LINGUA UNIVERSALIS

di
Simone Martini
Alma Mater Studiorum – Università di Bologna
Dipartimento di Scienze dell'Informazione e INRIA Sophia Antipolis – FoCUS

Il tessuto costitutivo della scienza moderna è un intreccio elaborato di *metodo* e *linguaggio*. La grande fondazione seicentesca delle «sensate esperienze» e delle «hypotheses non finctae» (il *metodo*) non sarebbe possibile senza la convinzione espressa nella celebrata frase del Saggiatore galileiano, che l'universo «è scritto in lingua matematica, e i caratteri son triangoli, cerchi ed altre figure geometriche»[1] (il *linguaggio*). La lingua informa di sé una scienza e connota il suo sviluppo. La matematica è stata certamente la lingua tecnica prevalente, ma molte discipline hanno sviluppato proprie notazioni sofisticate e precise, senza le quali sarebbe impossibile *fare* quella scienza (pensiamo alla chimica[2], per fare solo l'esempio più semplice).

Nana sulle spalle di giganti, l'informatica ha sviluppato sofisticati ed eleganti strumenti linguistici per esprimere procedimenti effettivi, che chiamiamo per semplicità «linguaggi di programmazione». Si tratta di linguaggi artificiali caratterizzati da sintassi (relativamente) semplice e definita in modo canonico. Soprattutto, sono linguaggi dotati di una semantica precisa e che potremmo dire *performativa*: una frase specifica univocamente un'azione, o una sequenza di azioni, che un ipotetico esecutore può eseguire su opportuni dati. Negli anni cinquanta del secolo scorso la soluzione di un problema computazionale avviene *prima* usando il linguaggio della matematica, o il linguaggio naturale; poi, alla fine del processo risolutivo, arriva la «*codifica*» di quella soluzione nel linguaggio di programmazione scelto. In quel tempo (immensamente lontano, nella scala dell'innovazione innescata proprio da quei processi di cui stiamo parlando), il linguaggio di programmazione è un utensile tecnologico, per realizzare un manufatto concepito con altra strumentazione concettuale; soprattutto, per ciò che ci interessa qui, concepito con altri linguaggi.

Il panorama concettuale cambia radicalmente a partire dagli anni sessanta, e poi sempre più velocemente nei decenni seguenti, con l'introduzione dei linguaggi di programmazione cosiddetti «ad alto livello». Se i loro predecessori erano definiti a partire dall'esecutore (cioè dalle caratteristiche del calcolatore elettronico che avrebbe eseguito le frasi), i linguaggi ad alto livello sono progettati a partire dai loro utenti (gli informatici, negli anni sessanta) e dai problemi che de-

Il tessuto costitutivo della scienza moderna è un intreccio elaborato di metodo e linguaggio

---

1. Galileo Galilei, *Il saggiatore*, cap. 6. Ricciardi, 1953.

2. Antoine Lavoisier inizia il suo *Traité élémentaire de chimie*, Paris 1789 (che avrebbe avuto grande influenza nell'affermarsi della notazione moderna) proprio con una citazione da É. de Condillac sul ruolo del linguaggio nella strutturazione del pensiero.

vono essere risolti. Svincolati dall'esecutore[3] i linguaggi di programmazione divengono notazioni sofisticate e sintetiche per descrivere i problemi e le loro soluzioni. Essi non sono più l'utensile che interviene al termine di un processo di elaborazione condotto con altri strumenti: l'informatico inizia a pensare usando il proprio linguaggio, anzi *i propri* linguaggi, vista la loro pluralità. E subito, ovviamente, scopre quanto la notazione influenzi il pensiero, lo possa guidare, aiutare e rendere più economico[4].

I linguaggi di programmazione sono uno dei contributi originali dell'informatica in quanto scienza. Nella loro varietà e diversità[5], forniscono ben più che una notazione concisa ed elegante per descrivere algoritmi. Fare *Problem Solving* significa decomporre, ristrutturare, risolvere sottoproblemi e ricomporre, poi, le loro soluzioni. In ogni scienza si fa *Problem Solving*; lo si fa anche nella vita di tutti i giorni. Ma l'informatica porta qui un suo contributo originale: mette a disposizione *strumenti linguistici* progettati affinché ciò sia possibile e, per quanto possibile, semplice. Cioè evocativo, sintetico, economico; talvolta anche bello. Un linguaggio di programmazione moderno è soprattutto questo: un insieme di meccanismi linguistici che permettono di esprimere in modo sintetico la decomposizione di un problema in sottoproblemi, la soluzione indipendente di ciascun sottoproblema, la ricomposizione, poi, dei pezzi nella soluzione del problema originale. In questo processo, l'aspetto strettamente operativo (le «istruzioni per risolvere un problema») è meno importante di quello di suddivisione e ricomposizione. È qui che i linguaggi di programmazione moderni si differenziano in modo sostanziale dai loro predecessori (e tra loro). Indichiamo genericamente con *meccanismi di astrazione* quei costrutti che assolvono a questa forma di *Problem Solving*. «Astrarre» significa nascondere: astraendo da alcuni dettagli concreti di più oggetti, si riesce a far emergere con più chiarezza un concetto comune. La descrizione scientifica di un fenomeno (naturale, artificiale, fisico ecc.) non è costituita dall'insieme di *tutti* i dati relativi a quel fenomeno. Altrimenti sarebbe come una carta geografica in scala 1:1, precisissima ma inutile.

Per catalogare o anche solo descrivere i meccanismi di astrazione dei linguaggi di programmazione ci vorrebbe un intero manuale[6]. Ci accontenteremo di ricorda-

Svincolati dall'esecutore i linguaggi di programmazione divengono notazioni sofisticate e sintetiche per descrivere i problemi e le loro soluzioni

---

3. Il rapporto con l'esecutore viene brillantemente risolto mediante la realizzazione di specifici programmi (i *compilatori*) che traducono i linguaggi ad alto livello nei linguaggi di più basso livello che saranno a loro volta eseguiti da un calcolatore.

4. «I linguaggi di programmazione, essendo progettati allo scopo di fornire direttive ai calcolatori, offrono importanti vantaggi in quanto strumenti del pensiero. Non solo sono universali (*general-purpose*), ma sono anche eseguibili e non ambigui. L'eseguibilità implica che è possibile utilizzare i calcolatori per effettuare esperimenti su idee espresse in un linguaggio di programmazione, e la mancanza di ambiguità rende possibile precisi esperimenti di pensiero.» K. Iverson, *Notation as a tool for thought*, Comm. of ACM, Vol. 33(8) pp. 444-465 (1980).

5. S. Martini, *Elogio di Babele*, Mondo Digitale, no. 2 – giugno 2008, 17-23.

6. Ci sia consentito rimandare a M. Gabbrielli, S. Martini, *Linguaggi di programmazione – principi e paradigmi*, (2a ed.) McGraw-Hill, 2011.

re che vi sono meccanismi di astrazione *sui dati*, e meccanismi di astrazione *sul controllo*. L'astrazione sul controllo[7] nasconde i dettagli procedurali e permette di suddividere un problema in sottoproblemi. A ogni sottoproblema è assegnata una diversa astrazione (per esempio, per usare qualche termine tecnico: una procedura, o un modulo). Per *usare* quella soluzione non sarà necessario conoscerne i dettagli risolutivi, ma solo le modalità convenzionali con le quali quell'astrazione può essere *invocata*. La soluzione del problema nella sua globalità è poco più che l'assemblaggio delle soluzioni dei sottoproblemi, di cui a questo punto non è necessario conoscere come ottengono il loro risultato, ma solo come invocarle. Questo semplice schema concettuale necessita tuttavia di sofisticati meccanismi linguistici per renderlo sufficientemente generale e flessibile. Si vogliono dare soluzioni *generali* a determinati problemi, e invocare poi quelle soluzioni su *istanze* specifiche. Oppure, si vuole impedire all'invocazione di una soluzione di poter accedere a parametri privati della soluzione stessa. È qui che sta, tecnicamente, il contributo proprio dell'informatica al *Problem Solving*: mentre le altre discipline usano il linguaggio naturale per descrivere questi parametri, i linguaggi di programmazione hanno identificato una batteria di costrutti pensati precisamente per questo. E che possono *verificare* (e, quindi, *garantire*) che questi vincoli di visibilità, o di ricomposizione corretta, sono soddisfatti.
Il controllo (cioè l'aspetto procedurale di una soluzione) agisce su opportuni *dati*, che sono descritti mediante altri, specifici meccanismi di astrazione[8]. Si usa dire, con un certo disprezzo, che i calcolatori manipolano solo sequenze di cifre binarie. Il che è un'ovvietà per la macchina *fisica*, ma non certo per le macchine *astratte* dei linguaggi di alto livello. Questi permettono la definizione di classi di dati sofisticate, ciascuna composta da una collezione di valori e dotata di un insieme opportuno di operazioni[9]. Chi usa il linguaggio, quindi, può direttamente descrivere i dati del proprio problema nel modo più congeniale e più vicino (o più utile) al suo modo di pensare. Numeri, collezioni, liste, alberi, grafi, pagine web, «bottoni»: tutti sono esprimibili come un'opportuna astrazione e, quindi, manipolabili con naturalezza mediante le operazioni ovvie che si applicano, rispettivamente, ai numeri, alle collezioni, alle liste, ecc. Se, poi, il linguaggio è dotato di opportune caratteristiche[10], è possibile impedire che le operazioni introdotte, per esempio, per i numeri interi, possano essere applicate anche alle sequenze di bit. *Impedire* significa qui, ancora una volta, che i meccanismi lin-

---

7. Meccanismi di astrazione sul controllo sono le procedure, le funzioni, il passaggio dei parametri, alcune forme di moduli, le eccezioni, ecc.
8. Meccanismi di astrazione sui dati sono i tipi, i tipi astratti, alcune forme di moduli, alcuni meccanismi di visibilità, ecc.
9. Tecnicamente questa è la definizione di *tipo di dato*: una collezione di valori omogenei ed effettivamente presentati dotata di un insieme di operazioni.
10. Se, cioè, è «sicuro rispetto ai tipi» (*type safe*), o ha «tipi di dato astratti».

guistici rilevano e segnalano come erronea una situazione come quella ipotizzata. Oppure, dualmente, è possibile permettere che le operazioni introdotte per i grafi siano applicate anche agli alberi (visto che ogni albero è anche un grafo), ma non viceversa (dal momento che vi sono grafi che non sono alberi)[11]. Opportuni meccanismi di *visibilità*, infine, intervengono a permettere, o impedire, che i dettagli di una collezione siano visibili (e quindi sfruttati) al di fuori della definizione della collezione stessa.

In informatica, la cui essenza primaria risiede nell'immateriale dell'*espressione linguistica* del calcolo e dell'interazione, davvero la *forma è sostanza*: il modo di esprimere un concetto è altrettanto importante del concetto espresso. E questa forma è influenzata in modo cruciale dal linguaggio che scegliamo per esprimerla.

Possiamo finalmente vedere le implicazioni del discorso che stiamo facendo. Se l'importanza primaria dei linguaggi (cosiddetti) di programmazione sta nel loro contributo all'espressione del pensiero, non solo di quello matematico-computazionale, è evidente che essi sono in realtà una feconda e potente *lingua universalis*, che può essere applicata ben al di là della realizzazione di programmi («applicazioni»). Questa consapevolezza si è affermata in anni recenti ed è esplicitata nell'espressione «*computational thinking*»[12], con la quale si indica l'uso di tecniche informatiche per la *descrizione* e *soluzione* di problemi di un vasto spettro di discipline, non solo scientifiche, ma anche sociali o letterarie. Ben al di là della famosa espressione *Computer science is no more about computers than astronomy is about telescopes*[13], con un certo orgoglio gli informatici rivendicano oggi di possedere (e saper insegnare) linguaggi, metodologie e risultati che servono a fare, e far meglio, anche molte altre discipline. All'aspetto linguistico (il solo che qui abbiamo potuto trattare nello spazio di un articolo), si associano aspetti algoritmici (p.e., teorie generali di soluzione per determinate classi di problemi), di complessità (p.e., informazioni precise sul costo di una soluzione in termini di risorse consumate; o sull'impossibilità di avere una soluzione efficiente), di rappresentazione della conoscenza (p.e., tecniche per la rappresentazione di grandi quantità di informazioni tra loro collegate in modo non banale), di interazione (p.e., tecniche per la gestione di processi paralleli o concorrenti sull'utilizzo di alcune risorse), di sicurezza (p.e., tecniche per la gestione di politiche di accesso a

In informatica, la cui essenza primaria risiede nell'immateriale dell'espressione linguistica del calcolo e dell'interazione, davvero la forma è sostanza

---

11. Possono intervenire qui meccanismi sia di *compatibilità* (un tipo T è compatibile col tipo *S* se un valore di *T* può essere usato laddove sarebbe atteso un valore di *S*) sia di *ereditarietà*, per riutilizzare le operazioni precedentemente definite su altre collezioni.
12. Jeannette M. Wing, *Computational Thinking*, Comm. of ACM, Vol. 49(3) pp. 33-35 (2006).
13. E.W. Dijkstra, premio Turing 1972. Il premio Turing, il più importante riconoscimento internazionale per un informatico, è attribuito ogni anno dall'associazione professionale degli informatici americani (ACM).

risorse condivise con diverso livello di autorizzazione), e così via. Tutte questioni, come dovrebbe essere esser chiaro dagli esempi portati, che trovano applicazioni anche (e vorremmo aggiungere: soprattutto) lontano dall'informatica. Il *computational thinking* non è pensare come un calcolatore. È un modo astuto e intelligente di pensare (influenzato dalla ricerca e dagli strumenti dell'informatica) in modo umano, che permette di gestire la complessità e l'interazione in modo coerente e semplice. A questa già importante caratteristica si aggiunge la possibilità di *eseguire* le soluzioni realizzate: *computational* significa (anche) che possiamo sfruttare la potenza di calcolo dei processori e costruire sistemi innovativi, le cui potenzialità e funzionalità rispondono all'innovazione del nostro *pensiero*.

Quell'universo «scritto in lingua matematica» è *anche* scritto in una lingua computazionale, l'uso della quale ci permette di raggiungere livelli di espressione ogni giorno più sofisticati.

Il *computational thinking* è un modo astuto e intelligente di pensare (influenzato dalla ricerca e dagli strumenti dell'informatica) in modo umano, che permette di gestire la complessità e l'interazione in modo coerente e semplice