

FONDAMENTI DI INFORMATICA

Corsi di Laurea in Ingegneria Elettrica,
Ingegneria Gestionale, Ingegneria Meccanica

Università degli Studi di Firenze

A. A. 2001-2002

APPUNTI DELLE LEZIONI

Prima parte

Prof. Alessandro Fantechi

Introduzione

Anche se in molte scuole superiori si insegna ad usare il calcolatore, il livello di approfondimento degli insegnamenti di informatica impartiti varia molto, dal semplice utilizzo come video scrittura e navigazione in Internet, fino alla progettazione di programmi complessi e loro programmazione, anche su calcolatori diversi dal classico PC, come avviene in alcuni Istituti Tecnici.

In questo corso si presuppone che lo studente abbia una conoscenza sommaria del calcolatore, quale quella che gli studenti odierni possono avere appreso nella scuola superiore o anche semplicemente a casa, data l'attuale diffusione di Personal Computers.

Si può quindi supporre che il lettore sia abituato all'aspetto del computer come quello di un oggetto di dimensioni limitate, provvisto di un monitor e di alcuni dispositivi che permettono di operare su di esso, quali tastiera e mouse. Questo è in realtà solo uno dei tanti modi in cui i computer pervadono ormai la nostra vita quotidiana: basti pensare al computer che gestisce un cellulare – che pure ha in comune con il PC un monitor, una tastiera, e spesso un dispositivo di puntamento simile a un mouse – oppure alle decine di computer che equipaggiano una automobile di alte prestazioni, o a quello che regola il funzionamento di una moderna lavatrice.

Sul monitor, una volta acceso il PC e terminata la fase di *start-up* o *bootstrap*, appare la metafora di una scrivania (*desktop*) su cui compaiono cartelle, documenti contenuti nelle cartelle ed altri oggetti. Con movimenti del mouse o pressione di tasti della tastiera, è possibile compiere operazioni su questi oggetti, che spesso costituiscono metafore di operazioni comuni su oggetti reali. Così, ad esempio, una operazione di “apertura” (doppio click) effettuata su un cartella permetterà di mostrarne i documenti in essa contenuti, mentre la stessa operazione effettuata su un documento permetterà di visualizzarne il contenuto.

Altra semplice operazione che si può fare su un oggetto visibile sul desktop è quello di spostarlo, trascinandolo con il mouse. In questo modo si può spostare un documento da una cartella ad un'altra, o spostare un documento nel cestino, decretandone così la cancellazione.

Nel sistema operativo Windows, il sistema più comune su un moderno PC, è possibile, una volta selezionando un oggetto con il mouse, conoscere tutte le operazioni immediatamente possibili su di esso con la pressione del tasto destro del mouse: appare un menù che lista tutte le operazioni eseguibili sull'oggetto, alcune delle quali possono apparire di colore grigio invece che nero e non sono selezionabili: segno che le operazioni possibili non sono sempre le stesse, ma dipendono dallo **stato** in cui si trova l'oggetto. L'ultima operazione del menù è solitamente l'operazione “Proprietà”, che visualizza un insieme di informazioni relative all'oggetto, come ad esempio la sua occupazione di memoria, misurata in Byte e suoi multipli (Kilobyte, Megabyte).

Delle operazioni eseguibili su un oggetto, possiamo notare che alcune (come l'operazione “Proprietà”) non hanno lo scopo di modificare l'oggetto, ma solo di visualizzarlo (in tutto o solo in alcune informazioni), altre hanno lo scopo di modificarne l'apparenza sul video (come l'operazione di spostamento), altre hanno lo scopo di modificare l'oggetto (ad esempio quando apriamo un documento Word per correggerne una frase), altre ancora di eliminare o creare oggetti o copie degli oggetti.

Formalizzazione del Desktop

Quindi il desktop potrebbe essere formalizzato come un insieme di oggetti:

$$\text{Desktop} = \{\text{Oggetto}_1, \text{Oggetto}_2, \dots, \text{Oggetto}_n\}$$

Ogni oggetto possiede una serie di *attributi*, o *proprietà*, ad esempio la posizione della sua icona sul desktop, il suo nome, la sua dimensione, il suo contenuto, il suo stato, ecc...

risoluzione dello schermo, si possono vedere invocando l'operazione "Proprietà" sul desktop stesso e scegliendo "Impostazioni". Una tipica risoluzione è 1024 x 768, intendendo una matrice di pixel di 768 righe e 1024 colonne. Ogni pixel ha un colore, ogni colore può essere rappresentato da un numero che dà la posizione di quel colore in una scala di colori: negli schermi monocromatici si avrà o un colore bianco o nero, rappresentati dai numeri 1 e 0 rispettivamente, o 16 livelli di grigi, numerati da 0 a 15; negli schermi a colori si possono avere 256, 65356, o 16,8 milioni di colori (vedere ancora le Impostazioni del desktop).

Per capire come un colore può essere rappresentato da numeri, si può anche osservare la palette dei colori di Word, dove ogni colore può essere ottenuto variando uno di 6 parametri diversi che assumono valori tra 0 e 255.

Supponendo quindi che un colore sia rappresentato da un numero in un intervallo finito dei naturali, il desktop può essere formalizzato come una matrice 1024 x 768 di naturali (per le usuali convenzioni sulle matrici, si dovrebbe dire piuttosto una matrice di 768 x 1024 naturali). Questa matrice può fornire anche la scala per il dominio delle posizioni prima considerato, quindi anche il dominio stesso delle posizioni può essere l'intervallo di \mathbb{N}^2 : $[(0,0), (767,1023)]$

L'operazione di spostamento che abbiamo considerato, deve quindi, oltre a modificare come abbiamo visto le coordinate della posizione dell'oggetto, anche modificare l'apparenza del desktop, in modo che si veda l'icona dell'oggetto spostarsi dalla posizione originaria alla nuova posizione (supponiamo di non considerare la visualizzazione delle posizioni intermedie occupate dall'icona durante il trascinamento: per far questo basta considerare lo spostamento non come una unica operazione, ma come una sequenza di operazioni di spostamento elementari).

Occorre perciò innanzitutto considerare una icona come una piccola matrice, ad esempio di 30 x 25 pixels, e di considerare come posizione dell'oggetto le coordinate della posizione del suo angolo inferiore sinistro.

Indicando con Desktop(i,j) il valore del colore del desktop nella posizione i,j del desktop, e usando una analoga notazione per l'icona, lo spostamento di un oggetto dalla posizione (x_0, y_0) a quella (x_1, y_1) ottiene quindi la seguente trasformazione sul desktop:

$$\begin{aligned} \forall 0 \leq i \leq 29, 0 \leq j \leq 24, \\ \text{Desktop}(x_1+i, y_1+j) &\leftarrow \text{Icona}(i,j) \\ \text{Desktop}(x_0+i, y_0+j) &\leftarrow \text{Sfondo}(x_0+i, y_0+j) \end{aligned}$$

dove la seconda riga serve a ripristinare lo sfondo del desktop nella zona prima occupata dall'icona spostata. Per questo si suppone esistere un'altra matrice di 768 righe e 1024 colonne che rappresenta lo sfondo del Desktop.

Abbiamo finora visto come una semplice operazione sull'oggetto possa comportare sia degli effetti sull'oggetto stesso, sia degli effetti sulla visualizzazione dell'oggetto. Questi due effetti sono in genere strettamente legati tra di loro, proprio in quanto le moderne metafore introdotte dalle interfacce grafiche sono quelle che permettono di eseguire operazioni sugli oggetti e di verificarne visualmente gli effetti.

Altri esempi di ambienti operativi

Possiamo notare che lo stesso tipo di metafora che abbiamo brevemente discusso per quanto riguarda il desktop si estenda ad altri *ambienti operativi*. Se apriamo un disegno Autocad, ad esempio, possiamo osservare che è possibile selezionare ogni oggetto del disegno, e con il tasto destro scegliere una delle operazioni possibili su quell'oggetto. In questo caso è ancora più evidente come le operazioni agiscono attraverso trasformazioni geometriche di oggetti del disegno: si pensi al caso dell'operazione di rotazione per cui è possibile indicare direttamente in gradi l'angolo di

rotazione desiderato. Si noti come sia visibile (in basso a sinistra) in ogni momento la coppia di coordinate della posizione del mouse sul foglio da disegno (che NON corrisponde esattamente, ma è proporzionale, alle coordinate della posizione del mouse sullo schermo in termini di pixel).

Se invece apriamo un documento Word, le operazioni possibili sono quelle di manipolazione di testi. In questo caso è meno immediato pensare a queste operazioni come operazioni matematiche, ma possiamo sempre pensare ad una operazione di inserimento di una parola in questi termini:

inserimento: Testo \times Parola \rightarrow Testo

anche se, per ora, non sappiamo come *rappresentare* un testo in modo matematico. Ma ci siamo riusciti con delle immagini, quindi non dovrebbe essere così difficile!

Sia Autocad che Word presentano degli *ambienti operativi* diversi dal Desktop, in cui si hanno a disposizione una serie di oggetti e una serie di operazioni, alcune anche molto complesse, come ad esempio l'ordinamento in ordine alfabetico di parole. Autocad e Word sono dei *programmi* (rispettivamente di disegno e di videoscrittura), che possono essere *invocati* dal Desktop, semplicemente aprendo un documento da essi generato, o esplicitamente con i sottomenu del menu Start. In generale, a parte alcune eccezioni, dall'ambiente Word non è possibile invocare Autocad e viceversa, ma bisogna passare per il Desktop o per i menu ad esso associati.

In effetti, il desktop si configura come l'interfaccia con cui accedere ad ogni oggetto o programma esistente nel sistema, come pure alle risorse hardware (dischetti, CD, etc...). E' cioè l'interfaccia del cosiddetto *Sistema Operativo*, programma che permette la gestione di tutte le risorse hardware e software del sistema. E' importante a questo punto distinguere tra interfaccia utente grafica e sistema operativo: il desktop è l'interfaccia che, mediante la metafora della scrivania, permette di accedere alle funzionalità del sistema operativo. Ma tutte queste funzionalità sono accessibili anche in modo testuale (da "linea di comando") attraverso il "prompt MS-DOS", semplicemente digitando un comando: ovviamente occorre conoscere quali sono i comandi da digitare, e anche avere informazioni sugli oggetti presenti nel sistema non è così semplice in ambiente DOS come andare a cercarli sul Desktop con il mouse. I vecchi sistemi operativi (ad esempio VAX-VMS, ma anche lo stesso MS-DOS sui primi PC, o i primi sistemi Unix) non avevano interfaccia grafica, ma solo interfaccia a linea di comando. Macintosh (MacOS) ha sempre avuto solo l'interfaccia grafica per interagire con il sistema, mentre Windows è nato più come interfaccia grafica a MS-DOS che come vero sistema operativo, e solo successivamente si è evoluto in un "vero" sistema operativo. Linux è un sistema Unix con interfaccia a linea di comando, che però è equipaggiato da alcuni "window managers" (Gnome, KDE) che forniscono un'interfaccia grafica paragonabile a quella di Windows.

Un esempio di controllo di un processo fisico

Consideriamo invece un calcolatore che gestisca un processo fisico, ad esempio un piccolo calcolatore (tali oggetti vanno sotto il nome di *microcontrollori*) che controlla il funzionamento di un semplice robot capace di spostare oggetti posti su un piano di lavoro (vedi figura sottostante, che lo rappresenta schematicamente in pianta).

La trave orizzontale, che si muove al di sopra di travi laterali di appoggio per mezzo dei due motori M1, M2, porta un carrello capace di spostarsi lungo la trave per mezzo del motore P3.

Al di sotto del carrello vi è un elettrocalamita capace di prelevare un oggetto (che quindi si considera come composto, anche parzialmente di materiale ferroso) una volta che il carrello si è portato sulle coordinate dell'oggetto, e poi di rilasciarlo in una nuova posizione.

Vediamo come si può formalizzare questa operazione,

Il piano di lavoro può ancora essere formalizzato come un insieme di oggetti:

Desktop = {Oggetto₁, Oggetto₂,.....Oggetto_n}

Ognuno degli oggetti sarà rappresentato ancora una volta da una coppia di coordinate: quindi l'operazione di spostamento dovrà modificare questa coppia di coordinate:

spostamento (ogg, x₁, y₁) =
ogg.x ← x₁
ogg.y ← y₁

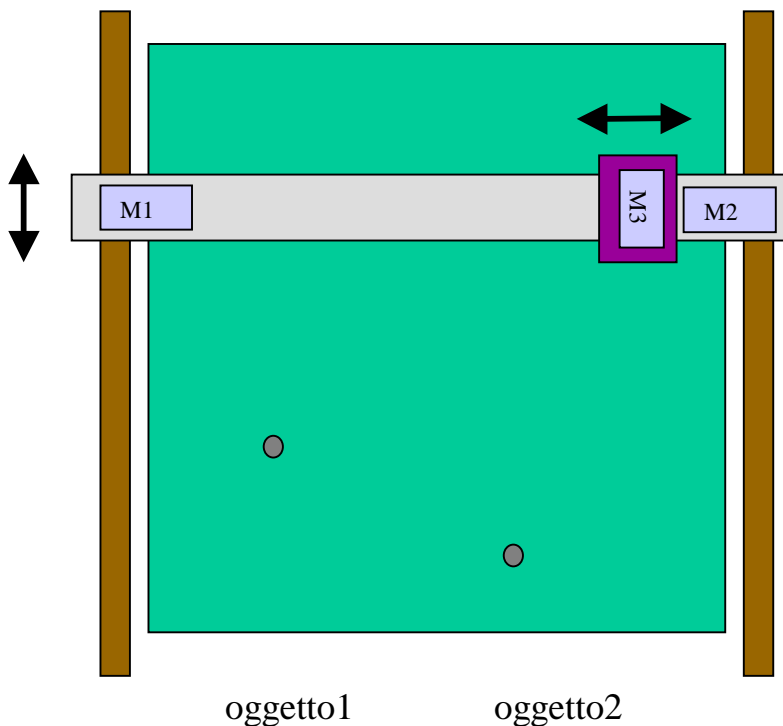
Questo “spostamento logico” sarà però accompagnato da uno “spostamento fisico” effettivo che nel passaggio da (x₀, y₀) a (x₁, y₁) esegue le seguenti operazioni:

attiva calamita
aziona M1 e M2 per $(y_1 - y_0) / V_{M1}$ secondi
aziona M3 per $(x_1 - x_0) / V_{M3}$ secondi
disattiva calamita

(dove si è supposto che M1 e M2 abbiano la stessa velocità V_{M1} ; inoltre, avremmo dovuto tener conto della direzione di avanzamento dei motori, che dipende dal segno della differenza $x_1 - x_0$ per M3 e $y_1 - y_0$ per M1 e M2).

Il calcolatore che controlla il nostro robot dovrà quindi essere in grado di compiere queste operazioni elementari, e di agire sul mondo esterno (calamita e motori) tramite un'opportuna *interfaccia* (cioè una circuiteria che permette di inviare comandi a tali elementi esterni).

In generale, un calcolatore che controlla un processo fisico deve essere dotato di una interfaccia che permette di ricevere valori riguardanti misure fisiche (temperatura, pressione, ecc.) da *sensori* (manometri, termometri, ecc..) applicati al processo e di inviare comandi agli *attuatori* (motori, elettrovalvole, ecc..).



Architettura a livelli di un calcolatore

Ritornando ad una visione più convenzionale del calcolatore, stiamo quindi cominciando a vedere come in esso si sovrappongono vari livelli a cui si può considerare un approfondimento delle sue funzionalità: abbiamo incontrato il livello del sistema operativo, e sopra di esso il livello dell'interfaccia grafica, e abbiamo visto il livello dei programmi applicativi. E, sicuramente, c'è il livello dell'hardware, cioè dei componenti elettronici che fisicamente costituiscono il computer.

Questo ragionamento a livelli si estende a tutti i settori dell'informatica: in genere, quando si tratta di un argomento di informatica, o quando si cerca di capire le cause di un apparente malfunzionamento, è importante porsi al livello giusto.

Quindi un calcolatore deve essere in grado di gestire svariati livelli; dovrebbe risultare però già chiaro che il calcolatore, come dice il suo nome, è principalmente uno strumento per il calcolo automatico: da qui nasce la necessità di formalizzare matematicamente ogni oggetto esistente all'interno del calcolatore, necessità di cui abbiamo già visto qualche esempio di realizzazione in precedenza.

Sarà quindi necessario anche formalizzare l'architettura a livelli tipica di un calcolatore: senza scomodare complessi strumenti matematici che sono stati studiati proprio a questo scopo, tenteremo di dare alcuni elementi alla base di tale formalizzazione.

Distinzione tra specifica di una operazione e algoritmo di implementazione

Proviamo a porci tre tipici problemi di natura matematica, che quindi immaginiamo di saper formalizzare facilmente:

- 1) Dato un insieme di elementi, trovarne il massimo
- 2) Dato un vettore (cioè una sequenza di elementi ordinati secondo un indice).
- 3) Trovare le radici reali di una equazione di secondo grado

In effetti i tre problemi si possono formalizzare come segue:

- 1) Dato un insieme di elementi I , trovare $x \in I$: $x > y$, $\forall y \in I$, $y \neq x$
- 2) Dato un vettore V di elementi (indicati ognuno con V_i , con $1 \leq i \leq n$), costruire il vettore V' tale che:

- i) $\forall i \exists j : V'_i = V_j$
- ii) $\forall j \exists i : V'_i = V_j$
- iii) $\forall i \forall j : i < j \Rightarrow V'_i < V_j$

Dove si suppongono i e j variare in $(1..n)$, e la i) e la ii) indicano che V' contiene tutti e soli gli elementi di V , mentre la iii) indica che V' è ordinato.

- 3) Dati tre coefficienti reali a, b, c , trovare quei valori reali x per cui: $a \cdot x^2 + b \cdot x + c = 0$

Si può notare come nelle precedenti formalizzazioni abbiamo definito i problemi come oggetti matematici (che chiameremo operazioni) in modo totalmente astratto, che non indica in nessun modo come poter calcolare l'operazione definita.

In termini informatici, chiameremo *specifica* una definizione astratta, ma precisa, di questo tipo, che definisce *cosa* fa una operazione ma non *come* lo fa.

Si può notare che in alcuni casi una specifica formale già definisce anche *come* fare una operazione: prendiamo il caso della risoluzione di una equazione di primo grado del tipo $x + b = 0$; possiamo direttamente definire questa operazione come quella che, dati due reali a e b , restituisce il reale $x = -b$. In questo caso la specifica che abbiamo dato dell'operazione indica anche *come* l'operazione si può risolvere in termini di semplici operazioni aritmetiche (si parla talvolta di

specifiche che hanno questa caratteristica come di *specifiche costruttive*, mentre di quelle che si limitano a dichiarare cosa fa una funzione come *specifiche dichiarative*).

Notiamo tuttavia che abbiamo considerato un caso particolare e che la stessa cosa non possiamo fare per il problema 3), dove la struttura dell'equazione considerata è più complessa. A questo proposito, possiamo dire che conosciamo già un metodo risolutivo, dato dalla nota formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

questa formula ci dice quali operazioni dobbiamo fare per calcolare le radici dell'equazione a partire dai suoi coefficienti, supponendo di saper calcolare le quattro operazioni aritmetiche sui reali, più la radice quadrata.

Ad esempio possiamo prima calcolare il prodotto $4ac$, poi il quadrato di b , sottrarre il primo al secondo, calcolarne la radice quadrata, calcolare i due risultati della somma e della sottrazione di questa dall'opposto di b , e infine dividere i risultati per $2a$.

Questa sequenza di operazioni elementari descritte a parole, sappiamo non essere l'unica sequenza possibile per calcolare le radici dell'equazione, per effetto delle proprietà commutative e associative di alcune delle operazioni elementari. Inoltre, sappiamo che la formula non dice tutto sulla risoluzione del problema, perché occorre verificare che a sia diverso da 0, in caso contrario ci si riconduce alla risoluzione di un'equazione di primo grado, con una formula diversa, altrimenti occorre anche verificare che il risultato sotto radice sia positivo, perché altrimenti non si hanno radici reali. Non basta perciò in questo caso definire una sequenza *lineare* di operazioni elementari, ma c'è la necessità di *condizionare* il *flusso* delle operazioni da svolgere a seconda del valore di certi risultati intermedi. Quindi la sequenza di operazioni da svolgere non sarà più lineare, ma potrà avere diversi *rami* alternativi. La nozione di semplice *composizione di funzioni* della matematica, che definisce un ordine sequenziale di applicazione delle funzioni non basta quindi più.

Inoltre, soprattutto in considerazione della natura ramificata del flusso delle operazioni da svolgere (che chiameremo *flusso di controllo*), sarà necessario avere a disposizione delle cosiddette *zone di memoria*, siano essi fogli di carta, porzioni di lavagna, scomparti della nostra mente, o porzioni della memoria di un calcolatore, in cui scrivere, cioè memorizzare, risultati intermedi.

Nasce così il concetto di *algoritmo*, cioè di un metodo risolutivo per un problema, che utilizzando la capacità di compiere operazioni elementari e la capacità di memorizzare risultati intermedi, definisce l'ordine delle operazioni da compiere.

Nozioni di algoritmo, linguaggio di programmazione, programma

Da F. Luccio: *La struttura degli algoritmi*:

"...che cosa si intende per algoritmo. Storicamente la parola *algorismus* nasce nel latino medievale per assonanza con il nome del matematico persiano al-Khuwarizmi, autore nel nono secolo di un famoso testo di algebra. In *algorismus* vi è un superamento dinamico del concetto di formula, attraverso la concatenazione di più formule derivate l'una dall'altra mediante regole di trasformazione. (Non sfuggirà come il meccanismo fosse già presente nella matematica antica, ove la dimostrazione euclidea di un teorema si spezza in una serie finita di trasformazioni che conducono da un insieme di assiomi all'affermazione di verità dell'enunciato). Un algoritmo è dunque una successione di azioni, o passi, che determinano la risoluzione di un problema. Pur se la descrizione dell'algoritmo ha lunghezza finita, essa potrebbe specificare che alcune operazioni sono ripetute illimitatamente..."

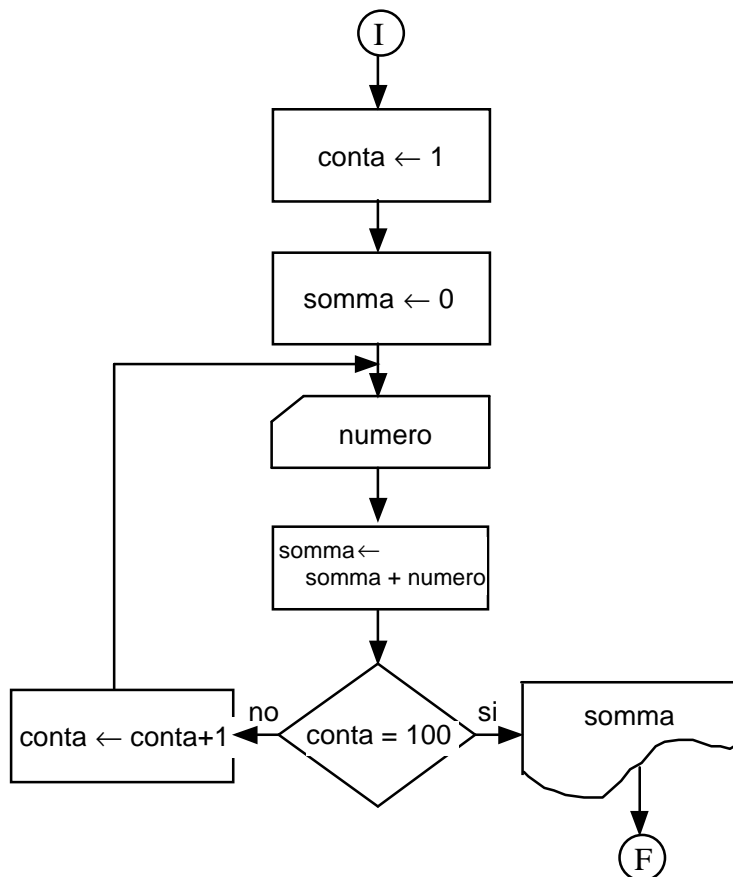
L'algoritmo è dunque una descrizione del procedimento di risoluzione del problema, attraverso una sequenza di passi elementari. Un algoritmo può essere scritto in un linguaggio che riesca a

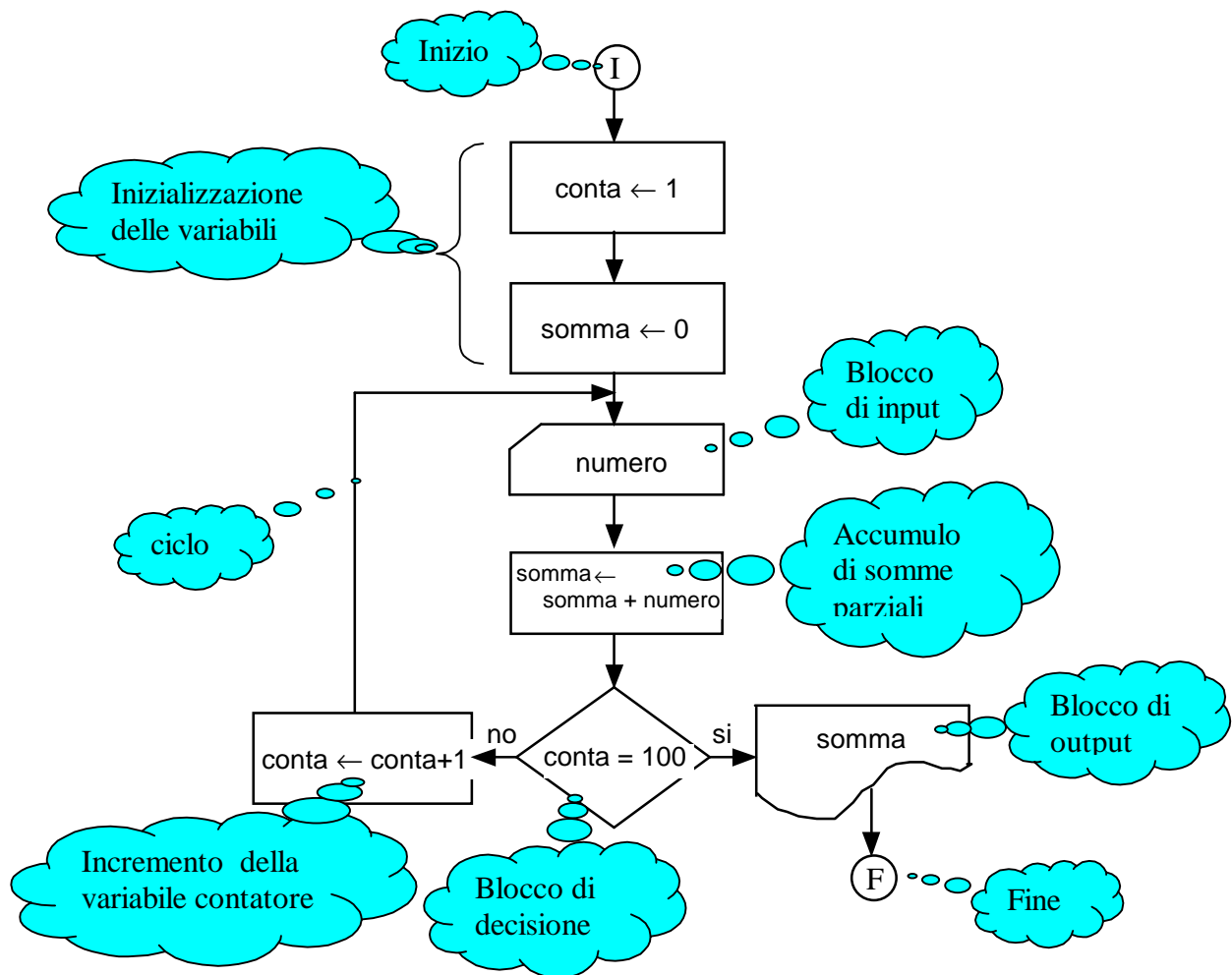
specificare esattamente quali sono i passi elementari e qual'è il loro ordine; si usano a questo scopo il linguaggio naturale, il linguaggio della matematica, o dei linguaggi grafici quali i diagrammi di flusso.

Quando un algoritmo sia descritto in un linguaggio di programmazione, cioè un linguaggio che un calcolatore può eseguire, si parla di *programma*. Un programma è quindi un'implementazione di un algoritmo per un esecutore automatico qual è un calcolatore. Un linguaggio di programmazione sarà perciò in grado di specificare dei passi elementari eseguibili dal calcolatore, e l'ordine con cui questi passi vengono eseguiti (controllo del flusso) attraverso opportune strutture di controllo che permettono di specificare, ad esempio, che una sottosequenza di passi elementari viene ripetuta finché non si verifica una certa condizione.

La notazione dei diagrammi di flusso per la descrizione degli algoritmi

Nella seguente figura introduciamo la notazione dei diagrammi di flusso (o diagrammi a blocchi) per descrivere algoritmi, con un esempio di algoritmo che somma iterativamente 100 numeri che vengono via via forniti all'algoritmo. I blocchi rettangolari rappresentano le operazioni elementari e la memorizzazione di risultati intermedi; i blocchi romboidali rappresentano i punti di scelta in funzione di una condizione; le frecce indicano l'ordine delle operazioni, cioè definiscono il flusso di controllo. Nella figura successiva evidenziamo, sullo stesso esempio, i principali elementi della notazione.





La notazione introdotta, con molte varianti, viene usata comunemente per descrivere algoritmi, senza dover riferirsi ad uno specifico linguaggio di programmazione, in cui peraltro un algoritmo dovrà essere descritto qualora si voglia eseguirlo con un calcolatore.

Una delle varianti, particolarmente interessante, anche perché ripresa dal linguaggio di programmazione C, è quella che vede ad esempio l'inizializzazione "conta ← 0" rappresentata come "conta = 0" ed analogamente l'incremento "conta ← conta + 1" come "conta = conta + 1".

Questa notazione merita un approfondimento:

il nome "conta", come pure i nomi "somma" e "numero" indicano delle *variabili*. Ora, il concetto di variabile noto in matematica è quello di un simbolo che in realtà indica un qualsiasi valore in un particolare dominio. Quando scriviamo "poniamo $x = 0$ " indichiamo che d'ora in avanti il nome x , che prima indicava un qualsiasi valore, ora indica il valore 0. L'espressione " $x = x + 1$ " la leggiamo in genere come una proposizione sempre falsa, perché non esiste un numero uguale al suo successore.

Il concetto di variabile dell'informatica è fondamentalmente diverso: una variabile rappresenta un elemento di memorizzazione, ad es. un foglio di carta, in cui si può scrivere un valore (preso da un certo dominio che chiameremo in termini informatici *tipo* – è opportuno pensare fin da ora che ad una variabile è associato un certo tipo, cioè che vi posso scrivere solo valori di quel tipo). Ma il modo più opportuno di pensare ad una variabile è quello di pensare ad un contenitore che contiene un valore. Scritture successive in una variabile cancellano il valore precedentemente contenuto. Il contenitore non può mai essere vuoto: se non è mai stato scritto un valore in una variabile, questa contiene comunque un valore non noto, che diremo *indefinito*.

Quindi le scritte “ $\text{conta} \leftarrow 0$ ” o “ $\text{conta} = 0$ ”, a seconda della notazione utilizzata si devono quindi leggere come “nella variabile *conta* si scrive il valore 0” o, meglio, “alla variabile *conta* si assegna il valore 0”. Questa operazione prende il nome di *assegnamento*.

Le scritte “ $\text{conta} \leftarrow \text{conta} + 1$ ” o “ $\text{conta} = \text{conta} + 1$ ” si leggono come: “alla variabile *conta* si assegna il risultato della somma del valore 1 al valore precedente della variabile *conta*”, vale a dire che il valore contenuto nella variabile *conta* viene incrementato di uno.

Si noti che il significato della occorrenza del nome “*conta*” varia a seconda che questo sia a sinistra o a destra del simbolo di assegnamento:

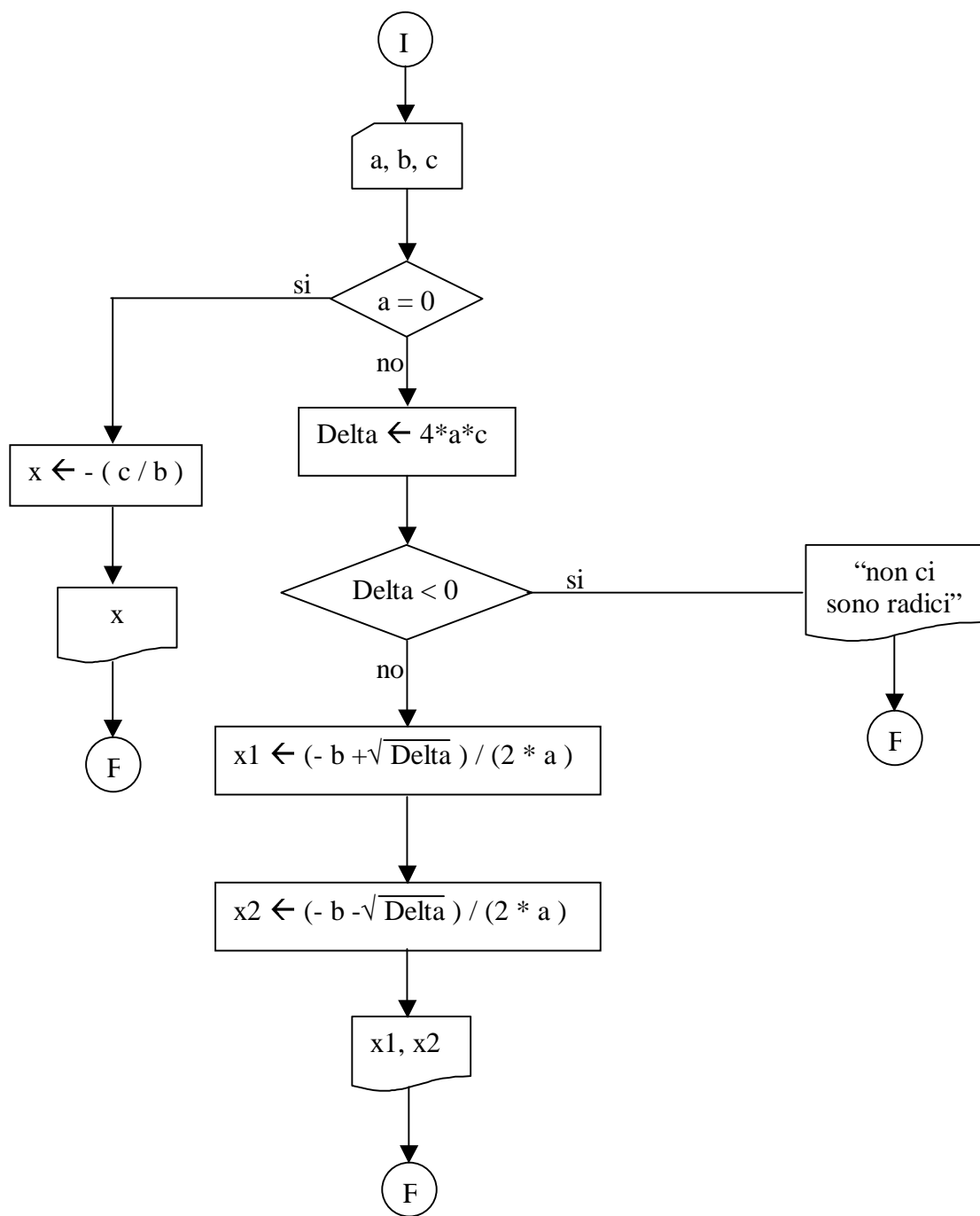
- a sinistra del simbolo di assegnamento vi si può trovare solo il *nome* di una variabile, che indica che l’assegnamento ha l’effetto di modificare il valore della variabile, scrivendoci un nuovo valore
- a destra del simbolo di assegnamento, vi si può trovare un’espressione, il cui valore viene scritto nella variabile indicata a sinistra. Un’eventuale occorrenza di un nome di variabile in tale espressione indica il *valore* contenuto nella variabile, prima che sia effettuato l’assegnamento.

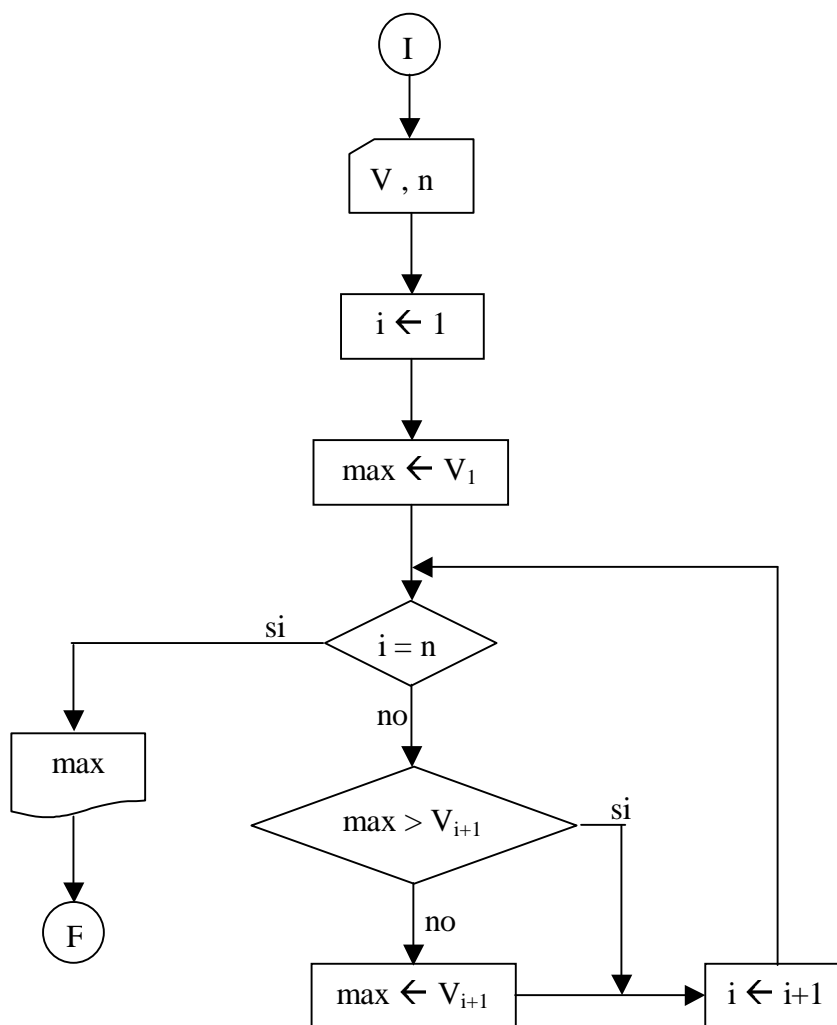
Diamo ora gli algoritmi per i problemi proposti, partendo da quello più familiare, cioè l’algoritmo di risoluzione di equazioni di secondo grado, e dando poi l’algoritmo che trova il massimo elemento di un insieme. Per quanto riguarda l’ordinamento, che necessita di un algoritmo più complicato, avremo modo di parlarne come uno degli ultimi argomenti di questo corso.

L’algoritmo per la risoluzione delle equazioni di secondo grado descritto nella seguente figura riflette il normale procedimento che seguiamo manualmente, e presenta alcune ramificazioni in dipendenza del valore del coefficiente a e del *delta*.

Nella figura successiva è descritto l’algoritmo per trovare il massimo elemento di un insieme, che agisce attraverso una *scansione* degli elementi dell’insieme, memorizzando via via in una variabile il maggior elemento finora trovato, ottenuto confrontando ogni elemento dell’insieme con il valore memorizzato fino a quel momento.

Si noti che per scandire l’insieme devo supporre che gli elementi dell’insieme siano memorizzati in una sequenza: di per se, la definizione matematica di un insieme non implica alcuna nozione di ordine tra gli elementi, ma per scandire gli elementi devo decidere un ordine in cui esaminarli. In questo algoritmo, si è deciso di rappresentare l’insieme come un vettore, i cui elementi sono quindi ordinati secondo l’ordine degli indici. Il modo con cui viene assegnata una posizione del vettore a un elemento dell’insieme è irrilevante, proprio perché stiamo trattando un insieme.





Si può notare che esiste una dipendenza degli algoritmi dalla rappresentazione dei dati e viceversa. Se avessimo adottato una diversa rappresentazione dell'insieme (ad esempio la rappresentazione mediante vettore caratteristico, di comune utilizzo, che qui non trattiamo) l'algoritmo sarebbe stato diverso.

Quindi una stessa funzione matematica può essere calcolata da diversi algoritmi, che possono variare per la rappresentazione dei dati, o semplicemente perché si possono ordinare diversamente alcune operazioni (calcolo del 2^a prima del Delta, o viceversa, nella risoluzione di equazioni di secondo grado).

E' possibile peraltro dimostrare l'esistenza di funzioni di cui è possibile dare una specifica formale, cioè una definizione matematica precisa, per cui non esiste un algoritmo che le calcola. La TEORIA

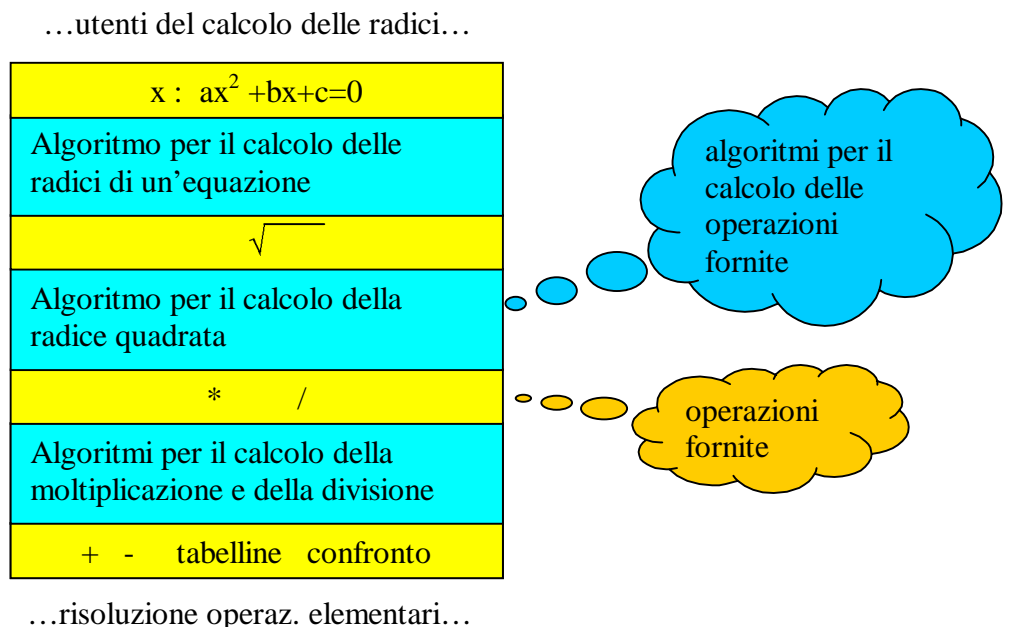
della CALCOLABILITÀ è la disciplina che tratta tali risultati di fondamentale importanza per l'informatica.

Possiamo notare che gli algoritmi proposti suppongono di utilizzare come operazioni elementari almeno le quattro operazioni aritmetiche, e l'algoritmo per la risoluzione di equazioni di secondo grado anche l'operazione di estrazione di radice quadrata: quest'ultima operazione non sempre viene però considerata elementare: in questo caso occorrerebbe definire un algoritmo che la calcola. D'altra parte, anche le operazioni elementari necessitano di procedimenti risolutivi, che usano operazioni ancora più elementari: abbiamo infatti imparato ad eseguire le moltiplicazioni con un algoritmo che esegue somme successive di prodotti parziali, e questi prodotti parziali li sappiamo fare grazie alle "tabelline" che abbiamo imparato alle elementari.

Possiamo quindi disegnare una struttura a livelli in cui collocare i vari algoritmi.

Ogni livello definisce un algoritmo che usufruisce di operazioni fornite dal livello sottostante (o dai livelli sottostanti) e fornisce una nuova operazione ai livelli sovrastanti.

Per usare operazioni di più basso livello un algoritmo non ha bisogno di conoscere l'algoritmo che le calcola, ma solo la loro specifica, cioè il *cosa* fanno, non il *come* lo fanno.



Al fine di calcolare automaticamente una operazione (funzione), occorrerà un esecutore di algoritmi (interprete, compilatore, processore) capace di eseguire algoritmi scritti in un dato linguaggio (linguaggio di programmazione) usando le operazioni elementari fornite dal linguaggio. Questo ragionamento si può applicare, RICORSIVAMENTE, a queste operazioni.

Cioè, data una operazione a livello i , questa viene eseguita tramite un algoritmo scritto in termini di operazioni più elementari, di livello $i-1$. Ognuna di queste viene eseguita tramite un algoritmo che usa operazioni a livello $i-2$, etc... Perché tutto funzioni (principio di induzione) occorre che esista un livello (livello 0) di operazioni elementari "atomiche" che siano eseguite automaticamente, senza ricorrere ad ulteriori operazioni di più basso livello.

Alla ricerca delle operazioni più elementari: L'algebra di Boole

Possiamo iniziare questa ricerca considerando quale sia l'informazione più elementare che possiamo considerare. Si tratta della cosiddetta informazione binaria, in cui i possibili valori informativi sono solo due:

- *assenza di informazione, presenza di informazione*

due valori che possiamo anche considerare in molte forme diverse:

- *falso, vero*
- *nero, bianco,*
- *0, 1*
- *diritto, rovescio*
- *si, no,*
- *caldo, freddo*
- *acceso, spento*

ecc...

In ognuna di queste forme l'informazione viene ridotta al minimo necessario per fornire un contenuto informativo significativo.

Questa informazione elementare viene detta **bit** e ha un immediato corrispettivo in elettronica in un circuito che possa essere sotto tensione (1) o in assenza di tensione (0).

Nel seguito utilizzeremo indifferentemente i simboli **T**, **true** (inglese per "vero") e **1** per un valore e **F**, **false** e **0** per l'altro.

Possiamo quindi considerare l'insieme $\{0,1\}$ come l'insieme di tutti i valori binari; questo insieme viene detto *dominio dei valori booleani*, e l'algebra che su di esso si costruisce con le operazioni che andremo a definire *algebra di Boole*. Le operazioni che lavorano su questo dominio saranno definibili come operazioni che, dati uno (operazioni unarie) o due valori booleani ritornano un valore booleano. Tali operazioni possono essere definite per enumerazione dei possibili valori in ingresso, attraverso delle tabelle dette *tabelle di verità*.

Nel caso di operazioni di un argomento, oltre alle funzioni banali costante 0, costante 1 e identità, troviamo una sola funzione significativa: la *negazione* o *complemento*, che indicheremo con **NOT**.

x	0	1	x	NOT x
0	0	1	0	1
1	0	1	1	0

Le funzioni booleane di un argomento sono riconducibili alle combinazioni di due valori booleani, sono quindi $2^2 = 4$.

Una tabella analoga può essere definita per le funzioni booleane di due argomenti:

x	y	0	1	x	y	NOT x	NOT y	x OR y	x AND y	x XOR y	x=>y	x<=y	x<=>y	x NAND y	x NOR y	NOT x AND y	x AND NOT y
0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	0	0
0	1	0	1	0	1	1	0	1	0	1	1	0	0	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0	1	0	1	0	0	1
1	1	0	1	1	1	0	0	1	1	0	1	1	1	0	0	0	0

Le funzioni booleane di due argomenti sono riconducibili alle combinazioni di quattro valori booleani, sono quindi $2^4 = 16$.

Per denotare le varie funzioni booleane, si usano anche le seguenti notazioni o denominazioni:

NOT a = $\sim a = \neg a = \bar{a}$ *negazione, complemento*
 x OR y = $x \vee y = x + y$ *disgiunzione*
 x AND y = $x \wedge y = x * y = xy$ *congiunzione*
 x => y *implicazione logica (x implica y)*
 x <= y *co-implicazione logica (y implica x)*
 x <=> y = x sse y = x iff y *equivalenza logica*
 x XOR y = $x \oplus y$ *or esclusivo*

Questa varietà di notazioni deriva dal fatto che l'algebra di Boole viene utilizzata come strumento base in molte discipline, tra cui primeggia la disciplina della LOGICA. Si può cioè dare un'interpretazione logica agli operatori booleani, chiamati anche *connettivi logici*, che traspare dall'utilizzo che abbiamo fatto di alcuni nomi dati agli operatori, e dall'uso dei valori vero/falso.

L'interpretazione logica dell'algebra di Boole prende il nome di *calcolo preposizionale*, perché permette di calcolare la verità di proposizione complesse, ottenute tramite connettivi logici, a partire dalla verità di proposizioni semplici.

Mentre l'interpretazione logica dell'AND è immediata (le due proposizioni connesse devono essere entrambe vere perché ne sia vera la congiunzione), per l'OR la cosa è meno immediata.

La verità della proposizione "la lampadina A è accesa O la lampadina B è accesa" può in realtà essere data in due modi in italiano:

1. la proposizione è vera quando una delle due lampadine è accesa, oppure entrambe sono accese, ed è falsa se entrambe sono spente (OR)
2. la proposizione è vera quando una delle due lampadine è accesa, ed è falsa se entrambe sono spente, ma anche se entrambe sono accese. (XOR)

Questo perché l'italiano ha perso la distinzione tra "vel" (disgiunzione) e "aut" (or esclusivo) presente in latino, lasciando al contesto di decidere del significato di una frase che usi "o" o "oppure".

Usando tabelle di verità come quelle viste in precedenza è possibile verificare l'equivalenza tra espressioni booleane ottenute come combinazioni di altre operazioni:

x	y	NOT x	NOT y	NOT x OR NOT y	x AND y	NOT (x AND y)	x NAND y
0	0	1	1	1	0	1	1
0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	1
1	1	0	0	0	1	0	0

Le colonne ombreggiate risultano uguali, e quindi esprimono l'equivalenza:

$$x \text{ NAND } y = \text{NOT } x \text{ OR NOT } y = \text{NOT } (x \text{ AND } y)$$

Analogamente, possono essere dimostrate (e la dimostrazione viene lasciata al lettore) le seguenti equivalenze:

$$\text{NOT NOT } x = x$$

$$x \text{ NOR } y = \text{NOT } x \text{ AND NOT } y = \text{NOT } (x \text{ OR } y)$$

$$x \Rightarrow y = \text{NOT } x \text{ OR } y$$

$$x \Leftarrow y = \text{NOT } y \text{ OR } x$$

$$x \Leftrightarrow y = \text{NOT } (x \text{ XOR } y)$$

$$x \text{ XOR } y = (x \text{ AND NOT } y) \text{ OR } (\text{NOT } x \text{ AND } y)$$

$$x \text{ OR } y = y \text{ OR } x \quad (\text{commutatività})$$

$$x \text{ AND } y = y \text{ AND } x \quad (\text{commutatività})$$

$$x \text{ XOR } y = y \text{ XOR } x \quad (\text{commutatività})$$

$$x \text{ OR } x = x \quad (\text{idempotenza})$$

$$x \text{ AND } x = x \quad (\text{idempotenza})$$

$$x \text{ XOR } x = 0$$

$$x \text{ OR NOT } x = 1$$

$$x \text{ AND NOT } x = 0$$

$$x \text{ XOR NOT } x = 1$$

$$x \text{ OR } 0 = x \quad (\text{scritto come } x + 0 = x, \text{ è familiare...})$$

$$x \text{ AND } 0 = 0 \quad (\text{scritto come } x * 0 = 0, \text{ è familiare...})$$

$$x \text{ XOR } 0 = x$$

$$x \text{ OR } 1 = 1 \quad (\text{scritto come } x + 1 = 1, \text{ non è affatto familiare!})$$

$$x \text{ AND } 1 = x \quad (\text{scritto come } x * 1 = x, \text{ è familiare...})$$

$$x \text{ XOR } 1 = \text{NOT } x$$

$$\text{NOT } (x \text{ OR } y) = \text{NOT } x \text{ AND NOT } y$$

$$\text{NOT } (x \text{ AND } y) = \text{NOT } x \text{ OR NOT } y$$

Le due ultime equivalenze vengono dette leggi di De Morgan e mostrano come l'operatore AND possa essere realizzato attraverso una combinazione di OR e NOT e, viceversa, come l'operatore OR possa essere realizzato attraverso una combinazione di AND e NOT.

Altra cosa che possiamo notare è che vale una regola di *dualità*: data una equivalenza tra espressioni contenenti NOT, AND e OR, vale anche una equivalenza analoga, ottenuta dalla prima sostituendo gli AND con OR e viceversa, e gli 1 con 0 e viceversa. Le leggi di De Morgan son un esempio di equivalenze duali.

Infine, poiché abbiamo definito tutti gli operatori introdotti in termini di AND, OR e NOT, grazie alle leggi di De Morgan possiamo dire che tutte le funzioni booleane in 2 argomenti sono esprimibili con combinazioni degli operatori AND e NOT, con OR e NOT.

Se inoltre consideriamo le seguenti ulteriori equivalenze:

$$\text{NOT } x = x \text{ NAND } x$$

$$x \text{ AND } y = (x \text{ NAND } y) \text{ NAND } (x \text{ NAND } y)$$

$$x \text{ OR } y = (x \text{ NAND } x) \text{ NAND } (y \text{ NAND } y)$$

abbiamo che gli operatori AND, OR e NOT possono essere espressi tramite combinazione di soli operatori NAND, e quindi anche tutte le funzioni booleane in 2 argomenti sono esprimibili con combinazioni di soli operatori NAND.

Per dualità, la stessa cosa vale per l'operatore NOR, quindi tutte le funzioni booleane in 2 argomenti sono esprimibili con combinazioni di soli operatori NOR.

Analogamente a quanto fatto per le funzioni su due argomenti, è possibile costruire una tabella di verità per definire tutte le funzioni di tre argomenti. Si tratterà di una tabella di $2^3 = 8$ righe (per contenere tutte le combinazioni dei valori dei tre argomenti), e $2^8 = 256$ colonne (per contenere tutte le combinazioni dei valori booleani sulle 8 righe). Esistono pertanto 256 funzioni booleane su tre argomenti.

Riguardo alle funzioni booleane su tre argomenti, è quindi possibile dimostrare le seguenti equivalenze (proprietà distributive):

$$x \text{ OR } (y \text{ AND } z) = (x \text{ OR } y) \text{ AND } (x \text{ OR } z)$$

$$x \text{ AND } (y \text{ OR } z) = (x \text{ AND } y) \text{ OR } (x \text{ AND } z)$$

mediante la seguente tabella di verità:

x	y	z	x OR y	x OR z	y OR z	x AND y	x AND z	y AND z	x OR (y AND z)	(x OR y) AND (x OR z)	x AND (y OR z)	(x AND y) OR (x AND z)
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	1	1	1	0	0
1	0	0	1	1	0	0	0	0	1	1	0	0
1	0	1	1	1	1	0	1	0	1	1	1	1
1	1	0	1	1	1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1

Se scriviamo le due equivalenze precedenti usando la notazione per i connettivi logici che ricorda la notazione aritmetica, otteniamo:

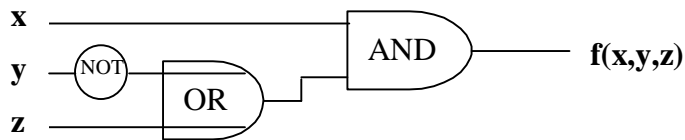
$$x + yz = (x + y)(x + z)$$

$$x(y + z) = xy + xz$$

Mentre riconosciamo la seconda come la regola di distribuzione della somma rispetto al prodotto, tipica dell'aritmetica, la prima equivalenza suona meno familiare, non avendo un corrispettivo in aritmetica. In aritmetica infatti non vige la dualità tra gli operatori di somma e prodotto. Nell'algebra di Boole, valgono invece sia la distributività dell'OR rispetto all'AND, sia quella dell'AND rispetto all'OR.

E' peraltro possibile dimostrare, sempre con lo stesso metodo, che tutte le funzioni booleane su due argomenti si possono realizzare mediante composizione delle funzioni booleane su due argomenti. Questa proprietà si estende a tutte le funzioni booleane su n argomenti, con n qualsiasi. Questo significa che è possibile costruire una qualsiasi funzione booleana di n argomenti mediante le sole funzioni basiche AND e NOT, oppure le sole OR e NOT, oppure solo NAND, oppure solo NOR.

E' possibile in effetti costruire mediante pochi semplici componenti elettronici, e a costo bassissimo, dei circuiti (dette *porte* o *gates*) che realizzano le funzioni basiche AND, OR, NOT, XOR, una volta che si siano assegnati a due distinti livelli di tensione (ad es. 0 volt e 5 volt) i valori booleani 0 e 1. La composizione di funzioni si ottiene semplicemente connettendo le uscite delle une con gli ingressi delle altre. Questo viene mostrato in figura, dove la funzione booleana in tre argomenti $f(x,y,z) = x \text{ AND } (y \Rightarrow z)$ viene realizzata mediante una porta AND, una OR e una NOT, rappresentate con una delle notazioni usuali per le porte logiche.



Un circuito come quello mostrato in figura viene chiamato *rete combinatoria* . E' quindi possibile realizzare un circuito elettronico che calcoli una qualsiasi funzione booleana, con i soli limiti alla sua complessità dati dal costo, dalle dimensioni fisiche e dalla complessità della rete risultante. Comunque la moderna tecnologia dei circuiti integrati permette di costruire *chip* di silicio dove vengono integrati qualche centinaia di migliaia di porte in un solo centimetro quadro, con un costo a chip di qualche euro e con un consumo di energia elettrica di qualche decina di milliwatt.