▼ **2.0 - Introduzione alla logica**

Introduzione

Nel corso di logica per l'informatica verranno studiate le **dimostrazioni**, ovvero sequenze di frasi che convincono il lettore che un ragionamento è valido. Individueremo linguaggi artificiali per scrivere i passi di una dimostrazione e un computer potrà dire se sono corretti.

La parola **valere** ha diversi significati, in base alla logica a cui si fa riferimento, ad esempio può indicare verità, programmabilità, conoscenza, possesso ecc.

La logica ha molti elementi in comune con la matematica e l'informatica, ma anche altrettanti elementi per cui differisce:

Matematica	Informatica	Logica
Risolve problemi tramite **dimostrazioni**.	Risolve problemi tramite **codice**.	Garantisce la **correttezza** di un ragionamento.
Si interessa all'**esistenza** delle soluzioni di un problema.	Si interessa al modo di **calcolare** le soluzioni di un problema.	

Un **codice è corretto** solo se c'è una dimostrazione che dice che quello che fa è quello che deve fare. I bug logici di un programma sono causati da errori logici che sarebbero evitati esplicitandone la dimostrazione. Per scrivere codice il più delle volte corretto occorre imparare a ragionare logicamente formulando nella propria mente le prove della correttezza.

La dimostrazione di codice viene però utilizzata solo per il **codice critico**, ovvero un codice i quali errori possono causare problemi importanti (es. pilota automatico di un aereo), in quanto è molto dispendiosa sia in termini economici che di tempistiche.

Paradossi

Differenza tra paradossi e antinomie

Un **paradosso** consiste in una conclusione contraria all'intuizione che deriva da premesse accettabili per mezzo di un ragionamento accettato.

Un **antinomia** consiste in una conclusione inaccettabile che deriva da premesse accettabili per mezzo di un ragionamento accettato.

Nel corso si parlerà di paradossi intendendo antinomie al fine di semplificare.

Linguaggio naturale

Il **linguaggio naturale** è il linguaggio alla base delle comunicazioni tra gli esseri umani. Esso viene spesso utilizzato per descrivere procedure di calcolo e ragionamenti logici, ma ciò non è corretto in quanto esso contiene molti paradossi, è ambiguo e dipende fortemente dal contesto.

Paradossi in matematica

Nel linguaggio matematico è semplice introdurre dei nuovi paradossi. Vediamo un esempio di paradosso matematico nel **paradosso di Russell**.

Prendiamo l'insieme di tutti gli insiemi che non contengono sè stessi $X = \{Y | Y \not\in Y\}$ e chiediamoci se X contiene sè stesso:

- Se sì, X contiene un insieme che contiene sè stesso, dunque la premessa viene violata.
- Se no, X non contiene un insieme che non contiene sè stesso, dunque non contiene nessuno dei suoi sottoinsiemi, i quali non contengono sè stessi.

L’insieme X formato da tutti gli insiemi che non contengono sè stessi è dunque un **paradosso**.

Per evitarlo occorre stabilire di non poter formare un insieme partendo da una qualsiasi proprietà, ma è possibile solo selezionare elementi a partire da un insieme già esistente. Ad esempio non è possibile formare l’insieme $\{Y | Y$ è uno studente di informatica$\}$, ma è possibile formare l’insieme $\{Y$ appartiene all’insieme degli studenti $| Y$ studia informatica$\}$.

Inoltre occorre stabilire che la collezione di tutti gli insiemi non è un insieme ma una **classe propria**, dunque non è possibile l’uso metalinguistico della nozione di insieme, ovvero un insieme che parla di sè stesso.

Paradossi in informatica

Molti dei paradossi in informatica sono dati dall’**uso metalinguistico delle funzioni**. Nei linguaggi higher order infatti una funzione può prendere in input/dare in output altre funzioni, mentre nei linguaggi imperativi o a oggetti una funzione può prendere in input/dare in output delle reference ad altre funzioni.

Esempio:

- Siano f e g due funzioni, e sia $f(g) = not(g(g))$, allora $f(f) = not(f(f))$, il che è assurdo.

  Dunque ci sono due possibilità; o f non è scrivibile nel linguaggio di programmazione, il quale risulterebbe molto inespressivo, oppure f non è totale, in quanto $f(f)$ diverge, ovvero non fornisce output in tempo finito.

Chiediamoci inoltre se esista un programma che sia in grado di stabilire se un altro diverga o meno ($\downarrow$: converge, $\uparrow$: diverge):

$f(g, x) =$ true iff $g(x) \downarrow$ è il programma che stabilisce se un altro diverga o meno (g è il programma e x è il suo parametro).

$h(g) =$ if $f(g, g)$ then $\uparrow$ else $\downarrow$ è il programma che abbiamo creato per dimostrare che non per tutti i programmi è possibile stabilire se divergano o meno.

$\implies h(h) \uparrow$ iff $f(h, h) =$ true iff $h(h) \downarrow$, il che è assurdo.

Chiediamoci infine se ogni linguaggio di programmazione può esprimere tutte le funzioni matematiche. Consideriamo un linguaggio di programmazione non tipato e T come l’insieme di tutti i valori possibili nel linguaggio (bool, funzioni ecc.).

$T = \{0, 1\} \cup T^T$ (T^T = tutte le funzioni matematiche che vanno dal dominio T all’immagine T)

T dunque contiene almeno i valori booleani 0 e 1 e tutte le funzioni matematiche che vanno da T a T. Per il teorema della diagonalizzazione di Cantor $|T| < 2 + |T^T|$ ($|| :$ cardinalità), il che è assurdo, dunque non tutti i linguaggi di programmazione possono esprimere tutte le funzioni matematiche.