Tratto da *Lettera matematica pristem*, n. 27-28 (1998)

Metafore matematiche dell'Informatica

di Corrado Bhöm

Evoluzione del *computer* e relativi linguaggi

I rapporti del matematico con l'Informatica sono certo più raffinati rispetto a quelli di un generico utente di *computer*, anche se talvolta traggono giovamento dai progressi generali.

Conviene soffermarsi un istante su quello che dovrebbe essere il comportamento del *computer* auspicato dagli umani. Il paradigma è *tutto e subito*, come spesso si pretende da una macchina, sia essa un'automobile o un elettrodomestico. La differenza fondamentale fra un *computer* e un automobile o un elettrodomestico consiste nel tipo d'informazione che noi dobbiamo dare alla macchina per la realizzazione dei nostri desideri. Nel secondo e terzo caso ci si può limitare ad azionare poche leve o manopole o pulsanti contrassegnati, mentre nel caso del *computer* non sarebbe sufficiente la pulsantiera di una cabina di pilotaggio aereo.

Schermo, tastiera e *mouse* sono venuti in soccorso, ormai da più di vent'anni. Non v'è dubbio, il modo principe della comunicazione umana è la parola, sia essa scritta o parlata. Con un po' di pena si è risolto il problema della dattilografa automatica ossia il riconoscimento da parte del *computer* della parola pronunciata. Tuttavia questo non basta ancora a far sì che il linguaggio umano sia il veicolo privilegiato per la comunicazione con il *computer*. Oggi, cinquant'anni dopo l'introduzione del *computer*, non possiamo fare a meno dei cosiddetti linguaggi programmativi che ben poco hanno d'umano. Il fatto è che ogni *computer* è realizzato per eseguire precise istruzioni, che constano di successioni di *bit* o cifre binarie che non lasciano nulla alla fantasia. Altrettanto si può dire degli indirizzi di memoria e del loro contenuto.

I primi linguaggi di programmazione hanno permesso di assegnare nomi fissi alla parte d'istruzione che indica l'operazione, e nomi a scelta per indicare le locazioni di memoria (linguaggi assemblativi o macrolinguaggi).

Nella seconda fase dello sviluppo dei linguaggi è stata adottata una maggiore disciplina nella chiamata dei programmi e nella loro concatenazione. Era la fase della cosiddetta programmazione strutturata.

Nella terza fase tale disciplina viene in uggia. Allora i linguaggi si biforcano:

- da un lato sono creati linguaggi più elastici quali il C;
- dall'altro lato fioriscono linguaggi *funzionali*, la cui semantica si allontana da quella più operativa del linguaggi dei *computer* (o alla von Neumann), per avvicinarsi maggiormente a quella *denotazionale*.

Nell'ultima fase, l'attuale, i linguaggi di programmazione si evolvono nel senso di riavvicinare e rifondere assieme la precedente biforcazione per mezzo della programmazione *a oggetti* e *a vincoli*. Questi ultimi linguaggi e quelli funzionali sono peraltro molto più vicini al linguaggio matematico che non tutti i precedenti.

L'uso del linguaggio naturale per programmare i *computer* non ha avuto finora un successo indiscusso; il COBOL, orientato alla descrizione di problemi commerciali o di gestione, ancora sussiste e usa qualche parola del linguaggio naturale.

Una delle ragioni di questo successo molto parziale è che noi umani siamo infarciti di metafore e luoghi comuni, e certo i *media* di massa (televisione, giornali e riviste) contribuiscono in larga misura a questa situazione. Invece oggi il *computer*, per funzionare adeguatamente, necessita *ancora* di comandi precisi, anche se contemperati dall'acquisizione di alcune metafore di comportamento, vale a dire: il

video come *scrivania*, su cui giace ogni sorta di *documenti colorati*, aperti o chiusi, e che è affiancata simbolicamente da un telefono, un fax, un lettore di CD, un altoparlante, un microfono, un sintonizzatore televisivo, una stampante, un lettore automatico di documenti, una protesi che rimpiazza il dito indice (il *mouse*) e persino un cestino di rifiuti.

Metafore matematiche in Matematica

Non vorrei trattare il problema del linguaggio umano e della sua comprensione da parte del *computer*, attività studiata dall'*Intelligenza artificiale*. Il problema è veramente interessante, ma forse ancora non maturo. Qualcosa di più accessibile e forse utile per raggiungere la soluzione del problema appena accennato (*computer knoledge*), mi sembra l'esame di certe metafore, da tempo usate in Matematica ed estensibili anche all'informatica.

Che cos'è una metafora? È un uso di parole destinato ad aiutare la memorizzazione di nuove situazioni, basato sull'uso di vecchi nomi o locuzioni in nuove situazioni. In sostanza, si dice una cosa intendendone un'altra.

L'azione in questo caso è tuttavia propedeutica, non disonesta. L'uso, infatti, di una metafora invoca, presso chi l'ascolta o legge, un processo di traduzione semantica. Una risposta più precisa alla domanda: una metafora è un nome o una locuzione, il cui significato originale è da supporre noto a chi l'ascolta, ma cui è attribuito un nuovo significato, intuitivamente suggerito da quello primitivo al quale si aggiunge, talvolta caricandolo più del previsto.

Un esempio matematico è la parola *tangente*, sinonimo di *toccante*; attribuita a un segmento di retta e a una curva s'intende che il segmento tocca la curva senza attraversarla. Parlando di funzioni angolari, se l'angolo è compreso fra l'ipotenusa e un cateto di un triangolo rettangolo, allora la tangente e la funzione il cui valore è il rapporto fra la lunghezza del cateto opposto a tale angolo e quella del cateto adiacente. Vorrei avvertire preliminarmente che, nel modo con cui esemplificherò la nozione di metafora, mi prenderò l'arbitrio di continuare a considerare come metafora una parola o locuzione, anche dopo che è stata definita in modo preciso o assiomatico.

Tra le molte metafore della Matematica vorrei citarne almeno due, quella degli enti (o numeri) *infinitesimi* e quella degli enti (o numeri) *infiniti*. Gli infinitesimi erano invocati ancora nell'Ottocento per spiegare in termini intuitivi e informali il calcolo *infinitesimale*, o *sublime*, com'era allora denominato.

I numeri (o insiemi) infiniti sono stati introdotti da Cantor in modo non metaforico ma strettamente matematico. Ciò che è curioso è il fatto che, intorno alla metà del secolo, gli infiniti e gli infinitesimi siano stati trattati come enti matematici i cui assiomi (nella cosiddetta "aritmetica non standard" del logico Abraham Robinson) erano molto più vicini alle metafore ottocentesche che non agli sviluppi dell'Analisi in questo secolo.

Metafore matematiche in Informatica

Volendo sintetizzare al massimo, e nominare le principali metafore in Matematica, Informatica e Fisica, potremmo dire: infinito, continuità e simmetria per la Matematica, finito e asimmetria per l'Informatica e unione di tutte le precedenti per la Fisica, mentre tutte e tre impiegano ovviamente materia e spazio.

Il multinsieme

La previa ripartizione è abbastanza superficiale, tuttavia può motivare la tendenza in Informatica a eliminare il concetto d'insieme a favore di quello più generale di multinsieme (dovuto a Foata e Knuth), che, com'è noto, fonde in uno i due concetti d'insieme e di funzione (introducendo la molteplicità d'appartenenza). Si noti l'innaturalezza con cui in Logica matematica si rompe la

simmetria e si forza un ordine nel concetto d'insieme (ovviamente disordinato) nei prossimi tre esempi:

1. definendo ricorsivamente un numero naturale come l'insieme di tutti i numeri che lo precedono, perciò zero coincide con l'insieme vuoto, uno con il singoletto che contiene zero ecc. come dallo schema seguente
   0 = { }, 1 = {0} = { { } },
   2 = {0, 1} = { { }, { { } } },
   3 = {0, 1, 2} = { { }, { { } }, { { } , { { } } } } ecc.
   Questa rappresentazione dei numeri non è molto perspicua. Quasi più semplice è la descrizione della funzione *succ* (che aggiunge un'unità ad un numero) come
   succ(n) = n ∪ {n}
   dove ∪ sta per l'unione di due insiemi, e per ogni insieme *x* si ha
   { } ∪ *x* = *x*;
2. definendo una coppia ordinata d'elementi < a, b > come l'insieme
   {{a}, {a, b}}.
   Così Kuratowski è riuscito a dimostrare, per quattro insiemi arbitrari a, b, c, d, l'implicazione fondamentale delle coppie
   < a, b > = < c, d > → (a = c) ∧ (b = d).
   Da notare che la dimostrazione si può fare solo perché
   {z, z} = {z}.
   L'implicazione nell'altro verso, che insieme alla prima caratterizza la coppia, è banalmente vera. Da notare che
   < b, b > = { {b}, {b, b} } = { {b}, { b} } = { {b} }.
   Non si è mai vista alcuna applicazione rilevante dell'ultima relazione.
3. definendo la nozione di terna dei tre elementi a, b, c come
   < < a, b >, c > = { { { {a}, {a, b} } }, { { {a}, {a, b} }, c} },
   e in modo simile dando la definizione di *n*-upla.

A un multinsieme, invece, un elemento può appartenere più volte, quindi {z, z} è diverso da {z}. Con i multinsiemi la terna è definibile, e in modo naturale, come
< a, b, c > = {{a}, {a, b}, {a, b, c}},
e similmente la *n*-upla con $n \geq 1$. La dimostrazione è semplice, per induzione su *n*, partendo da
< *x* > = *x*.
In Informatica, cioè, si preferisce prendere come fondamentale la nozione di sequenza dove, a differenza di ciò che succede per gli insiemi, sia l'ordine in cui appaiono gli elementi sia la loro eventuale ripetizione sono indicativi. La nozione è così importante che viene introdotta nelle tre varianti di stringa, *n*-upla e lista, dove rispettivamente gli elementi o appartengono a un insieme finito (stringa), o sono in numero prefissato (*n* per la *n*-upla) o il numero, detto anche lunghezza, è un grado di libertà della lista. Il concetto di lista (stringa) corrisponde a quello di sequenza finita e libera dalla necessità di conoscere *a priori* il numero degli elementi della *n*-upla. Cioè, mentre < a, b > è una coppia e < a, b, c > è una terna, gli oggetti analoghi [a, b] e [a, b, c] ("ab", "abc") sono ambedue liste (stringhe) di lunghezza differente.

Stato di un calcolo

Intuitivamente, la nozione di stato istantaneo di un calcolo è il complesso dei risultati intermedi finora raggiunti più la memoria di tutto quello che eventualmente servirà per il proseguimento del calcolo. Secondo ogni specifico modello di calcolo, sia esso fisicamente costruito o solo teorico, si hanno le seguenti varianti:

- nella macchina di von Neumann (l'attuale *computer*) lo stato istantaneo è lo stato globale della memoria più l'indicatore della prossima istruzione da eseguire;
- nella macchina di Turing o automa lo stato istantaneo è l'attuale scrittura sul nastro (finita), più un puntatore sull'attuale carattere letto sul nastro, più il nome dell'attuale stato interno della macchina;
- in alcuni sistemi di riscrittura, inizialmente lo stato è il messaggio iniziale ovvero i dati per la trasformazione ingresso-uscita, quest'ultima rappresentata dall'insieme o lista di regole di calcolo, mentre in un momento intermedio del calcolo è : un puntatore alla regola che si sta per applicare e un altro alla trasformata attuale del messaggio iniziale;
- nel lambda calcolo, visto come un sistema di riscrittura con una sola regola di trasformazione (la riduzione immediata) lo stato è un lambda-termine rappresentante inizialmente l'applicazione di una funzione a dati specifici e in ogni momento intermedio un'espressione che ha come significato lo stato finale, cioè il valore dell'espressione iniziale.

Facciamo un esempio di programma che usa alcune delle metafore precedentemente introdotte. L'idea di questo programma mi è venuta in mente qualche tempo fa, durante un'ora di coda in un ufficio bancario.
Vogliamo organizzare il servizio in un ufficio postale o in una banca. Ogni utente indica il tipo di operazione che deve fare, preleva un numero d'ordine e si va a sedere in una delle seggiole libere. Ogni sportello può fare qualsiasi operazione e mostra al pubblico una file di numeri, il primo dei quali è il prossimo ad essere servito. Ogni sportello ha un solo cliente, quello che sta servendo.
Quando un cliente è servito, suona un campanello e il numero del primo della fila si evidenzia per alcuni secondi, dopodiché, se il relativo cliente non si presenta, egli perde definitivamente il turno. In ogni caso il numero scompare.
Al tipo di operazione indicato all'inizio dal cliente è assegnata una durata (minima) dell'operazione. La composizione delle file (o code virtuali) può cambiare durante il tempo di attesa. La legge che regola la composizione di tali code è: tutte le file devono essere in ordine crescente. Lo scopo da raggiungere è: massimizzare il numero delle persone che escono dall''ufficio nello stesso ordine in cui sono entrate. Le eccezioni sono regolate dalla legge seguente: se gli sportelli sono in numero di *n+1* un cliente può uscire dopo *h* persone con un numero superiore al suo, solo se il tempo da lui impiegato allo sportello supera un ennesimo del tempo globale impiegato dalle *h* persone. Com'è strutturato l'algoritmo?
Lo stato di calcolo e una *n+1*-pla di liste di numeri in ordine crescente. In funzione del dato d'ingresso vogliamo calcolare il nuovo stato di calcolo.
Il dato d'ingresso è di due tipi, corrispondenti all'ingresso o all'uscita di un cliente.
Nel primo caso, il nuovo numero del cliente è piazzato in coda a una lista di sportello, corrispondente a una durata globale minima. Così si è determinato il nuovo stato.
Nel secondo caso supponiamo che lo sportello che il cliente lascia, dopo essere stato servito, corrisponda alla lista *i*-esima aggiornata, il cui numero di testa è stato cancellato.
Si operi allora nel modo seguente: consideriamo la lista *i*-esima e le altre *n* liste. Confrontiamo fra loro le *n+1* teste di lista. Il minimo fra questi numeri diventerà la nuova testa della lista *i*-esima effettuando l'eventuale scambio: ed ecco il nuovo stato.
Il significato del precedente esempio consiste nel paragone fra due quantità. Sono occorse una trentina di righe per spiegare il problema usando il linguaggio naturale senza l'uso di nessuna metafora matematica. Sono occorse una trentina di righe per descrivere la soluzione del problema, sempre nel linguaggio naturale, ma usando metafore matematiche come: *n*-upla, lista, testa di lista. L'identificazione della struttura dei dati insita nel problema è stata l'azione decisiva per la sua risoluzione, ma l'uso delle metafore ha abbreviato molto la descrizione del procedimento.

Teoremi e *computer*

Lo scopo che mi sono proposto è essenzialmente quello di mostrare il valore semantico della metafora in Informatica.

Volgiamoci ora ad un altro aspetto che lega Matematica e Informatica. Si tratta di un aspetto logico che spazia fra due estremi; come certificare la correttezza dei programmi e come dimostrare teoremi con l'ausilio del *computer* o addirittura automaticamente. Da un punto di vista teorico si entra nel dominio della cosiddetta *teoria formale della dimostrazione*.

All'inizio, cioè intorno al 1967, il problema della correttezza dei programmi fu affrontato da Naur, Floyd e Hoare, sostanzialmente accompagnando ogni frase del programma con asserzioni che aggiornavano le verità scaturite dall'esecuzione di quella frase di programma.

Oggi si tende a seguire un processo opposto, quello dell'estrazione di programmi da prove che dimostrano l'esistenza della funzione entrata-uscita relativa ad ogni singolo programma.

In ambedue i casi però, il processo non è ancora, in generale, completamente automatizzato, anche perché bisogna possedere l'arte di non cadere in proposizioni indecidibili, arte non ancora delegabile a un *computer*. Il fatto che una dimostrazione formale contenga molto più di quanto serve per costruire un programma, lascia sospettare che vi sia troppa sintassi nella dimostrazione e che quindi sia opportuno orientarsi maggiormente verso la semantica. Esistono metodi semantici di controllo di correttezza.

Voglio citare un aneddoto tratto dall'autobiografia*: Sicuramente sta scherzando, Mr. Feynman!*

Alla *Graduate School* di Princeton, Feynman era studente di Fisica, ma spesso s'incontrava con altri studenti di Matematica per discutere di teoremi. I fisici tendevano polemicamente ad affermare che tutti i teoremi presentati dai matematici avevano un contenuto banale. Talvolta succedeva anche che, dopo l'enunciazione di un teorema, Feynman esplodesse nell'esclamazione: Falso! Discutendo risultava poi che, se il teorema era vero, nell'affrettata esposizione qualcosa era stato tralasciato.

Effettivamente Feynman aveva un metodo. Mentre ascoltava costruiva un modello mentale che adattava via via per soddisfare alle premesse del teorema. Durante l'esposizione di un teorema di Topologia, seguendo questo metodo, appena il teorema fu enunciato disse appunto: "Falso!". Presentò poi un contro esempio e qualcuno osservò: "Ci siamo dimenticati di premettere che siamo nel caso della classe 2 d'omomorfismo di Hausdorff". Al che Feynman esclamò: "Giusto! Quindi il teorema è banale!". Nel libro poi egli aggiunge che non sapeva minimamente che cosa significasse "omomorfismo di Hausdorff".

La morale della storiella è che l'immagine mentale che Feynman costruiva per capire le premesse di un teorema può essere chiamata una metafora personale. Sarebbe molto bello se tali costruzioni potessero essere indotte da qualche programma di *computer* e acquistare così un aspetto oggettivo.

Accenniamo ora a metodi pragmatici già usati, se non per provare la correttezza di un programma, almeno per aumentare la fiducia verso di essa.

Si tratta di fare dei *test*. Se il programma, come d'uso, è fornito di diramazioni o salti condizionati, la convalidazione per mezzo di esempi, in modo che ogni possibile diramazione sia *coperta* da almeno un esempio, è una condizione necessaria per la correttezza.

Qualcosa di simile ho sperimentato, qualche anno fa, per la funzione *Tak*, inventata come *benchmark* da un ricercatore giapponese di nome Takeuti o Takeuchi (la traslitterazione può divergere, ma si tratta dello stesso nome) allo scopo di misurare la complessità di calcolo di programmi ricorsivi. Poiché ho notato che esisteva una funzione *tak* (minuscolo), descrivibile essenzialmente come un albero di decisioni binarie, i cui valori coincidevano con la *Tak* laddove fossero calcolati per entrambe le funzioni, si trattava di dimostrare l'uguaglianza delle due funzioni ovvero l'equivalenza dei rispettivi programmi.

Il problema era: possono due funzioni appartenenti a una determinata classe coincidere se il loro valore coincide per un numero finito d'argomenti?
Ciò è noto essere vero per la classe delle funzioni polinomiali, ma non nel caso descritto.
Questo tipo di ragionamento, prove sostituite da esempi, è molto affascinante. A tali ragionamenti si può affiancare anche il calcolo probabilistico.
La verifica probabilistica d'identità polinomiali di grado molto elevato è il soggetto di un lavoro di Schwartz del 1980, in cui fra l'altro si mostra come a identità polinomiali si riducano gli enunciati di molti teoremi della geometria piana euclidea.
Quest'ultimo argomento è stato portato a perfezione formale dal cinese Jawei Hong con un lavoro dal titolo *Prove mediante esempi* (1989).
In quell'articolo si dimostra come, per ogni teorema di geometria elementare la cui verifica ottica si possa fare mediante una costruzione con riga e compasso (per esempio le proprietà che le tre mediane di un triangolo s'incontrano in un punto), tale verifica, ipotizzando la *genericità* dell'esempio scelto, è essa stessa una prova del teorema.
La dimostrazione è algebrica, ma contiene anche le informazioni di qual è l'errore massimo di precisione che la costruzione può contenere senza invalidarne il significato.

Conclusione

La vita umana è costellata da atti ripetitivi, che avviliscono le possibilità creative, portando al tedio e forse anche alla depressione. Le macchine calcolatrici prima, e i *computer* poi, sono stati inventati *anche* con lo scopo di eliminare tali atti ripetitivi e il loro successo si deve al parziale raggiungimento di questo scopo. Credo però che, se gli utenti del *computer* si chiedono onestamente se sono riusciti a eliminare gesti ripetuti, evitabili in linea di principio ma in qualche modo forzati dalla macchina, dovrebbero, per lo più, rispondere negativamente.
A queste per ora inevitabili ripetizioni si sono aggiunte recentemente lunghe attese, o che lunghe appaiono anche se sono di qualche minuto, tra il momento in cui si accende un computer e quello in cui esso diventa disponibile. Ovviamente, a questo inconveniente si può rimediare lasciando il *computer* sempre acceso.
Per quanto riguarda le ripetizioni qualcuno potrebbe obiettare che gli atti ripetitivi materiali costituiscono la base della nostra vita e quindi sono ineliminabili; questo è vero, ma mi riferivo, all'inizio, ad atti che riguardano la comunicazione con altri o in ogni caso ad attività che hanno a che fare con la mente.
Esiste quindi la necessità di sveltire le comunicazioni dell'uomo verso la macchina e viceversa, evitando le ripetizioni. Noi, come ci comportiamo nelle comunicazioni con gli altri? Cerchiamo di evitare ripetizioni usando locuzioni piazzate al momento giusto del tipo: "Eccetera, eccetera". Poi riduciamo il numero dei vocaboli, usando parole plurisignificanti, sperando che il nostro interlocutore, sfruttando il contesto verbale, spaziale e temporale, capisca. In altri termini usiamo metafore a tutto spiano. È quanto mi sono proposto di fare in questo intervento, mostrando la potenza espressiva di certe parole e dei concetti sottostanti, e cercando di incoraggiarvi a fare ricerche simili: trovare nuove e potenti metafore, mediante le quali il dialogo umanità–*computer* diventi sempre più conciso e rapido nella messa a fuoco degli argomenti che si vogliono trattare.