linguaggio digitale e precisione della scrittura

Cosimo Laneve

L'informatica è la scienza che studia il modo di calcolare. Discuterò del concetto di precisione della scrittura, uno dei più spinosi problemi dell'informatica, che è stato risolto dopo sessant'anni di ricerca. Mi soffermerò su come la scrittura possa influenzare il modo di calcolare noto agli addetti ai lavori, meno ai non addetti ai lavori. Accennerò, infine, alla relazione esistente fra sintassi di un linguaggio e macchine calcolatrici. L'articolo intende raggiungere non già la comunità scientifica in senso stretto, ma quella in senso ampio.

Parole chiave: calcolare, scrivere, informatica.

Computer science is the science that studies "how to compute". I will discuss the concept of "unambiguous definition of the way of computing", one of the most difficult problems of computer science that has been solved after more than 60 years of research. This is a sort of "dark corner" of computer science (though programmers are aware of it), which is far to be obvious to non-experts. Finally, I will address the relationship between the syntax of a (programming) language and the machines (computers). This contribution intends to reach a large community (which is different from the scientific one that I usually address).

Key words: to compute, writing, computer science.

Quando l'omonimo direttore di questa rivista mi ha invitato a scrivere un articolo sui linguaggi digitali, sono stato, con tutta onestà, tentato di negarglielo. La sensazione che avevo avuto era che l'oggetto di questo articolo dovesse essere la discussione sui modi di scrivere dei nuovi media. Per esempio, discernere sulle varie paroline "tvb", con un numero a

Articolo ricevuto nel marzo 2014; versione finale del maggio 2014.

piacere di "t" iniziali, oppure discutere delle innumerevoli faccine fatte con i simboli di punteggiatura, tipo ;-) e della loro evoluzione nei cosiddetti *emoticons*. Sebbene io sia un informatico, appartengo alla generazione precedente ai nuovi media ed ho sempre evitato questi espedienti comunicativi, ritenendo che scrivere o leggere "ti voglio bene" riempia di maggiore soddisfazione che il corrispondente acronimo.

Poi, mi sono chiesto: è possibile che il contributo della mia scienza (l'informatica) alla scrittura si riduca a queste cose? O forse c'è un contributo più importante che richiede una riflessione più profonda e che va oltre i modi di scrivere degli studenti.

L'informatica è la scienza che studia il calcolo (e non il calcolatore, come erroneamente la chiamano gli anglosassoni). Più precisamente è la scienza che studia il modo di calcolare.

È intorno a questa definizione che si svilupperà questo mio contributo.

Innanzitutto, cercherò di spiegare la definizione, visto che non è affatto evidente cosa significhi "modo di calcolare". Sicuramente il lettore avrà già intuito che ci sono *diversi* modi di calcolare, una faccenda che sconvolgerà i più. Per il momento, mi preme osservare che il modo di calcolare è fortemente connesso a come questo modo è descritto, ovvero scritto. Quindi la *scrittura del modo di calcolare* diventa essenziale in informatica.

Di seguito, discuterò il concetto di precisione della scrittura, ovvero di definizione inequivocabile del modo di calcolare, uno dei problemi più spinosi dell'informatica che è stato risolto dopo oltre sessant'anni di ricerca.

Infine, mi dilungherò su come la scrittura possa influenzare il modo di calcolare. Questa è una faccenda piuttosto nota alla stretta cerchia degli adepti (i programmatori informatici, sebbene, forse in molti, non ne siano davvero coscienti), ma molto poco nota ed altrettanto meno evidente ai non addetti ai lavori.

Accennerò anche alla relazione che c'è tra la sintassi di un linguaggio e le macchine (calcolatrici) in grado di riconoscere se una frase appartiene o meno al linguaggio in questione.

Evidentemente questo contributo ha l'ambizione di voler raggiungere una comunità che è differente da quella scientifica, in senso forte, a cui di solito mi rivolgo, con la speranza di essere utile alla comunità scientifica *tout court* come fonte di riflessione. Dunque non sarà un articolo specialistico, anche se mi auguro di non sacrificare troppo gli aspetti formali alla comunicazione.

1. La scrittura del modo di calcolare: i linguaggi di programmazione

Ci sono tanti *modi di calcolare*. Ad esempio, la settimana scorsa sono andato al supermercato per comprare della birra. La mia birra preferita costava 3 euro a bottiglia e ho deciso di acquistare 12 bottiglie. Ho speso 36 euro. I matematici sono usi esprimere questo calcolo con la notazione

3x12=36.

Ieri invece sono tornato nello stesso supermercato e ho deciso di prendere una sola birra, ma dopo un po' ci ho ripensato e ne ho preso un'altra. Subito dopo ho chiamato mia moglie chiedendomi di prendere un'altra bottiglia. La cosa si è ripetuta per altre nove volte. Man mano che prendevo le bottiglie di birra calcolavo il totale parziale perché temevo di non avere abbastanza soldi. I matematici esprimono questo processo con la notazione

3+3=6, 6+3=9, 9+3=12, 12+3=15, 15+3=18, 18+3=21, 21+3=24, 24+3=27, 27+3=30, 30+3=33, 33+3=36.

Ebbene, questi sono due modi (molto) differenti di calcolare la somma di 12 addendi uguali: il primo calcola direttamente il prodotto, il secondo fa via via la somma degli addendi con il totale parziale. Evidentemente i due modi, seppur differenti, sono equivalenti, nel senso che producono lo stesso risultato e si può decidere in tutta libertà se usare l'uno o l'altro.

Chiarita la presenza dell'esistenza di diversi modi di calcolare, vengo al *come* questi modi possono essere descritti. La notazione usata negli esempi di sopra è quella matematica che abbiamo appreso alle scuole elementari.

Purtroppo non è possibile utilizzare sempre questa notazione, perché i calcoli non sono sempre così semplici e perché i calcoli non sono svolti dagli uomini. Nello specifico, si vuole delegare una macchina calcolatrice a calcolare, e questa va istruita. A tal fine occorre indicare in maniera precisa tutti i *passi computazionali* che essa deve fare per espletare il suo compito. Questa indicazione precisa avviene attraverso la *scrittura dei passi* ed è chiamata *programma*.

Ad esempio, una possibile scrittura dei programmi corrispondenti ai due calcoli di sopra è:

1. prendi il primo numero
2. prendi il secondo numero
3. calcola il prodotto dei numeri presi ai passi 1 e 2
4. restituisci il valore calcolato al passo 3

1. il totale parziale è 0
2. prendi un numero
3. fai la somma del numero preso al passo 2 con il totale parziale
4. aggiorna il totale parziale al valore ottenuto al passo 3
5. ripeti i passi 2, 3 e 4 altre 10 volte
6. restituisci il totale parziale

Sebbene queste scritture siano intelligibili, o almeno lo spero, esse palesano un problema che non è affatto trascurabile: siamo proprio sicuri che tutti noi interpretiamo allo stesso modo quei passi che sono scritti nei due programmi? È evidente che una cattiva interpretazione di un passo può portare a un differente risultato finale. Potremmo ritrovarci nella *situazione anacronistica di un programma* che, quando eseguito su una macchina, ritorna un certo valore e, quando eseguito su un'altra, ritorna un valore differente.

Per evitare queste spiacevoli situazioni, è necessario che il linguaggio con cui sono scritti i passi abbia una semantica non ambigua (formale), cosa che notoriamente i linguaggi naturali non posseggono. Per inciso, si osservi che le due scritture matematiche dei miei calcoli per la birra sono inequivocabili, perché sono da tutti interpretabili allo stesso modo, avendo io utilizzato della matematica elementare.

I linguaggi con cui sono scritti i passi si chiamano *linguaggi di programmazione* e differiscono da quelli naturali proprio per avere una precisa (ossia univoca, non ambigua) semantica (formale) (cfr. Hofstadter, 1990, parte II).

Sicuramente il lettore curioso si chiederà: come si fa a dare una semantica a un linguaggio? La risposta non è poi così complicata per i linguaggi di programmazione: dire quale è il *significato di un programma* significa definire in maniera esatta *cosa fanno* i singoli passi di quel programma quando essi sono eseguiti su una macchina calcolatrice. Quindi, noi informatici definiamo una macchina calcolatrice astratta (o virtuale, se preferite) e definiamo (in maniera matematica) i passi di un linguaggio di programmazione in termini di essa. Si noti che non è possibile fare la stessa cosa per i linguaggi naturali: le frasi che noi diciamo, per fortuna, raramente sottendono dei calcoli.

Un'importante conseguenza della presenza di una semantica formale è quella di poter identificare lo stesso significato in calcoli diffe-

renti. Come ad esempio i calcoli delle bottiglie di birra all'inizio della sezione. Non c'è più posto per dietrologie sui possibili significati di frasi (o calcoli o programmi): ce n'è sempre e soltanto uno! Non voglio dilungarmi su questo aspetto che richiederebbe una lunga e approfondita disamina.

Piuttosto, mi preme sottolineare un altro aspetto della questione che è altrettanto importante. Ho appena detto che i due calcoli delle bottiglie di birra hanno lo stesso significato. Ma siamo sicuri che sia proprio così? Se per significato si intende che i calcoli producono lo stesso risultato finale, allora siamo tutti d'accordo che sono uguali. Ma c'è almeno un altro significato, che è altrettanto importante. Immaginate per un attimo che ogni passo (di calcolo) costi un euro. Se facciamo due conti, risulta che il calcolo della moltiplicazione costa un euro, quello con le addizioni costa undici euro. Non sono particolarmente parsimonioso, ma visto che posso scegliere, io calcolo con la moltiplicazione. Ritornerò su questa questione. Per il momento è appena il caso che in informatica non è importante solamente il risultato dei calcoli, ma anche come questo risultato sia stato ottenuto (cfr. Montanari, 2007). Per fare un'analogia con il linguaggio naturale: una persona prolissa può dilungarsi a esprimere un concetto che una persona normale esprimerebbe brevemente, e ciò può essere importante per un interlocutore che ha molto poco tempo.

2. La autoreferenzialità, ovvero "questo titolo è sbagliato"

Vengo a un problema molto intrigante, che spero di riuscire a spiegare in maniera sufficientemente semplice (cfr. Hofstadter, 1990).

Con i linguaggi di programmazione si possono scrivere cose molto interessanti. E questo lo si può immaginare, visto i programmi che si usano nel quotidiano quando si accende un computer. Un programma che è possibile scrivere è quello che interpreta i programmi del suo stesso linguaggio di programmazione. Non è facile da immaginare, ma, visto che la macchina che deve eseguire i passi di un programma e che abbiamo usato per dare semantica a quei passi è virtuale, non è una assurdità pensare che essa stessa possa essere definita attraverso il linguaggio di programmazione stesso. E assicuro che è davvero così.

Bene. A questo punto, si può pensare di definire una sequenza di passi che è in grado di calcolare se stessa: cioè un programma che calcola se stesso. E che significa questo programma?

È su questa domanda che per molti anni si sono arrovellati matematici, scienziati, logici e filosofi. Il problema è quello della ben nota *autoreferenzialità*, che è causa di paradossi. La frase "questo titolo è sbagliato" non può essere né vera né falsa: se fosse vera, allora il titolo non potrebbe essere sbagliato; se fosse falsa, allora il titolo sarebbe corretto. Lo stesso problema lo si ritrova in matematica, nella teoria degli insiemi: l'insieme di tutti gli insiemi è esso stesso un insieme oppure no? Questo è il paradosso attraverso il quale Kurt Gödel (1906-1978) fece crollare tutta la teoria di Gottlob Frege (1848-1925). Lo si ritrova persino nella pittura, in *Mani che disegnano* (del 1948, dove sono rappresentate due mani, ognuna delle quali assurdamente disegna l'altra) dell'artista olandese Maurits Cornelis Escher (1898-1972): in realtà tutti i quadri di Escher sono paradossali (un suggerimento: chi non li ha mai visti, li cerchi su Internet!).

Ritorno ai linguaggi di programmazione e alla questione dei programmi che calcolano se stessi. Questo problema deriva dal fatto che abbiamo definito il significato dei passi computazionali attraverso macchine che a loro volta possono essere definite attraverso passi.

Ma è proprio necessario definire i passi computazionali attraverso macchine (virtuali)? In fondo i matematici hanno fatto calcoli per millenni senza utilizzare alcuna macchina. Se invece dessimo una definizione di passo computazionale che sia avulsa dalle macchine potremmo evitare o comprendere meglio la situazione paradossale di prima.

Questa intuizione è stata la chiave di volta. Ogni passo computazionale è stato definito in termini di funzioni matematiche ed è stato necessario scoprire un modello per cui lo spazio delle funzioni abbia la stessa cardinalità dello spazio stesso. Detta così rischio di perdere anche quei pochi lettori che hanno seguito le argomentazioni fino a questo punto. Per comprendere cosa intendo dire si considerino i numeri (*naturali*) 0, 1, 2, 3 ecc. Sappiamo tutti che sono infiniti, cioè è possibile contare senza fine, un'attività molto poco interessante. Poi ci sono altri numeri, quelli con la virgola (ovvero col punto[1]), che con gli euro sono ormai molto noti: una birra di una marca meno buona (a mio avviso) costa 2,35 euro. Questi numeri, detti numeri *reali*, sono sempre infiniti ma sono "molto di più" dei numeri come 0, 1, 2, 3 ecc.

[1] Invertendo gli usi aritmetici che l'Europa continentale fa della virgola e del punto, nei paesi anglofoni il segno per separare la parte intera dalla parte decimale di un numero è infatti il punto, mentre la virgola serve a separare a gruppi di tre le cifre della parte intera (ad esempio, il numero che noi indichiamo con 27.305,12 viene scritto 27,305.12).

I matematici usano dire che la cardinalità dei numeri naturali è più piccola della cardinalità dei reali.

Ora occorre immaginare che se uno spazio è tanto grande quanto i numeri naturali, lo spazio delle funzioni su di esso è grande quanto i numeri reali. Quindi è contraddittorio identificarli, a meno che uno non cambi la definizione di funzione. Non voglio farla troppo lunga, ma negli anni Sessanta il logico Dana Scott riuscì a definire un modello per cui esso stesso e lo spazio delle sue funzioni hanno la stessa cardinalità (Scott, 1982). In questo modo fu possibile definire il significato dei passi computazionali prescindendo dalle macchine, e fu possibile comprendere che un programma che calcola se stesso è un programma che non calcola un bel niente.

3. L'efficienza della scrittura

Riprendo la questione che ogni passo computazionale ha un costo e che uno preferisce calcolare spendendo meno. Questo costo, a cui ho precedentemente associato un valore pecuniario, è in realtà quantificabile in termini di tempo. Ad esempio, ogni passo computazionale costa un secondo per valutarlo; quindi non sono stato troppo approssimativo, visto che il tempo è denaro. È evidente che uno preferisca modi di calcolare che "facciano perdere meno tempo", cioè che siano più efficienti.

La questione che mi accingo a discutere riguarda proprio l'efficienza.

Ci sono modi di rappresentare (mediante la scrittura) gli oggetti del calcolo che sottendono modi di calcolare più efficienti di altri. Quindi la scelta di come si scrivono gli oggetti è fondamentale per fare meglio i calcoli. Non è banale spiegare questa faccenda e, come dicevo nell'introduzione, i programmatori informatici sono a conoscenza del fenomeno. Lo spiego in maniera "romanzata". Non so se sia andata veramente così, e non è importante: sono sicuro di veicolare meglio i concetti in questo modo (cfr. Kaplan, 1999).

Siamo circa a metà del Medioevo, intorno all'anno 900 dopo Cristo. Nel mondo il numero zero non era stato ancora scoperto, o meglio: qualcuno lo conosceva, ma non se ne comprendeva l'utilità. Pensate che, nonostante ciò, Eratostene nel III secolo avanti Cristo aveva calcolato la circonferenza della Terra, sbagliandosi di soli 600 km, e che, più o meno nello stesso periodo, Aristarco aveva calcolato le dimensioni e le distanze del Sole e della Luna. Insomma tutto ciò che c'era da calcolare nel 900 dopo Cristo lo avevano calcolato, e lo avevano calcolato anche bene, pur non utilizzando lo zero.

Nelle corti medievali europee, come è noto, andavano di moda i tornei matematici. In pratica si chiedeva ai partecipanti di calcolare moltiplicazioni, come il nostro 3x12 (ma in verità un po' più complicate), e chi rispondeva per primo in modo corretto vinceva ricchi premi in denaro. Naturalmente tutti i matematici del tempo partecipavano a questi tornei e tentavano di vincerli. Nel 900 dopo Cristo, però, i numeri non erano scritti come li scriviamo ora, ma erano scritti in romano: 3 era scritto III, mentre 12 era scritto XII.

Che modo di calcolare si usava per fare la moltiplicazione III per XII? Ebbene si usava un modo di calcolare simile a quello delle somme successive che io ho fatto nel mio secondo esempio. Quindi si facevano undici somme di III (oppure, che è meglio, 2 somme di XII).

A un certo punto arrivò un tale con un turbante: si chiamava Al-Khwarizmi e veniva dalla Persia. Costui incominciò a vincere tutti i tornei. Era bravissimo. Inizialmente si pensò che Al-Khwarizmi fosse semplicemente più veloce nel fare gli stessi passi computazionali. In seguito, i colleghi matematici europei iniziarono a pensare che Al-Khwarizmi usasse qualche trucco (a loro ignoto) che gli consentiva di essere più efficiente nel calcolo. Dopo un po' il trucco fu scoperto, e sapete quale era? Al-Khwarizmi scriveva i numeri in maniera diversa e quel modo diverso di scrivere i numeri consentiva di fare più velocemente le moltiplicazioni. Nel modo di scrivere i numeri di Al-Khwarizmi ogni cifra da 0 a 9 aveva un peso differente a seconda della posizione in cui si trovava. In questo modo di scrivere, che è quello che utilizziamo oggigiorno, quando si scrive 33, il 3 a destra ha il peso delle unità e vale veramente 3, mentre il 3 a sinistra ha il peso delle decine e vale 30.

Si provi ad immaginare quanto sia stata rivoluzionaria questa idea per quei tempi, quando X valeva 10 in qualunque posto si trovasse. E si comprende anche l'utilità dello 0: in questa scrittura, lo 0 è necessario se si vuole scrivere 30. L'effetto di questo tipo di scrittura era che Al-Khwarizmi utilizzava modi di calcolare che gli facevano vincere tutti i tornei, che poi sono i modi che abbiamo imparato alle scuole elementari. Quando si sparse la voce che Al-Khwarizmi vinceva i tornei grazie a un nuovo modo di scrivere i numeri e di calcolare, tutti vollero "calcolare secondo Al-Khwarizmi" e ciò divenne una moda, tanto che, per contrazioni successive, il nome del povero Al-Khwarizmi fu storpiato in "algoritmi", e si volle "calcolare secondo gli algoritmi".

È andata a finire che oggigiorno il termine "algoritmo" è cruciale in informatica ed identifica appunto il "modo di calcolare tramite una

sequenza di passi". Spero che, mediante questa storiella, sia riuscito a spiegare che in informatica "il modo di scrivere" è strettamente correlato al "modo di calcolare".

Quando noi informatici programmiamo, il primo problema che affrontiamo è quello di come rappresentare gli oggetti che manipoliamo (che per fortuna non sono sempre numeri). Un buon programmatore è colui che è in grado di trovare un equilibrio tra rappresentazione degli oggetti ed efficienza dei programmi, cosa che in molti casi non è banale. Ad esempio, quando si devono organizzare i dati in un *database*, il programmatore deve scegliere se organizzarli in forma tabellare oppure in forma di lista. Nella prima organizzazione è possibile ricercare le informazioni molto velocemente (tramite un algoritmo che si chiama ricerca dicotomica, che è simile a quello che usiamo quando cerchiamo qualche parola in un dizionario). Il difetto della organizzazione tabellare è che consente di ospitare solamente un numero prefissato di elementi, e quindi è inappropriata in tutti quei casi in cui gli elementi possono crescere col tempo. In una organizzazione a lista, questo problema non si manifesta, ma v'è lo svantaggio di avere delle ricerche meno veloci.

4. La sintassi dei linguaggi di programmazione: le grammatiche di Chomsky

Finora ho trattato i linguaggi di programmazione attraverso una chiave di lettura semantica, che, a mio avviso, è quella più avvincente per le relazioni che ha col calcolo e l'efficienza. In realtà anche la sintassi dei linguaggi possiede una forte relazione con le macchine calcolatrici (e la relativa efficienza).

Un linguaggio (formale o naturale) è un insieme di frasi (ovvero di espressioni linguistiche significative) (cfr. Hofstadter, 1990; Chomsky, 1956). Queste frasi sono (formalmente) sequenze (o stringhe) di lunghezza finita costruite mediante un alfabeto finito di caratteri. Un tipico problema che si ha con un linguaggio è comprendere se una frase appartenga o meno ad esso. Per i linguaggi formali (e di programmazione, in particolare) questo problema ha soluzione ed il fatto importante è che questa soluzione è realizzata da un programma che prende in input le possibili frasi e ne verifica l'appartenenza. Tale soluzione si basa sulla proprietà che hanno i linguaggi formali di poter essere definiti mediante una collezione di regole, detta *sintassi*, attraverso la quale si generano tutte le frasi (o programmi) corrette. Cosicché ogni testo di un linguaggio di programmazione ha un'appendice con le regole sintattiche (in

questo modo esse sono di facile consultazione), ed uno dei primi giochetti che fanno gli studenti di informatica è quello di consultarla per scrivere dei programmi che, seppure strani, sono sintatticamente corretti (e quindi appartengono al linguaggio).

Le regole sintattiche dei linguaggi di programmazione seguono i formati definiti negli anni Cinquanta da Noam Chomsky (cfr. Chomsky, 1956). In realtà Chomsky aveva obiettivi molto più ambiziosi che studiare linguaggi di programmazione (che peraltro non esistevano ancora): definire un insieme di regole che consentissero di generare tutte e sole le frasi della lingua inglese. Questi studi portarono Chomsky a definire una gerarchia di "formati di regole" sempre più complesse che erano in corrispondenza con una gerarchia di macchine calcolatrici sempre più potenti. Tale risultato, di cui vi parlerò brevemente, fu il contributo che ha avuto una profonda influenza sui linguaggi di programmazione e sulla loro sintassi.

Innanzitutto, le macchine calcolatrici a cui faceva riferimento Chomsky sono dei modelli astratti che permettono di decidere se una frase appartiene o meno a un linguaggio definito da una sintassi scritta seguendo un opportuno formato. Questi modelli astratti sono oggigiorno integrati in tutti i compilatori dei linguaggi di programmazione in una maniera molto vantaggiosa per chi definisce linguaggi.

Un compilatore trasforma un programma in un linguaggio di programmazione in un programma nel linguaggio del processore di un computer. Questa trasformazione è definita utilizzando la sintassi del linguaggio di programmazione ed in parte è svolta automaticamente mediante dei programmi, gli *analizzatori lessicali e sintattici*, che prendono in input la sintassi ed implementano esattamente le macchine virtuali definite da Chomsky. La parte restante del compilatore (che è esso stesso un programma) è sviluppata *ad hoc*, a seconda delle caratteristiche del linguaggio e seguendo delle tecniche ben note.

Mi preme sottolineare che un compilatore è un esempio di sistema complesso, che può essere implementato efficientemente soltanto combinando teoria e pratica seguendo i dettami di Chomsky. Questa peculiarità è di solito assente nello sviluppo dei programmi commerciali.

Note finali

Vorrei concludere con alcune brevi considerazioni:

– i linguaggi digitali (o di programmazione) sono differenti da quelli naturali perché posseggono una semantica che consente di associare ad ogni frase (o programma) esattamente un solo significato;

– i linguaggi digitali sono differenti da quelli naturali perché sottendono un concetto di "efficienza dello scrivere", che si declina tipicamente in tempo per ottenere i risultati dei calcoli fatti.

Questi due aspetti, mentre sono un innegabile valore aggiunto quando si calcola, non è detto che lo siano nella vita quotidiana (dove si utilizzano i linguaggi naturali), sebbene, in tante occasioni sentiamo l'esigenza di una maggiore chiarezza espositiva, in particolare quando l'esposizione è verbale (e quando l'interlocutore è un uomo politico).

Mentre scrivevo questo articolo mi sono chiesto più volte se sarei stato in grado di fare degli esempi – avulsi dal calcolo – utilizzando il linguaggio naturale. Sicuramente questo tipo di esempi sarebbe stato più gradito ai lettori di questa rivista.

Sebbene io abbia esempi per le sezioni sulla semantica e sull'autoreferenzialità, non ho esempi sulla sezione che riguarda l'efficienza della scrittura. A mio avviso – ma io sono di parte – efficienza dello scrivere va declinata come "elaborazione che il testo comporta". L'efficienza della scrittura presupporrebbe quindi una metrica attraverso la quale misurare il grado di emozione che suscita un testo in un lettore. Non mi aspetto che una tale metrica si possa definire; per fortuna non c'è (e non ne intravedo) modo di misurare emozioni e sentimenti*.

Riferimenti bibliografici

Chomsky N. (1956), *Three Models for the Description of Language*, in "IRE Transactions on Information Theory" 2/3, September, pp. 113-24.

Hofstadter D. R. (1990), *Gödel, Escher, Bach: un'Eterna Ghirlanda Brillante*, Adelphi, Milano.

Kaplan R. (1999), *Zero. Storia di una cifra*, trad. it. Rizzoli, Milano 1999.

Montanari U. (2007), *Idee per diventare informatico*, Zanichelli, Bologna.

Scott D. S. (1982), *Domains for Denotational Semantics*, in *Automata, Languages and Programming*, Ninth Colloquium, Aarhus, Denmark, July 12-16, 1982 (*Lecture Notes in Computer Science*, 140), Springer Verlag, Berlin-Heidelberg-New York, pp. 577-613.

* La dott.ssa Marisa Ghionna e l'ing. Francesco Giretti hanno corretto con solerzia e pazienza le bozze iniziali di questo articolo, che sarebbe stato molto più ostico senza il loro contributo.