli e Q come insieme degli stati.

Fissiamo ora un qualsiasi ordinamento, ad esempio lessicografico, sia sul dominio $(Q \times A)$ che sul codominio $(Q \times A \times \{R, L, S\}) \cup \{\perp\}$ delle δ; infine, definiamo $M < M'$, dove M e M' sono due MT con k stati, se e solo se esiste una coppia $\langle q, a\rangle$ tale che $\delta(q, a) < \delta'(q, a)$ e, per ogni $\langle q', a'\rangle < \langle q, a\rangle$, $\delta(q', a') = \delta'(q', a')$. L'ordinamento così stabilito ci permette di indicare con M_0, la prima (meglio, la 0-esima) MT (quella a due stati con δ ovunque indefinita se abbiamo posto $\perp < \langle q, a, m\rangle$ per qualsiasi elemento in $Q \times A \times \{R, L, S\}$), con M_1 la seconda, e così via. Abbiamo dunque ottenuto una corrispondenza, ovviamente biunivoca, $\mathcal{E} : \mathcal{MT} \leftrightarrow \mathcal{N}$. È fondamentale rimarcare che

([4]) In fin dei conti i normali calcolatori elettronici usano proprio un alfabeto binario. Occorre però osservare che ciò non comporta un'immediata corrispondenza tra lo 0 e l'1 di un normale calcolatore e il blank e non-blank della MT: essa ha infatti un nastro infinito contenente, durante la computazione solo un numero finito di non-blank. La memoria centrale di un calcolatore consiste invece in un numero finito di byte (pacchetti di 8 bit l'uno). Volendo quindi usare i due simboli della MT nel modo più vicino possibile all'uso dei bit nei calcolatori reali conviene, ad esempio, codificare il bit '1' con una sequenza '$x\not b$', dove x denota il simbolo non-blank, e il bit '0' con la sequenza '$xx\not b$' in modo tale che una sequenza '$\not b\not b$' segnali automaticamente alla MT la fine dell'informazione significativa, come avviene in pratica con l'insieme dei caratteri ASCII dove una particolare sequenza di bit viene riservata per indicare End of Text.

$\mathcal{E}$ è anche una corrispondenza algoritmica, o calcolabile, o "effettiva ([5])". Chiunque abbia un minimo di esperienza di programmazione infatti si sentirebbe in grado di scrivere un programma che, data una tabella rappresentante la δ di una macchina M, calcoli il corrispondente numero d'ordine $\mathcal{E}(M)$ e viceversa. Allora, in base alla tesi di Church-Turing, sia la $\mathcal{E}$ che la sua inversa $\mathcal{E}^{-1}$ sono calcolabili da qualche MT.

D'ora innanzi indicheremo con M_i la macchina in posizione i-esima nell'enumerazione $\mathcal{E}$, e con f_i la funzione da essa calcolata. Si noti che ciò induce un'enumerazione anche delle funzioni calcolabili (è pleonastico aggiungere "da una MT"); tale enumerazione non è però biunivoca perché evidentemente potrà accadere che $f_i = f_j$ per qualche $i \neq j$. Nel seguito, inoltre, assumeremo che le MT siano costruite in modo da calcolare funzioni $f_i : \mathcal{N} \to \mathcal{N}$; questa assunzione non causa ovviamente perdita di generalità poiché qualsiasi alfabeto A contenente almeno due simboli può codificare tutti i numeri naturali e qualsiasi insieme (infinito) di stringhe di caratteri – quindi qualsiasi informazione in un insieme discreto – può essere messo in corrispondenza biunivoca con $\mathcal{N}$; di conseguenza ogni problema di elaborazione di informazione può essere formalizzato come una funzione $\mathcal{N} \to \mathcal{N}$.

Il procedimento di enumerazione – nel nostro caso algoritmica – degli elementi di un insieme è una tecnica largamente usata in matematica: essa permette infatti di "predicare" proprietà dell'insieme considerato, o di suoi elementi o di suoi sottoinsiemi, esprimendole in termini di relazioni tra i corrispondenti numeri; in particolare, essa era già stata utilizzata estensivamente da Gödel pochi anni prima di Turing per enumerare formule e teoremi della logica del prim'ordine e giungere ai suoi fondamentali teoremi di incompletezza dell'aritmetica. Per questo motivo si usa spesso il termine gödelizzazione e si parla di *numero di Gödel* dell'elemento x per indicare il numero naturale corrispondente a x nell'enumerazione fissata.

([5]) Come si usa dire con una sgradevole traduzione letterale dall'inglese.

La macchina di Turing universale

Una prima fondamentale applicazione dell'enumerazione delle MT si ottiene definendo la funzione $U : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$, come $U(x, y) = f_y(x)$; ossia $U(x, y)$ è il valore definito dalla y-esima MT nella precedente gödelizzazione. Nella sua scarna semplicità la funzione U racchiude un risultato di grande portata per l'intera teoria della computazione. In primo luogo infatti, non è difficile dimostrare il seguente enunciato.

TEOREMA 1. – *La funzione U è calcolabile.*

Traccia della dimostrazione

Descriviamo a grandi linee un algoritmo per il calcolo della U.

- In primo luogo, dato y, l'algoritmo calcola $\mathcal{E}^{-1}(y)$, ossia (un'opportuna rappresentazione del) la y-esima MT, ad esempio, una tabella che rappresenti la funzione δ di M_y.
- Successivamente acquisisce il dato x per cui calcolare il valore di $f_y(x)$ e non fa altro che simulare il comportamento di M_y quando opera su x, producendo alla fine lo stesso risultato che produrrebbe M_y. □

A questo punto è del tutto naturale invocare la Tesi di Church-Turing per ricavarne la seguente riformulazione del Teorema 1:

COROLLARIO 2. – Esiste una MT in grado di calcolare la funzione U. Di conseguenza essa avrà un suo indice, che denotiamo con u, nella precedente gödelizzazione.

A prima vista l'affermazione può risultare strana, poiché $\mathcal{E}$ enumera MT che calcolano funzioni $f_i : \mathcal{N} \to \mathcal{N}$ mentre il dominio di U è $\mathcal{N} \times \mathcal{N}$. Tuttavia è facile costruire a sua volta una corrispondenza biunivoca, calcolabile algoritmicamente, $\mathcal{N} \times \mathcal{N} \leftrightarrow \mathcal{N}$. In conclusione una MT M_u, che calcoli U descritta come funzione $U : \mathcal{N} \to \mathcal{N}$ opera secondo le linee seguenti:

- decodifica la coppia di dati $\langle x, y \rangle$ a partire dall'unico valore $n \in \mathcal{N}$;
- decodifica il valore y in una descrizione della macchina M_y, ad

esempio rappresentandone la funzione δ in un'apposita porzione del suo nastro;

- in un'altra porzione di nastro alloca (una rappresentazione del) dato x;
- Mediante continui spostamenti della testina:
  - accede alle singole posizioni nella rappresentazione di x, tenendo memoria dello stato in cui si troverebbe M_y a quel punto della sua computazione;
  - rintraccia nella parte di nastro che rappresenta la δ di M_y la definizione di come essa si comporterebbe in corrispondenza del dato letto e dello stato del suo organo di controllo;
  - individua l'azione corrispondente specificata da δ;
  - ritornando sulla porzione di nastro dedicata al dato x la modifica come farebbe M_y e tiene memoria dello stato in cui essa si verrebbe a trovare;
- Così di seguito finché non individua – se ciò accade! – una situazione in cui la δ di M_y non è definita e in tal caso si ferma avendo nell'apposita porzione di nastro lo stesso risultato che avrebbe prodotto M_y. □

Non a caso, dunque, la MT che calcola U (in realtà ne esistono infinite come per ogni altra funzione calcolabile da una qualsiasi MT) viene chiamata *macchina di Turing universale* (MTU). La MTU, e soprattutto il procedimento che ha portato alla sua costruzione, fornisce ulteriori elementi per valutarne l'efficacia come modello fondamentale per rappresentare macchine per il calcolo automatico e studiarne proprietà di validità generale. Si noti infatti che una singola e specifica MT codifica, nel suo semplicissimo ed elementare "linguaggio", un particolare algoritmo per risolvere uno specifico problema: essa si presenta quindi come un pezzo di hardware "special-purpose", costruito ad hoc per il singolo problema. Nella gran maggioranza dei casi invece la risoluzione automatica di un problema impiega un calcolatore "general-purpose" che viene appositamente programmato codificando nel suo linguaggio l'algoritmo individuato per la soluzione del problema del caso. Si noti infatti la fortissima analo-

gia tra lo schema di funzionamento della MTU e la classica "catena di programmazione":

- acquisizione del programma e dei dati su cui esso deve operare (un tempo era un unico pacco di schede perforate); ciò corrisponde all'input e alla decodifica del numero n che originariamente codifica la coppia $\langle x, y \rangle$;
- compilazione del programma nel linguaggio oggetto dell'hardware: ciò corrisponde alla decodifica di y nella rappresentazione della δ.
- esecuzione del codice oggetto mediante l'interprete hardware della macchina fisica: ciò corrisponde alla simulazione della M_y da parte della MTU.

Non a caso dunque i moderni calcolatori sono per lo più macchine a programma memorizzato (come la MTU memorizza nel suo nastro il "programma" δ), in contrapposizione alle macchine a programma cablato che, come le singole MT, eseguono sempre lo stesso algoritmo, codificato una volta per tutte nella propria circuiteria.

Sui numeri computabili

Ora che siamo in possesso di uno strumento astratto e generale al massimo livello possibile possiamo affrontare l'affascinante problema di stabilirne potenzialità e limiti raccogliendo le sfide poste da Hilbert; ossia, secondo la formulazione proposta da Turing nel suo articolo originario, quali "numeri" siano computabili mediante il suo strumento. Con terminologia più adeguata alla moderna teoria della computabilità, posto che ogni problema – nel senso matematico del termine – può essere formulato come una funzione sui numeri naturali: quali funzioni sono computabili algoritmicamente? In termini ancora più precisi: data una funzione $f : \mathcal{N} \to \mathcal{N}$, esiste una M_i tale che $f_i : \mathcal{N} \to \mathcal{N}$ coincida con f?

Ora che la domanda è stata formulata in modo così chiaro e preciso, la risposta si può ottenere in modo in un certo senso sorprendentemente semplice.

In primo luogo alcune semplici considerazioni legate alla cardinalità degli insiemi mostrano come la "stragrande maggioranza" delle funzioni (d'ora innanzi, a meno di avviso contrario, sottintenderemo $f : \mathcal{N} \to \mathcal{N}$) non è computabile: più precisamente, l'insieme $\{f : \mathcal{N} \to \mathcal{N}\}$ ha notoriamente la cardinalità del continuo; la funzione $\mathcal{E}$ invece stabilisce una corrispondenza biunivoca tra MT e $\mathcal{N}$, quindi la cardinalità dell'insieme delle funzioni computabili non può essere maggiore di $\aleph_0$.

Questa constatazione però non ha un grande impatto pratico perché la "stragrande maggioranza" delle funzioni non solo non è computabile ma non è neanche *definibile*. Infatti per definire una qualsiasi funzione dobbiamo usare una qualche forma di espressione, in ultima analisi una frase in un opportuno linguaggio. I linguaggi, visti come oggetti matematici, altro non sono che sottoinsiemi del monoide libero costruito su un alfabeto finito, ossia insiemi a loro volta numerabili. Quindi l'insieme delle funzioni cui si può fare riferimento esplicito per porsi poi il problema se siano calcolabili o meno, è a sua volta un insieme numerabile.

Il risultato fondamentale della teoria della computabilità fornito da Turing e i suoi contemporanei, non solo dimostra come sia possibile definire problemi ben posti, ossia definiti in modo matematicamente inequivoco, ma non risolvibili algoritmicamente, ossia da una macchina, ma ne fornisce anche alcuni fondamentali esempi che hanno poi permesso di costruire una serie di risultati su "che cosa sia calcolabile e che cosa non lo sia", dando quindi una risposta conclusiva alla sfida lanciata da Hilbert ([6]).

Il problema della terminazione del calcolo

Soffermiamoci su una proprietà finora non sottolineata delle funzioni computabili: per definizione $f_y(x)$ è il valore – codificato in termini dell'alfabeto A – che la macchina M_y lascia sul suo nastro al termine

([6]) Anche se per ottenere la risposta al famoso decimo problema si è dovuto attendere fino al 1970 [Mat93], e alcuni problemi della lista sono ancora aperti.

della sua computazione, ossia quando si trova in uno stato q e legge dalla testina un simbolo s tale che $\delta(q, s) = \perp$. Nulla di quanto è insito nella definizione della MT e del suo funzionamento però garantisce che, a partire da una configurazione iniziale, dopo un numero finito di mosse essa si venga a trovare in una tale *configurazione di arresto*: in taluni casi la computazione potrebbe non terminare mai. Ne consegue che, se per un certo dato di ingresso x, M_y non termina la sua computazione, il valore $f_y(x)$ non è definito. In generale quindi, una generica funzione $f_y : \mathcal{N} \to \mathcal{N}$ risulta essere una funzione parziale. Da un punto di vista della matematica classica, ciò non è un grave inconveniente; infatti raramente nella matematica tradizionale si pone l'accento sulla distinzione tra funzioni totali e funzioni parziali: basta aggiungere un valore convenzionale al codominio della funzione, ad esempio il simbolo $\perp$ già utilizzato per il valore indefinito della δ, e definire convenzionalmente $f_y(x) = \perp$ tutte le volte che M_y non termina la sua computazione in corrispondenza del dato di ingresso x.

Poniamoci però la domanda: dopo aver esteso in maniera così scontata la definizione di f_y, – denotiamola con $\widehat{f_y}$ per chiarezza – possiamo ancora dire che M_y calcola $\widehat{f_y}$? O, ancor più in generale, possiamo ancora dire che $\widehat{f_y}$ è computabile, ossia che esiste una MT, magari anche diversa da M_y, che calcola $\widehat{f_y}$? La definizione di calcolo di una funzione da parte di una macchina richiede che il suo valore sia il *risultato finale* di una computazione: il contenuto del nastro, nel caso di una MT: non si può quindi dire che la M_y calcola $\widehat{f_y}$ perché essa non scrive sul suo nastro il valore $\perp$ "indefinito" che è stato convenzionalmente "definito" per $f_y(x)$. Dobbiamo quindi metterci alla ricerca di una *nuova* macchina che sia capace di "calcolare" quando, per un certo dato x, M_y non si arresterebbe (e scrivere $\perp$ in output di conseguenza).

Prima di procedere con l'analisi del problema appena posto, sottolineiamone la generalità: chiunque abbia una minima esperienza di programmazione sa che, quando si lancia l'esecuzione di un programma, solitamente non si è sicuri che esso terminerà la sua esecuzione in un tempo finito e, siccome invece il nostro tempo è finito, dopo un eventuale lungo periodo di attesa, se l'esecuzione non è ancora terminata, siamo costretti a sospenderla senza sapere se siamo soltanto stati troppo im-

pazienti o se la nostra attesa era comunque destinata a non avere termine. Nuovamente, grazie alla tesi di Church-Turing, il problema di stabilire se una generica MT operando su un generico dato terminerà o meno la sua computazione trascende il semplice modello di Turing e diventa, molto più in generale il *problema della terminazione del calcolo*.

Orbene, il risultato fondamentale stabilito da Turing nel suo famoso articolo stabilisce che il *problema della terminazione del calcolo (automatico) non ha soluzione all'interno del calcolo (automatico).* A nostro parere, più ancora della tecnica dimostrativa usata da Turing per giungere a questa affermazione, è illuminante la sua formulazione matematica, che si può enunciare nel modo seguente:

TEOREMA 3. – *Nessuna* MT *può calcolare la funzione* $f_H : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$ *così definita:*

$f_H(y, x) = 1$ *se* M_y *termina la sua computazione in corrispondenza del dato* x, $f_H(y, x) = 0$ *altrimenti.* □

Si noti la somiglianza tra la definizione di f_u e quella di f_H: a parte l'aver sostituito il valore 1 a $f_u(y, x)$ quando la computazione di M_y termina, la differenza apparentemente inessenziale ma in realtà fondamentale tra le due definizioni sta nell'aver sostituito al valore convenzionale $\perp$ il risultato 0; ma lo 0, per essere un risultato della f_H, deve essere scritto sul nastro e la computazione deve terminare: esattamente il contrario della condizione che attribuisce il valore $\perp$. Proprio su questa fondamentale differenza si basa la dimostrazione formale del teorema.

DIMOSTRAZIONE. – Si supponga, per assurdo, che f_H sia computabile da qualche MT. Allora la funzione *parziale* h definita come

$$h(x) = \textbf{if } f_H(x, x) = 0 \textbf{ then } 1 \textbf{ else } \perp$$

risulta pure computabile. Infatti, una volta ottenuta una MT M in grado di calcolare f_H, sono sufficienti solo semplici modifiche per adattarla al calcolo di h: basta infatti porsi sulla diagonale $y = x$, trasformare un risultato 0 in 1 e forzare la nuova macchina a non fermarsi mai ogni volta che M produce il risultato 1.

Sia allora x_0 il numero di Gödel di una MT M_{x_0} che calcola h: ossia $f_{x_0} = h$. Allora, per la definizione di h, se $h(x_0) = f_{x_0}(x_0)$ è definita, $h(x_0) = 1$ e $f_H(x_0, x_0) = 0$. Tuttavia, per la definizione di $f_H(x, y)$, ciò implica che $f_{x_0}(x_0)$ sia indefinito.

Se invece $h(x_0)$ risulta indefinito, $f_H(x_0, x_0) = 1$ che, a sua volta, implica che $f_{x_0}(x_0) = h(x_0)$ sia definito.

In entrambi i casi si ottiene una contraddizione, quindi f_H non è computabile. □

Radici filosofiche e impatto pratico dell'indecidibilità del problema della terminazione del calcolo.

A chiunque abbia un po' di dimestichezza con alcune classiche tecniche dimostrative logico-matematiche non sarà sfuggito che i diversi passi utilizzati per giungere alla dimostrazione del Teorema 3 non impiegano meccanismi particolarmente innovativi:

- L'enumerazione algoritmicamente definita di un insieme di oggetti è tecnica che era già stata portata alle estreme conseguenze da Gödel – e non a caso da allora chiamata gödelizzazione – per enunciare e dimostrare i suoi teoremi di incompletezza.
- La dimostrazione diagonale del Teorema 3 segue le linee del classico ragionamento di Cantor per dimostrare che $\mathcal{N}$ non può essere messo in corrispondenza biunivoca con il suo insieme delle parti – e quindi non può esserne equinumeroso – ragionamento che a sua volta affonda le sue radici in ben noti paradossi filosofici, con origini nel periodo ellenistico e ripresi svariate volte fino al paradosso di Russell.

La grandezza del risultato di Turing quindi, pur sempre inquadrato nel contesto dei contributi contemporanei e della lunga storia pregressa, sta a nostro avviso nella formidabile sintesi filosofico-matematico-ingegneristica che esso ci offre. Degli aspetti più squisitamente matematico-filosofici ci occupiamo prevalentemente nella sezione seguente. Qui invece ci preme sottolineare l'impatto pratico che ancor oggi lega la teoria della computabilità, di cui l'enunciato di Turing

rappresenta senz'altro l'origine storica, a tutta una serie di problemi di carattere applicativo.

1. L'indecidibilità ([7]) della terminazione del calcolo, già di per sé di grande impatto pratico, si è rivelata la radice concettuale di un gran numero di altri risultati di incomputabilità: la tecnica base utilizzata per il Teorema 3 si può generalizzare e applicare a molti altri problemi; ancor più naturale e comoda è la tecnica di riduzione tra problemi (introdotta da Turing nella sua tesi di dottorato [Tur39]): così come è possibile sfruttare un algoritmo che, data la soluzione del problema P_1 calcoli la soluzione di un problema P_2 per derivare la computabilità di P_2 dall'ipotesi di computabilità di P_1, lo stesso algoritmo dimostrerebbe l'incomputabilità di P_1 sulla base di una già nota incomputabilità di P_2. In questa maniera si ottiene immediatamente, ad esempio, l'indecidibilità di tutte le classiche proprietà dei programmi che espongono al rischio dei cosiddetti errori a "run-time": la divisione per 0, l'overflow di memoria, la mancata inizializzazione di variabili, ecc. La derivazione e la valutazione dell'impatto di questi risultati è molto più evidente e naturale facendo riferimento a un modello di macchina astratta – che in modo naturale rappresenta qualsiasi architettura di calcolo o linguaggio di programmazione – piuttosto che a formulazioni di tipo logico-deduttivo.
2. Il confronto tra la calcolabilità della funzione f_u e la non calcolabilità di f_H porta naturalmente al concetto di *semidecidibilità*. Si noti infatti che la "parte positiva" delle due funzioni, ossia laddove f_u è diverso da $\perp$ e $f_H = 1$, è effettivamente computabile: la MTU calcola $f_u(y, x)$ arrestandosi in un tempo finito e similmente può produrre il risultato 1 di f_H; il problema della terminazione del calcolo – e quindi tutti quelli ad esso riducibili – è quindi *semidecidibile* nel senso che, se la risposta ad esso è positiva, la

([7]) È prassi consolidata usare il termine "indecidibile" come sinonimo di "non computabile" quando il problema cui si fa riferimento è espresso in termini booleani, ossia richiede una risposta binaria (0 o 1, vero o falso, Sì o No).

MTU è in grado di fornirla, non così però in caso contrario. Questa constatazione è all'origine di numerosi *semialgoritmi* che "procedono per tentativi": in buona sostanza essi *enumerano* tutte le possibili – solitamente infinite – soluzioni di un problema e per ognuna di esse *verificano* se effettivamente la candidata è la (o una) soluzione cercata. In moltissimi casi infatti è molto più facile verificare se una potenziale soluzione è effettivamente tale, piuttosto che individuarla *ex novo*. Un tale approccio esaustivo – detto anche "brute force" nella letteratura internazionale – è molto più usato in pratica di quanto non si pensi, grazie anche alla grande potenza di calcolo raggiunta dalla tecnologia moderna. Un classico esempio di applicazione di questo approccio è il "testing del software", laddove si enumerano – seguendo un qualche criterio euristico – i possibili dati di test: parafrasando liberamente un celebre slogan dovuto a E.W. Dijkstra "se l'errore c'è si trova... ma se non c'è non si avrà mai la certezza della sua assenza": il problema della *presenza* di errori in un programma – non della sua assenza! – è dunque semidecidibile, ma non decidibile. Anche in questo caso appoggiarsi al semplice modello astratto di Turing favorisce la comprensione di queste proprietà in modo intuitivo e generale, generalizzando i meccanismi base di tipo enumerativo.

3. Il modello di Turing non solo ha permesso di arrivare in maniera conclusiva e intuitiva al fondamentale risultato sui "numeri non calcolabili", ma ha rappresentato anche il punto di (ri)partenza verso i risultati successivi sulla teoria della computazione (termine ora più adeguato e generale del precedente "teoria della computabilità"). Infatti, dopo aver distinto con chiarezza problemi risolvibili da problemi irrisolvibili (sempre sottintendendo l'avverbio "algoritmicamente"), lo stesso modello e le stesse tecniche hanno permesso di impostare e affrontare il problema della *complessità computazionale*, ottenendone nuovi fondamentali risultati.

   La complessità computazionale affronta il problema di valutare non solo se un problema sia risolvibile meccanicamente, ma anche quanto costi ricavare la sua soluzione. Anche in questo caso il

modello di Turing si è rivelato un ottimo e generale "strumento di misura": associando in modo naturale la singola transizione della macchina all'unità di tempo e la singola cella all'unità di memoria è possibile ottenere stime contemporaneamente astratte – ossia indipendenti dalle numerosissime variabili di carattere implementativo, a cominiciare dalla velocità del processore fisico utilizzato per eseguire l'algoritmo – e di sicuro riferimento per una valutazione realistica dei costi della computazione.

Il classico *teorema dell'accelerazione lineare* in sostanza afferma che è sempre possibile ottenere miglioramenti *lineari* (ossia di un fattore costante rispetto alla dimensione dell'input) di prestazioni nell'esecuzione di algoritmi a parità di modello di calcolo pur di aumentare le risorse fisiche a disposizione; ad esempio, il numero di bit per parola di memoria, la velocità del processore, il numero stesso di processori; tale accelerazione però non altera l'ordine di grandezza (denotato mediante le notazioni asintotiche O-grande o Θ-grande, introdotte da Landau e popolarizzate in informatica da Knuth [Knu69]) della funzione che formalizza il costo dell'esecuzione. Inoltre la tesi di Church-Turing nell'ambito della complessità computazionale – spesso chiamata "versione *forte* (cioè quantitativa) della tesi di Church-Turing" [MS11] – sancisce che è sempre possibile simulare la computazione di un modello di calcolo $\mathcal{M}_1$ attraverso un modello $\mathcal{M}_2$ garantendo una relazione polinomiale tra la complessità ottenuta attraverso il modello $\mathcal{M}_2$ e quella relativa al modello $\mathcal{M}_1$; in altre parole, dette f_1 e f_2 rispettivamente le due funzioni rappresentanti la complessità dei due modelli per risolvere un generico problema, esistono sempre un polinomio P_1 tale che $f_2(x) = P_1(f_1(x))$ e un polinomio P_2 tale che $f_1(x) = P_2(f_2(x))$. In particolare, nonostante l'evidente differenza tra l'architettura della MT e quella della macchina di von Neumann, permettendo la seconda l'accesso diretto a una posizione di memoria al contrario della prima che impone una scansione sequenziale da cella a cella, un *teorema di correlazione polinomiale* dimostra che

una MT può simulare il comportamento di un qualsiasi calcolatore reale con una complessità al più quadratica ([8]) rispetto a quest'ultimo.

Questi fondamentali risultati sono alla base di tutta una serie di risultati della moderna teoria del calcolo di grandissimo impatto anche pratico. Ne citiamo qui alcuni esempi, fortemente correlati tra loro ([9]):

- Grazie al teorema di correlazione polinomiale la MT continua ad essere il riferimento principale per formulare e dimostrare proprietà generali sulle caratteristiche del calcolo automatico, complessità computazionale in primis.
- Grazie, anche, al teorema di correlazione polinomiale è da tempo convenzione universalmente accettata individuare nella classe $\mathcal{P}$, ossia l'insieme dei problemi risolvibili in tempo polinomiale – senza bisogno di dover specificare mediante quale modello di calcolatore! – la classe dei *problemi trattabili,* ossia di quei problemi ritenuti risolvibili con costi accettabili.
- La classificazione e il relativo ordinamento dei vari problemi rispetto alla loro complessità computazionale: ad esempio la classe dei problemi risolvibili in tempo $O(n^3)$ contiene strettamente quella dei problemi risolvibili in tempo $O(n^2)$.
- Il concetto di *completezza* di un problema rispetto ad un'intera classe di problemi simili: generalizzando la tecnica di riduzione (della soluzione) di un problema ad uno o più altri problemi si ottengono dei "campioni" di famiglie di problemi: se si è capaci di risolvere con una certa complessità un problema P se ne ricava automaticamente la possibilità di risolvere con la stessa complessità, o con complessità correlata mediante una ben precisa funzione, un'intera classe di problemi rappresentata da P.

([8]) O cubica, a seconda di come si valuti la complessità delle operazioni moltiplicative nei calcolatori reali.

([9]) Per motivi di brevità e per non deviare troppo rispetto al tema principale, siamo costretti a riassumere in poche righe e quindi in modo necessariamente superficiale, risultati di grande ampiezza e portata.

Un caso particolare di completezza che da decenni appassiona e sfida una vasta schiera di ricercatori di informatica teorica e al momento continua ad eludere i loro sforzi è il caso dell'*NP-completezza*. Esso fa riferimento alla computazione *nondeterministica*, ossia a un processo di calcolo in cui, in un certo stato della computazione, la transizione verso il prossimo stato non sia univocamente definita ma possa essere arbitrariamente scelta dall'esecutore all'interno di un numero finito di possibili alternative. In tempi relativamente recenti si sono sviluppati numerosi settori applicativi che si giovano di un tale modello di calcolo; per citarne uno tra tutti, si immagini un algoritmo per la ricerca di un elemento all'interno di una data struttura, ad esempio un grafo: se da un nodo si dipartono più archi e ognuno di essi potrebbe portare all'elemento cercato, in mancanza di specifiche informazioni per indirizzare la ricerca, l'algoritmo potrebbe non specificare quale nodo scegliere per proseguire la ricerca, lasciando l'esecutore libero e responsabile della propria scelta.

Anche il concetto di meccanismo di calcolo nondeterministico appartiene alle intuizioni di Turing, il quale, sempre nel suo articolo del 1936, distingue tra "automatic machine" la cui evoluzione è completamente determinata dalla configurazione corrente, e "choice machine" la cui evoluzione è invece solo parzialmente determinata dalla configurazione.

Non sappiamo fino a che punto si sia spinta l'intuizione di Turing nel valutare l'impatto potenziale di questa versione della sua macchina; sta di fatto che, dopo una facile constatazione che il passaggio dalla versione deterministica a quella nondeterministica non aumenta la potenza di calcolo della macchina nel senso della classe di problemi risolvibili (non cambia il contenuto della tesi di Church-Turing), a partire dalla fine degli anni sessanta ha assunto un'importanza fondamentale e sempre crescente l'obiettivo di stabilire una relazione tra la complessità della soluzione nondeterministica e di quella deterministica di un dato problema: fino ad oggi non si è ancora riusciti a dimostrare con certezza che il modo naturale per simulare una computazione nondeterministica mediante una deterministica con un salto esponenziale nella corrispondente complessità sia anche l'unico modo possibile, nonostante molti risultati parziali suggeriscano fortemente che questo limite sia intrinseco.

L'accoppiamento del concetto di nondeterminismo con quello di completezza ha poi portato alla definizione di *problema NP-completo*, ossia di un "rappresentante" di tutti i problemi risolvibili in tempo polinomiale da una macchina nondeterministica, il cui insieme viene denominato $\mathcal{NP}$, e, soprattutto, all'individuazione di una quantità incredibile – decine di migliaia, quasi tutti di grande interesse applicativo – di tali problemi: se per uno solo di essi si trovasse una soluzione deterministica a complessità polinomiale l'intera classe $\mathcal{NP}$ coinciderebbe con la classe $\mathcal{P}$; viceversa, se per uno solo di essi si dimostrasse la non esistenza di un algoritmo di soluzione deterministico a complessità polinomiale, la stessa proprietà negativa varrebbe automaticamente per tutti gli altri problemi NP-completi.

Algoritmi e intelligenza: macchina, uomo, o dio?

Il pensiero e l'opera di Alan Turing si collocano in una lunga e grandiosa tradizione filosofico/scientifica che affronta il problema formidabile di caratterizzare la natura e i limiti ultimi del pensiero razionale, nel contesto del confronto dell'uomo tra se stesso (γνωθι σεαυτον) e il mondo che lo circonda, tra le proprie possibilità di agire su di esso (la costruzione di macchine che operino su di esso in propria vece e possibilmente più efficientemente di se stesso) e l'eventualità di una potenza trascendente dotata di capacità che esulano dalle proprie. Una delle caratteristiche più ambiziose di questa linea speculativa è la ricerca di una caratterizzazione *universale*, ossia che trascenda i limiti contingenti della "macchina" cervello e comprenda una nozione affatto generale di pensiero e conoscenza.

Questo tipo di domande ha un carattere che trascende facilmente la pura curiosità intellettuale, acquisendo anche un forte significato pratico e ingegneristico. Il filosofo illuminista Leibniz è stato uno dei primi a intuire, e ad esplicitare, l'impatto della ricerca sulla computazione e a compiere dei passi fondamentali in questa direzione. Con i suoi *calculus ratiocinator* e *characteristica universalis*, Leibniz ambiva niente meno che a "rettificare il nostro ragionare" rendendolo "tangibile come quello della matematica" [Lei85]. Questo avrebbe

portato a un modo infallibile di risolvere dispute e contrasti tra persone: sarebbe stato sufficiente "calcolare", ossia applicare in maniera meccanica le regole di ragionamento per stabilire con certezza chi abbia ragione. La visione di Leibniz è quella di una conoscenza assoluta raggiunta attraverso un procedimento meccanico realizzabile da una macchina.

Occorreranno circa due secoli prima che le idee di Leibniz diventino parte della ricerca matematica e oggetto di uno studio sistematico. Verso la fine dell'ottocento, Frege introduce le idee fondamentali della logica matematica moderna, che può essere vista come realizzazione del linguaggio universale di Leibniz. Mentre la logica diventa gradualmente una parte importante della matematica – fino a diventare una componente fondamentale del programma di Hilbert, al quale abbiamo già accennato nelle sezioni precedenti – alcuni risultati storici ne delineano i limiti intrinseci. Russel e Whitehead [WR09] con i loro paradossi mostrano le difficoltà di formalizzare la teoria degli insiemi senza introdurre contraddizioni. Gödel con i suoi teoremi di incompletezza mostra che la potenza espressiva della logica matematica comporta anche dei limiti: esistono delle verità matematiche che non sono teoremi, ossia non sono dimostrabili con le regole "universali" della logica matematica. Turing amplia il risultato di Gödel e ne dà una caratterizzazione pratica in termine di macchine: il calcolo automatico ha dei limiti intrinseci che discendono direttamente dalle proprie potenzialità; esistono delle funzioni perfettamente definite ma che non sono costruibili con l'applicazione meccanica di un numero finito di regole.

Stabiliti questi limiti del calcolo meccanico, gli interrogativi originari di confronto tra uomo e macchina riprendono forma in un contesto più definito. La domanda fondamentale è quindi in che misura questi limiti intrinseci siano applicabili anche al pensiero umano. La caratteristica che sembra mancare ai calcolatori artificiali è una forma di coscienza di sé che riconosciamo negli esseri umani (e in una certa misura anche negli animali). Possiamo dunque chiederci se l'uomo può essere l'artefice di un'intelligenza qualitativamente confrontabile con la propria: un cervello artificiale dotato di *intelligenza artificiale* commensurata con quella naturale.

Il tentativo di comprendere l'essenza delle relazioni tra *naturale* e *artificiale* è stato al centro degli interrogativi affrontati da scienza e, prima di essa, filosofia e religione. Molte rivoluzioni della conoscenza sono caratterizzate proprio dall'introduzione di un nuovo livello di comprensione di questo aspetto – tipicamente nel senso di un *assottigliamento* del confine tra naturale e artificiale. La rivoluzione copernicana ad esempio, con il suo compimento nel quadro cosmologico sviluppato da Galileo e Newton, ha rimosso i confini tra "terrestre" e "celeste" mostrando principi universali che governano la materia e il moto di pianeti e stelle. Nonostante lo scetticismo di pensatori del calibro di Kant – che riteneva non ci sarebbe mai stato un "Newton per la foglia d'erba" [Kan92] – la teoria dell'origine delle specie di Darwin e la sintesi in laboratorio di materia organica hanno dimostrato la continuità tra materia "vivente" e "non vivente" contrariamente alla teoria del "vitalismo" in voga all'inizio del diciannovesimo secolo.

Anche a queste domande al confine tra scienza e filosofia Turing fornisce un contributo lucido ed essenziale che è ancora di grande attualità. Con la consueta capacità di dare concretezza e rigore a questioni astratte, Turing affronta la domanda "le macchine sono capaci di pensare?" fornendo una definizione operativa di intelligenza. Nel suo articolo "Computing machinery and intelligence" [Tur50b] descrive una procedura sperimentale – passata alla storia con il nome di *test di Turing* – per stabilire se possiamo considerare "intelligente" una macchina capace di interazioni verbali con persone. Turing immagina un "gioco dell'imitazione" dove un umano interrogante siede davanti a un terminale dotato di tastiera e schermo. Lo schermo è diviso in due finestre, una collegata con un altro operatore umano (nascosto e comunicante attraverso un terminale), l'altra con un "computer macchina". L'interrogante può dialogare in tempo reale attraverso del testo con l'uno o l'altro interlocutore a piacere. Il suo scopo è quello di identificare a quale finestra corrisponda la macchina e a quale la persona, basandosi unicamente sul comportamento nel dialogo testuale. Se in una serie di esperimenti ripetuti l'interrogante indovina correttamente chi è l'uomo e chi la macchina nella maggior parte dei casi, il computer ha *fallito* il test di Turing. Se invece l'interrogante è spesso indotto a confondere uomo e macchina, questa ha *passato* il test

di Turing nel senso che esibisce comportamento intelligente – per analogia col fatto che attribuiamo intelligenza a una persona sostanzialmente sulla base del suo comportamento in una chiacchierata. Turing è attento a dare delle previsioni quantitativamente precise su un possibile esito positivo del test: specula che "nel giro di circa 50 anni sarà possibile programmare dei computer che indurranno un interrogante medio in errore nel 30% dei casi dopo una sessione di 5 minuti di domande". La sua conclusione è che questa procedura permette di porre delle domande sul comportamento delle macchine che *rimpiazzano* la domanda originaria "le macchine sono capaci di pensare?" con una in stretta relazione ma che sia "relativamente non ambigua".

Il valore del test di Turing, al di là della sua carica suggestiva, risiede sostanzialmente in due aspetti. In primo luogo, si tratta di una *definizione* di comportamento intelligente verificabile sperimentalmente. Questo fornisce fondamenta per fare del problema dell'intelligenza artificiale un problema investigabile con i metodi della scienza, strappandolo alla sola speculazione filosofica. In secondo luogo, la definizione originaria del test di Turing pone dei limiti stringenti alle modalità e dinamiche di interazione tra l'interrogante e la macchina (ad esempio, come abbiamo detto il dialogo non dovrebbe durare più di 5 minuti). Questa attenzione agli aspetti quantitativi, se da un lato limita la generalità dell'"intelligenza" misurabile attraverso il test, ne fa anche uno strumento più utile in pratica per misurare il progresso concreto delle macchine nel cammino dell'intelligenza artificiale [War12].

Il fascino del test di Turing è certificato anche dai numerosissimi esperimenti che si sono susseguiti per applicarlo nell'ambito di specifici domini applicativi (conversazioni con persone affette da disturbi mentali, oppure relative a domini ristretti e semplici come mondi geometrici, ordinazioni in ristoranti, ...). Warwick [War12] riferisce in maniera circostanziata di tali esperimenti. Il fatto però che essi siano stati quasi sempre interpretati in maniera controversa annunciando di volta in volta clamorosi successi o eclatanti fallimenti dimostra che al di là delle semplici e precise regole definite da Turing, molti margini di interpretazione rimangono per individuare il vero significato da attribuire ai risultati di tali esperimenti in termini di livello di intel-

ligenza raggiunto. Mettiamo inoltre in guardia dal confondere il test di Turing come test della capacità della macchina di simulare (o realizzare?) un comportamento "intelligente" con il confronto tra le prestazioni di macchine e di uomini in specifici settori: se da un lato nessuno si stupisce per il fatto che un treno o un automobile siano molto più veloci di un uomo, maggiori rischi di confusione si corrono confrontando le prestazioni delle macchine con quelle dell'uomo su attività mentali: anche in questo caso, il fatto che un calcolatore possa eseguire diversi miliardi di moltiplicazioni in un secondo non è considerato sintomo di intelligenza, ma quando per la prima volta il calcolatore Deep Blue dell'IBM sconfisse il campione mondiale di scacchi Garry Kasparov si è rimesso in discussione la caratterizzazione goethiana del gioco degli scacchi come "pietra miliare dell'intelletto".

Dunque le due idee probabilmente più importanti nella produzione di Turing – la macchina di Turing e il test di Turing – non sono altro che *definizioni* che pongono solide fondamenta per lo sviluppo dell'informatica. Ma se queste definizioni sono sopravvissute allo scorrere del tempo è anche perché Turing non si è limitato a definire ma ha sempre complementato la propria investigazione studiando in che misura il "reale" e la "natura" offrano delle possibilità di implementare e meccanizzare processi computazionali e mentali. La ricerca di una spiegazione – logica e fisica – dell'attività mentale è stato uno dei tratti distintivi di tutto il suo percorso scientifico, durante il quale Turing ha esplorato le possibilità offerte dalla meccanica quantistica, dalla logica matematica, e dalla materia biologica. In questa ricerca ha probabilmente attraversato dubbi, ripensamenti, e nuove prospettive, riuscendo sempre però a trovare sintesi rilevanti, originali, e influenti.

Infatti, oltre settanta anni dopo la sua discussione nell'articolo "On computable numbers", la tesi di Church-Turing rimane insuperata. A questo proposito, non devono trarre in inganno alcuni articoli apparsi nell'ultimo decennio anche su autorevoli riviste di informatica [WG03,Coo12], che sembrano mettere in dubbio la validità della tesi. Si tratta per lo più di presentazioni che tendono a confondere il problema di definire un modello di computazione universale, che possa cioè simulare ogni processo computazionale,

con lo sviluppo di modelli computazionali che catturino in maniera diretta e semplice certe istanze non-tradizionali di computer. Ad esempio, è innegabile che descrivere le interazioni di processi concorrenti distribuiti in una rete di calcolatori (come Internet) per mezzo di un tradizionale modello sequenziale come la macchina di Turing risulterebbe estremamente tedioso e produrrebbe un modello formale difficile da analizzare; ciononostante, non c'è niente nell'interazione di processi concorrenti che non possa essere simulato con una semplice macchina di Turing, come per ogni altro processo computazionale noto per quanto bizzarro o complicato. Insomma, a distanza di decenni la tesi di Church-Turing dorme ancora sonni tranquilli ([10]). Fortnow [For10] fornisce una buona sintesi e una disamina approfondita di queste recenti affermazioni.

Possiamo dunque concludere che i risultati di Turing rimangono dei capisaldi della speculazione scientifica e filosofica, ma dove ci lasciano rispetto all'interrogativo fondamentale della relazione tra intelligenza umana e macchine calcolatrici? Se il pensiero è un processo computazionale come gli altri, come possiamo spiegare la differenza tra computer e persone? Forse si tratta solo di una differenza di prestazioni: il cervello umano non è altro che un computer, e quello che chiamiamo "intuito" non è altro che un insieme sofisticato di euristiche ("programmate" dall'evoluzione naturale) che sono in grado di esplorare gli spazi infiniti di problemi anche indecidibili in maniera da estrarre soluzioni particolari ma rilevanti e utili in pratica.

([10]) Il discorso si arricchisce di nuove sfumature nel caso della "versione forte" della tesi di Church-Turing. Esistono infatti modelli di calcolo (basati sulla meccanica quantistica) che permetterebbero di risolvere problemi come la fattorizzazione, presumibilmente intrattabili (ma non NP-completi!) con modelli di calcolo tradizionali. D'altro canto, a dispetto del fatto che sia ormai universalmente congetturato che $\mathcal{P} \neq \mathcal{NP}$, sono ora disponibili diversi strumenti in grado di risolvere problemi NP-completi in tempi "spesso" accettabili. Questi risultati sperimentali non violano i risultati teorici affermati in precedenza ma suggeriscono di estendere la teoria della complessità computazionale mediante meccanismi di misura diversi dai tradizionali *caso pessimo* e *caso medio*. Anche in questo caso siamo sicuri che Turing avrebbe fornito adeguati e oggettivi criteri per superare l'attuale impasse tra risultati teorici "troppo generali o troppo pessimistici" e esperienze pratiche di dubbia generalità.

Non abbiamo ancora delle risposte complete e definitive a questi interrogativi fondamentali, una costante che accompagna la storia dell'uomo e del pensiero razionale anche se sempre in nuove forme e contesti. Rimaniamo testimoni dell'entusiasmante divenire e del progresso della conoscenza e della tecnica che sembra avvicinarsi sempre più ad una risposta – anche senza raggiungerla mai. Ci restano la ammirazione e gratitudine nei confronti di chi, come Turing, ha contribuito in maniera così fondamentale e imperitura alla comprensione di questi interrogativi eternamente affascinanti.

BIBLIOGRAFIA

[BG03] ANDREAS BLASS and YURI GUREVICH, *Algorithms: A quest for absolute definitions.* Bulletin of the EATCS, 81:195-225, 2003.

[Chu36] ALONZO CHURCH, *A note on the Entscheidungsproblem.* Journal of Symbolic Logic, 1:40-41, 1936. Ristampato in [Dav04].

[Chu56] ALONZO CHURCH, *Introduction to Mathematical Logic*, Princeton University Press, 1956.

[Coo12] S. BARRY COOPER, *Turing's titanic machine?* Commun. ACM, 55(3):74-83, 2012.

[Dav04] MARTIN DAVIS, editor. *The Undecidable*. Dover Publications, 2004.

[EW03] EUGENE EBERBACH and PETER WEGNER, *Beyond Turing machines.* Bulletin of the EATCS, 81:279-304, 2003.

[For10] LANCE FORTNOW, *Ubiquity symposium 'What is computation?': The enduring legacy of the Turing machine.* Magazine Ubiquity, Article no. 5, December 2010, ACM.

[Göd31] KURT GÖDEL, *Über formal unentscheidbare sätze der Principia Mathematica und verwandter Systeme*. Monatshefte für Mathematik und Physik, 38:173-198, 1931. Ristampato tradotto in inglese in [Dav04].

[HMU09] JOHN E. HOPCROFT, Rajeev Motwani, and Jeffrey D. Ullman. *Automi, linguaggi, e calcolabilità*. Pearson, versione italiana a cura di G. Pighizzini, 2009.

[Hod83] ANDREW HODGES, *Alan Turing: The Enigma*. Burnett Books, 1983. Edizione italiana: Bollati Boringhieri, 1991.

[Kan92] IMMANUEL KANT, *Critica del giudizio* Laterza, 1997 (data di pubblicazione originaria: 1892).

[Knu69] DONALD KNUTH, *The Art of Computer Programming*, Addison Wesley, Vols. 1, 2, 3, 1969-1973.

[Lei85] GOTTFRIED WILHELM LEIBNIZ. *Scritti di Logica*. A cura di F. Barone, Laterza, 1992 (data di pubblicazione originaria: 1685).

[MS11] DINO MANDRIOLI and PAOLA SPOLETINI, *Informatica Teorica*. Città studi edizioni, 2011. II edizione.

[Mar54] ANDREJ MARKOV, *The Theory of Algorithms*, Transactions of the American Mathematical Society, 2(15): 1-14, 1954.

[Mat93] YURI MATIYASEVICH, *Hilbert's 10^{th} Problem*, MIT Press, 1993

[Pea81] GIUSEPPE PEANO, *Sul concetto di numero*, Rivista di Matematica, 1: 87-102 e 256-267, 1981

[Sha38] CLAUDE SHANNON, *A Symbolic Analysis of Relay and Switching Circuits*, Transactions American Institute of Electrical Engineers, 57: 713-723, 1938.

[Sip05] MICHAEL SIPSER, *Introduction to the Theory of Computation*. Course Technology, 2nd edition, 2005.

[Soa09] ROBERT I. SOARE, *Turing oracle machines, online computing, and three displacements in computability theory.* Annals of Pure and Applied Logic, 160:368-399, 2009.

[Tur34] ALAN M. TURING, *On the Gaussian error function*. Manoscritto (dissertazione per la fellowship a Cambridge), 1934.

[Tur37] ALAN M. TURING, *On computable numbers, with an application to the Entscheidungsproblem.* Proceedings of the London Mathematical Society, 42:230-265, 1937. Ristampato in [Dav04].

[Tur39] ALAN M. TURING, *Systems of logic based on ordinals.* Proceedings of the London Mathematical Society, 45, 1939. Tesi di Ph.D. presentata a Princeton, ristampato in [Dav04].

[Tur43] ALAN M. TURING, *A method for the calculation of the Zeta-function.* Proceedings of the London Mathematical Society, 48, 1943.

[Tur45] ALAN M. TURING, *Proposed electronic calculator*. (Il "rapporto ACE" verrà pubblicato da NPL nel 1972), 1945.

[Tur48] ALAN M. TURING, *Rounding-off errors in matrix processes.* Quarterly Journal of Mechanics and Applied Mathematics, 1, 1948.

[Tur50a] ALAN M. TURING, *Checking a large routine*. Technical report, Cambridge Mathematical Laboratory, 1950. Apparso in "Annals of the History of Computing", vol. 6, 1984, a cura di F.L. Morris e C.B. Jones.

[Tur50b] ALAN M. TURING, *Computing machinery and intelligence.* Mind, 59:433-460, 1950.

[Tur52] ALAN M. TURING, *The chemical basis of morphogenesis.* Philosophical Transactions of the Royal Society (B), 237, 1952.

[War12] KEVIN WARWICK, *Not another look at the Turing test!* In *SOFSEM 2012: Theory and Practice of Computer Science – 38th Conference on Current*

Trends in Theory and Practice of Computer Science. Proceedings, volume 7147 of *Lecture Notes in Computer Science*, pages 130-140. Springer, 2012.

[WG03] PETER WEGNER and DINA Q. GOLDIN, *Computation beyond Turing machines*. Commun. ACM, 46(4):100-102, 2003.

[WR09] ALFRED NORTH WHITEHEAD and BERTRAND RUSSELL, *Principia Mathematica*. Merchant Book, 2009. (Pubblicazione originaria: Cambridge University Press, 1910, 1912, 1913; seconda edizione 1925, 1927).

Carlo Alberto Furia:
Chair of Software Engineering – ETH Zurich
e-mail: carlo.furia@inf.ethz.ch

Dino Mandrioli:
Dipartimento di Elettronica e Informazione – Politecnico di Milano
e-mail: dino.mandrioli@polimi.it