Fondamenti

Chapter 1 - Data storage

1.1 - Bit and their storage

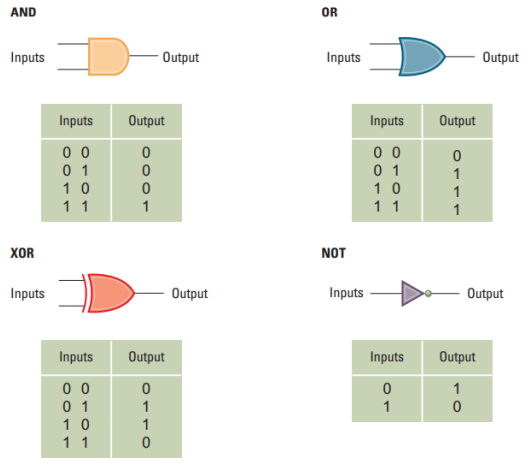
Fondamenti di memorizzazione dei dati:

A livello più basso, tutte le informazioni di un computer sono memorizzate come schemi di 0 (falso) e 1 (vero) (bit), che possono rappresentare vari tipi di dati, tra cui numeri, caratteri, colori e suoni.

Operazioni booleane:

Le operazioni di base (AND, OR, XOR, NOT) manipolano valori vero/falso (rappresentati da 1 e 0).

Ogni operazione produce output specifici in base a diverse combinazioni di input.



Inputs	Output
0 0	0
0 1	0
1 0	0
1 1	1

Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	1

Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	0

Inputs	Output
0	1
1	0

Porte logiche e circuiti:

Le porte logiche sono dispositivi che implementano operazioni booleane, generalmente costruite utilizzando transistor nei computer moderni.

I flip-flop sono unità fondamentali di memoria che possono memorizzare un singolo bit, mantenendo il suo valore (0 o 1) fino a quando non viene modificato da un input esterno.

Implementazione tecnologica:

L'integrazione su larga scala (VLSI - Very Large Scale Integration) consente di costruire milioni di componenti su un singolo chip.

La notazione esadecimale è utilizzata come metodo più gestibile per rappresentare lunghe sequenze di bit, usando un singolo simbolo per ogni quattro bit.

Bit pattern	Hexadecimal representation
0000	0x0
0001	0x1
0010	0x2
0011	0x3
0100	0x4
0101	0x5
0110	0x6
0111	0x7
1000	0x8
1001	0x9
1010	0xA
1011	0xB
1100	0xC
1101	0xD
1110	0xE
1111	0xF

1.2 - Main memory

Organizzazione della memoria

La memoria principale di un computer è composta da circuiti organizzati in unità chiamate **celle**, tipicamente di 8 bit (un byte). Ogni cella ha un indirizzo unico, analogo al sistema di indirizzi di una città, per identificarle in modo univoco. Le celle possono essere accessibili indipendentemente in ordine casuale, da cui il nome **RAM** (Random Access Memory).

Bit all'interno di una cella

I bit di una cella sono disposti in fila:

- L'estremità sinistra è il **bit più significativo** (high-order), che rappresenta il valore numerico maggiore.
- L'estremità destra è il **bit meno significativo** (low-order).

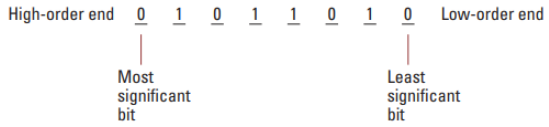


Figure 1.7 The organization of a byte-size memory cell

Estensione della memoria

La memoria può contenere bit pattern più lunghi di una singola cella utilizzando celle consecutive.

- **Lettura**: Recupero dei dati memorizzati in una cella.
- **Scrittura**: Memorizzazione di un pattern di bit in una cella specifica.

Tecnologie di memoria

La RAM moderna utilizza tecnologie avanzate come **DRAM** (Dynamic RAM), che memorizza i bit come cariche elettriche volatili e richiede circuiti di **refresh** per mantenere i dati. Versioni migliorate come **SDRAM** offrono tempi di accesso ridotti.

Capacità della memoria

Le dimensioni della memoria sono spesso potenze di 2 e misurate in unità come:

- **Kilobyte (KB)**: 1024 byte
- **Megabyte (MB)**: 1024 KB
- **Gigabyte (GB)**: 1024 MB
- **Terabyte (TB)**: 1024 GB

Tuttavia, queste unità sono talvolta confuse con i multipli di 10 (1000) usati in altri campi. Per chiarezza, termini come **kibi-**, **mebi-**, ecc. vengono usati per riferirsi a potenze di 1024.

Nota importante: L'ambiguità rimane diffusa, quindi è necessario fare attenzione al contesto.

1.3 - Mass storage

I sistemi di **memoria di massa** (o memoria secondaria) sono dispositivi utilizzati per superare i limiti di volatilità e dimensione della memoria principale di un computer. Offrono maggiore capacità di archiviazione, minori costi e possibilità di rimuovere i supporti per scopi archivistici. Tuttavia, i sistemi con componenti meccanici, come dischi magnetici e ottici, sono più lenti e meno affidabili rispetto ai sistemi elettronici come le unità a stato solido (SSD).

Tecnologie principali:

1. **Sistemi Magnetici**
    - **Dischi rigidi (HDD)**:Utilizzano dischi magnetici rotanti con testine di lettura/scrittura. I dischi sono organizzati in **tracce**, suddivise in **settori**.
        - **Prestazioni**:
            - Tempo di accesso: somma di tempo di ricerca (seek time) e ritardo rotazionale.
            - Velocità di trasferimento dipende dalla posizione sulla traccia (più veloce nelle zone esterne).
        - Problemi: movimenti meccanici lenti e rischio di **head crash** (urto delle testine).
    - **Tecniche avanzate**:
        - **Zoned-Bit Recording**: tracce esterne contengono più settori per un uso più efficiente dello spazio.
2. **Sistemi Ottici**
    - **CD, DVD e Blu-ray**:Memorizzano dati con variazioni nella superficie riflettente, letti da un laser.
        - I dati sono registrati su una traccia a spirale unica, con settori uniformi.

    - Capacità:
        - CD: 600-700MB.
        - DVD: fino a diversi GB.
        - Blu-ray: oltre 100GB con i modelli multilayer.
    - Ideali per grandi volumi di dati lineari (es. musica o video).
3. **Memorie Flash**
    - **Flash Drive e SSD (Solid-State Drives)**:
        - Archiviazione elettronica non volatile senza parti in movimento.
        - Vantaggi:
            - Velocità maggiore rispetto ai sistemi magnetici/ottici.
            - Resistenza agli urti.
        - Svantaggi:
            - Limitata durata a causa dell'usura delle celle.
            - Tecniche come il "wear-leveling" distribuiscono l'usura.
    - **SD Cards**:Piccole schede di memoria utilizzate in dispositivi portatili.
        - Capacità:
            - SD: fino a 2GB.
            - SDHC: fino a 32GB.
            - SDXC: oltre 1TB.

Conclusioni:

Le tecnologie di memoria di massa differiscono per velocità, capacità, affidabilità e costi.

- **SSD** e **flash drive**: preferiti per applicazioni portatili e prestazioni elevate.
- **HDD**: scelta economica per grandi volumi di dati.
- **Supporti ottici**: ideali per l'archiviazione a lungo termine.

1.4 - Representing information as bit patterns

Testo

- **Codifica ASCII**: Utilizza sequenze di 7 bit per rappresentare lettere, numeri e simboli. Viene esteso a 8 bit per facilitare l'archiviazione in memoria e aggiungere altri simboli.
- **Limiti di ASCII**: Non sufficiente per lingue asiatiche o documenti multilingue.
- **Unicode**: Standard moderno che utilizza fino a 21 bit per simbolo, con varianti come UTF-8 per compatibilità con ASCII e rappresentazione di caratteri internazionali.

Valori numerici

- **Notazione binaria**: Utilizza solo 0 e 1, permettendo una rappresentazione più efficiente rispetto alla codifica in caratteri (es. 16 bit per numeri fino a 65535).
- **Varianti binarie**:
  - *Complemento a due*: Per numeri interi positivi e negativi.
  - *Floating-point*: Per numeri con frazioni.

Immagini

- **Bitmap**: Rappresenta immagini come insiemi di pixel.
  - *Semplice*: 1 bit per pixel (bianco e nero).
  - *Più complesso*: 8 bit per sfumature di grigio o 24 bit per colore (RGB).
- **Problemi di scala**: L'ingrandimento porta a immagini "sgranate".
- **Geometria analitica**: Utilizzata per immagini scalabili e caratteri (es. TrueType, PostScript).

Suono

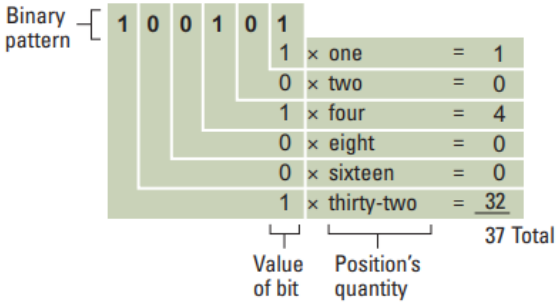
- **Campionamento**: Registrazione di ampiezze a intervalli regolari (es. 8000 campioni al secondo per voce, 44100 per audio ad alta fedeltà).

- **Dati richiesti**: 16 bit per campione (32 per stereo), con oltre 1 milione di bit al secondo per musica stereo.

1.5 - The binary sistem

Notazione binaria

- **Sistema binario**: Utilizza solo i numeri 0 e 1, a differenza del sistema decimale che usa 0-9.
- **Posizioni**: Ogni posizione in un numero binario è associata a potenze di 2. La posizione più a destra corrisponde a $2^0 = 1$, la successiva a$2^1 = 2$, poi $2^2 = 4$, e così via.
- **Esempio**: In binario, 1011 rappresenta 1×8 + 0×4 + 1×2 + 1×1 = 11 in decimale.



Algoritmo per ottenere la rappresentazione binaria

1. Dividere il numero per 2 e registrare il resto.
2. Continuare a dividere il quoziente per 2 fino a ottenere un quoziente pari a zero.
3. La rappresentazione binaria è data dai resti letti in ordine inverso.

- **Esempio**: Il numero 13 diviso per 2 dà i resti 1, 0, 1, 1, che corrispondono a 1101 in binario.

Aggiunta binaria

- Simile all'addizione decimale: sommare i bit da destra a sinistra, scrivere il bit meno significativo sotto la colonna e "trasportare" il bit più significativo alla

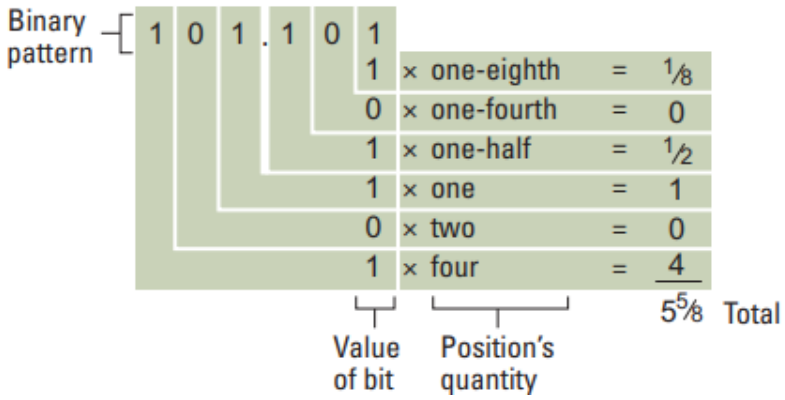
colonna successiva.

- **Esempio**: 111010+11011=1010101 in binario.

```
  111010
+  11011
1010101
```

Numeri frazionari in binario

- Il punto binario (floating point) separa la parte intera dalla parte frazionaria. Le posizioni dopo il punto rappresentano frazioni di potenze di 2: $2^{-1}, 2^{-2}, 2^{-3}...$
- **Esempio**: 101.101 rappresenta 5+4/8=5.625.



Somma di numeri frazionari binari

- Per sommare numeri con il punto binario, si allineano i punti e si segue lo stesso processo di addizione dei numeri interi.
- **Esempio**: 10.011+100.11=111.001 in binario.

1.6 - Storing integers

Notazione in Complemento a Due:

- **Rappresentazione**: La notazione in complemento a due rappresenta sia i numeri positivi che quelli negativi utilizzando un numero fisso di bit. Ad esempio, un sistema a 4 bit può rappresentare valori da -8 a 7.

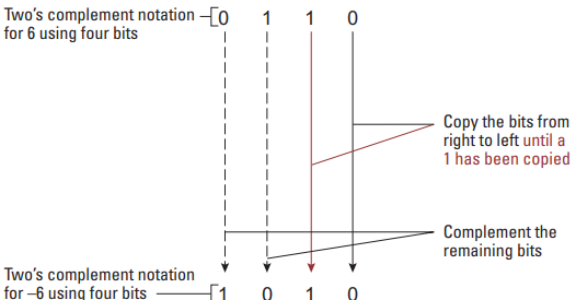
a. Using patterns of length three

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	–1
110	–2
101	–3
100	–4

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	–1
1110	–2
1101	–3
1100	–4
1011	–5
1010	–6
1001	–7
1000	–8

- **Bit di Segno**: Il bit più a sinistra (bit di segno) indica il segno del numero. Un `0` rappresenta un numero positivo, mentre un `1` rappresenta un numero negativo.
- **Conversione**:
  - I numeri positivi sono rappresentati come numeri binari usuali.
  - I numeri negativi sono rappresentati invertendo la rappresentazione binaria del numero positivo e aggiungendo 1 (questo è il metodo **copia e complementa**).



- **Addizione**: La notazione in complemento a due permette di sommare numeri positivi e negativi utilizzando lo stesso algoritmo. La sottrazione può essere eseguita sommando il valore negativo di un numero.
- **Overflow**: Esiste un limite alla gamma di valori che possono essere rappresentati. L'overflow si verifica quando il risultato supera l'intervallo

rappresentabile di valori (ad esempio, 5 + 4 in un sistema a 4 bit produce un risultato errato a causa dell'overflow).

Notazione Eccesso:

- **Rappresentazione**: La notazione eccesso utilizza anche pattern di bit a lunghezza fissa, ma sposta la rappresentazione dei numeri di un valore fisso, noto come "valore eccesso".
- **Notazione Eccesso Otto**: In un sistema a 4 bit, il valore 0 è rappresentato dal pattern binario che corrisponde a 8 nel binario tradizionale. Ad esempio, nella notazione eccesso otto:
  - 1100 in binario rappresenta 12, ma è interpretato come 4 in eccesso otto.
  - 0000 in binario rappresenta 0, ma è interpretato come -8.

Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

Figure 1.22 An excess eight conversion table

- **Generalizzazione**: Il sistema è chiamato **notazione eccesso N**, dove N è il valore di offset utilizzato per rappresentare zero.

1.7 - Storing fractions

La rappresentazione dei numeri frazionari nei computer avviene tramite la **notazione in virgola mobile**, che è basata sulla notazione scientifica. Poiché lo spazio di archiviazione nei computer è limitato, la precisione con cui possiamo

rappresentare frazioni è anch'essa limitata, e questo può portare a **errori di troncamento**.

Notazione in Virgola Mobile (Floating-Point):

In un esempio semplificato con un byte (8 bit), la notazione in virgola mobile usa:

- **Bit di segno** (1 bit): 0 per valori positivi, 1 per valori negativi.
- **Campo esponente** (3 bit): rappresenta l'esponente in notazione eccesso.
- **Campo mantissa** (4 bit): rappresenta la parte frazionaria del numero.

Per esempio, il pattern di bit `01101011` rappresenta il numero `2¾`. La parte frazionaria viene estratta dal campo mantissa e la posizione della virgola (floating point) viene determinata dal campo esponente.

Formato a Precisione Singola:

I moderni computer usano formati più complessi come il **floating-point a precisione singola** (32 bit), che usa 1 bit per il segno, 8 bit per l'esponente e 23 bit per la mantissa, permettendo di rappresentare numeri molto grandi o molto piccoli con una precisione di 7 cifre decimali.

Errori di Troncamento:

Un problema comune è il **troncamento** quando la mantissa non ha abbastanza bit per rappresentare un numero con precisione. Ad esempio, rappresentare il numero `25/8` con un byte potrebbe comportare la perdita di parte del numero, portando a errori di arrotondamento.

Altri Problemi:

- **Espansioni non terminanti**: Alcuni numeri non possono essere rappresentati con precisione in binario (come `1/3` in decimale), causando errori anche con numeri che sembrano semplici.
- **Errori dovuti all'ordine delle operazioni**: In alcuni casi, l'ordine in cui si sommano i numeri può influire sul risultato a causa di errori di troncamento.

1.9 - Data compression

La compressione dei dati è una tecnica per ridurre la dimensione dei dati da archiviare o trasferire, mantenendo l'informazione. Si distingue tra compressione **lossless** (senza perdita di dati) e **lossy** (con perdita di dati).

Tecniche di compressione generiche

1. **Run-Length Encoding (RLE)**: Tecnica lossless che comprime sequenze di elementi identici rappresentandole con il valore e il numero di ripetizioni.
2. **Codifica dipendente dalla frequenza (Huffman)**: Codifica lossless che usa schemi di lunghezza variabile, più brevi per i dati più frequenti e più lunghi per quelli rari.
3. **Codifica relativa o differenziale**: Registra le differenze tra unità di dati consecutive. Può essere lossless o lossy.
4. **Codifica basata su dizionario**:
    - **Fisso**: Usa un dizionario predefinito per rappresentare parole o unità.
    - **Adattivo (LZW)**: Aggiorna il dizionario durante il processo di codifica, aggiungendo pattern ricorrenti per migliorare l'efficienza.

Compressione delle immagini

- **GIF**: Usa codifica basata su dizionario e riduce i colori a 256. È una tecnica lossy se i colori originali non sono rappresentati esattamente.
- **JPEG**: Tecnica principalmente lossy. Riduce i dati considerando le limitazioni dell'occhio umano, come la minore sensibilità ai cambiamenti di colore rispetto alla luminosità. Utilizza trasformazioni matematiche e codifiche successive per ottenere un alto rapporto di compressione (fino a 30 volte).
- **TIFF**: Usato più per lo storage standardizzato che per la compressione. È lossless e conserva informazioni accessorie come data e impostazioni della fotocamera.

Compressione audio e video

- **MPEG**: Standard per video. Comprende I-frame (immagini complete) e P/B-frame (immagini codificate con differenze relative). Gli I-frame usano tecniche simili al JPEG.

- **MP3**: Standard per l’audio che sfrutta le proprietà dell’orecchio umano, come il "mascheramento temporale" e "mascheramento di frequenza", per rimuovere dettagli non percepibili. Questo riduce significativamente la dimensione mantenendo un’alta qualità.

1.10 - Communication errors

Durante il trasferimento o l'archiviazione dei dati, possono verificarsi errori che alterano i bit originali. Le cause includono malfunzionamenti hardware, polvere, radiazioni o disturbi elettromagnetici. Per affrontare questi problemi, si usano tecniche di **rilevamento** e **correzione degli errori**, spesso integrate nei dispositivi moderni.

Rilevamento degli errori

- **Bit di parità:**
  - Si aggiunge un bit (parity bit) per garantire che ogni pattern contenga un numero specifico di 1 (pari o dispari).
  - Errori si rilevano quando il numero di 1 non rispetta la regola.
  - **Limitazione:** non rileva errori multipli pari (es. 2 errori nello stesso pattern).
- **Checkbyte e checksum:**
  - In lunghe sequenze di bit, si usano più bit di parità (checkbyte) per coprire varie porzioni dei dati, aumentando la probabilità di rilevare errori concentrati.
- **Cyclic Redundancy Checks (CRC):**
  - Una variante avanzata per rilevare errori su dati trasmessi o archiviati.

Correzione degli errori

- **Codici di correzione:**
  - Basati sulla distanza di Hamming, che misura il numero di bit diversi tra due pattern.

- Un codice con distanza di Hamming minima di 3 può:
  - **Rilevare fino a 2 errori** per pattern.
  - **Correggere 1 errore**.
- Codici con distanze maggiori consentono di rilevare e correggere errori più complessi.

Applicazioni pratiche

- **Memorie principali:**
  - Usano bit di parità per garantire l'affidabilità dei dati.
- **Dischi magnetici e CD:**
  - Tecniche di correzione minimizzano gli errori dovuti a difetti fisici.
  - Nei CD per dati, sistemi avanzati riducono gli errori a uno su 20.000 dischi, rispetto a quelli per audio (1 su 2).

Chapter 1 - Sets and Logic

1.1 - Sets

Il concetto di insieme è fondamentale in tutta la matematica e nelle sue applicazioni. Un insieme è semplicemente una **collezione di oggetti**, detti **elementi** o **membri**. Se un insieme è finito e non troppo grande, possiamo descriverlo elencando i suoi elementi. Per esempio, l'equazione:

$A=\{1,2,3,4\}$ descrive un insieme A composto dai quattro elementi 1, 2, 3 e 4. L'ordine degli elementi in un insieme non è importante; quindi, possiamo altrettanto bene scrivere $A=\{1,3,4,2\}$. Inoltre, gli elementi di un insieme sono sempre **distinti**: anche se per errore vengono elencati duplicati, essi vengono considerati una sola volta. Ad esempio, A può essere descritto come $A=\{1,2,2,3,4\}$.

Se un insieme è molto grande (finito o infinito), possiamo descriverlo usando una proprietà caratteristica per i suoi membri. Per esempio:

B={x ∣ x e` un numero intero positivo e pari} descrive l'insieme B composto da tutti gli interi positivi pari (cioè 2, 4, 6, ecc.). Il simbolo " ∣ " si legge come "tale che". Questa notazione si chiama **notazione a costruzione di insieme** (o **set-builder notation**).

Un insieme può contenere qualsiasi tipo di elementi, anche di tipi diversi. Per esempio,

{4.5,Lady Gaga,π,14} è un insieme perfettamente valido composto da quattro elementi: il numero 4.5, la persona Lady Gaga, il numero π (=3.1415...) e il numero 14.

Un insieme può anche contenere **altri insiemi** come elementi. Per esempio, l'insieme

{3,{5,1},12,{π,4.5,40,16},Henry Cavill} contiene cinque elementi: il numero 3, l'insieme {5,1}, il numero 12, l'insieme {π,4.5,40,16} e la persona Henry Cavill.

Esempi di insiemi numerici

Alcuni insiemi numerici ricorrenti in matematica sono:

- Z **:** Insieme degli interi (−3,0,2,145, ecc.).
- Q : Insieme dei numeri razionali (−1/3,0,24/15, ecc.).
- R : Insieme dei numeri reali (−3,−1.766,2,π, ecc.).

I numeri reali possono essere rappresentati come **punti su una retta numerica**, che si estende all'infinito in entrambe le direzioni.

Cardinalità degli insiemi

Se un insieme X è **finito**, indichiamo con $\left| X \right|$ il numero di elementi di X, detto **cardinalità**.

Esempio: Se A={1,2,3,4}, allora ∣A∣=4.

Per gli insiemi **infiniti**, la cardinalità può essere descritta con simboli speciali. Ad esempio, la cardinalità dell'insieme degli interi Z si indica con $\aleph 0$ ("aleph nullo").

Operazioni tra insiemi

1. **Unione (U)**: L'unione di due insiemi X e Y è l'insieme degli elementi che appartengono a X, a Y, o a entrambi:

   $X \cup Y = \{x \mid x \in X \text{ o } x \in Y\}$.

2. **Intersezione ($\cap$)**: L'intersezione di X e Y è l'insieme degli elementi che appartengono sia a X che a Y:

   $X \cap Y = \{x \mid x \in X \text{ e } x \in Y\}$.

3. **Differenza (−−)**: La differenza X- X è l'insieme degli elementi che appartengono a X, ma non a Y:

   $X - Y = \{x \mid x \in X \text{ e } x \notin Y\}$.

1.2 - Propositions

Definizione di proposizione

Una proposizione è una frase dichiarativa che può essere vera o falsa, ma non entrambe. Ad esempio:

- **Proposizioni valide:**
  - (a) "Gli unici numeri interi positivi che dividono 7 sono 1 e 7" (vera).
  - (c) "Per ogni numero intero positivo n, esiste un numero primo maggiore di n" (vera).
- **Non proposizioni:**
  - (e) "Compra due biglietti per il concerto" (un comando, quindi non può essere vera o falsa).
  - (f) "x + 4 = 6" (dipende dal valore di x).

Operatori logici

1. **Connettivo AND ($\wedge$):**

   La congiunzione $p \wedge q$ è vera solo se entrambe p e q sono vere.

   - Esempio: p: "Un decennio ha 10 anni" (vero)
   - q: "Un millennio ha 100 anni" (falso)

- p ∧ q: falso

2. **Connettivo OR (∨):**

   La disgiunzione p∨q è vera se almeno una tra p o q è vera.

   - Esempio:
   - p: "Un millennio ha 100 anni" (falso)
   - q: "Un millennio ha 1000 anni" (vero)
   - p ∨ q: vero.

3. **Negazione (¬):**

   ¬p è la proposizione "non p" e ha valore opposto rispetto a p.

   - Esempio:
   - p: "Parigi è la capitale dell'Inghilterra" (falso)
   - ¬p: "Parigi non è la capitale dell'Inghilterra" (vero).

Tabelle di verità

Le tabelle di verità rappresentano tutte le possibili combinazioni di valori di verità per le proposizioni coinvolte e i risultati degli operatori applicati.

Esempio di tabella per p ∧ q:

p	q	p ∧ q
V	V	V
V	F	F
F	V	F
F	F	F

Ordine di precedenza

1. Valutare prima ¬ (negazione).
2. Poi ∧ (AND).
3. Infine ∨ (OR).

Esempi pratici:

- **Combinazioni logiche:**

  Se p: falso, q: vero, r: falso, allora:

  ¬p ∨ q ∧ r ¬p :

  1. ¬p: vero.
  2. q ∧ r: falso.
  3. ¬p ∨ q ∧ r ¬p: vero.

- **Motori di ricerca:**

  I motori di ricerca (es. Google) usano operatori logici:

  - Spazio = AND.
  - OR = OR logico.
  - "-" = NOT.
  - Esempio: "Shonda Rhimes" Grey's OR Scandal -Murder"

1.3 - Conditional propositions and logical equivalence

Proposizioni condizionali

- Una proposizione condizionale assume la forma: **se p allora q** (scritta simbolicamente come p → q).
- **Ipotesi (antecedente):** p.
- **Conclusione (conseguente):** q.

Valutazione delle proposizioni condizionali

La verità di p → q è determinata come segue:

1. Se p è **vero** e q è **vero**, allora p → q è **vero**.
2. Se p è **vero** e q è **falso**, allora p → q è **falso**.

3. Se p è **falso**, p → q è sempre **vero** (vacuamente vero).

Bicondizionale

- $p \iff q$ (se e solo se p, allora q) è vero solo quando p e q hanno lo stesso valore di verità.

Equivalenza logica

Due proposizioni P e Q sono logicamente equivalenti ($P \equiv Q$) se hanno gli stessi valori di verità per tutte le combinazioni possibili dei loro componenti.

Negazione di una proposizione condizionale

- La negazione di p → q è logicamente equivalente a $p \land \neg q$.

Esempi

1. **Proposizione condizionale:**
   - *Se Jerry riceve una borsa di studio (p), allora andrà al college (q).*
   - Negazione: *Jerry riceve una borsa di studio ma non va al college.* ($p \land \neg q$).
2. **Equivalenze logiche:**
   - $\neg(p \lor q) \equiv \neg p \land \neg q$(Prima legge di De Morgan).
   - $\neg(p \land q) \equiv \neg p \lor \neg q$(Seconda legge di De Morgan).
3. **Esempio pratico (Java):**
   - $x<10 \lor x>20$x < 10 è quivalente a $\neg(x \geq 10 \land x \leq 20)$.

1.4 - Arguments and rules of inference

Definizioni fondamentali:

- Un argomento deduttivo consiste in ipotesi (premesse) e una conclusione.

- Un argomento è valido se, quando tutte le ipotesi sono vere, anche la conclusione deve essere vera.
- La validità dipende dalla forma dell'argomento, non dal contenuto.

Forma di un argomento: $p_1, p_2, \ldots, p_n \implies$ q

- Le ipotesi sono $p_1, p_2, \ldots, p_n$, mentre q è la conclusione.
- La validità implica che, se tutte le ipotesi sono vere, q non può essere falso.

Esempi di regole di inferenza:

1. **Modus Ponens**:

   p→q, p $\implies$ q (Se p→q e p sono vere, allora q è vera.)

2. **Modus Tollens**:

   p→q, ¬q $\implies$ ¬p (Se p→q è vera e q è falsa, allora p è falsa.)

3. **Sillogismo Ipotetico:**

   p→q, q→r $\implies$ p→r

4. **Disgiunzione**:

   p∨q, ¬p $\implies$ q (Se p∨q è vera e p è falsa, allora q è vera.)

5. **Coniugazione**:

   p, q $\implies$ p∧q

Esempi di argomenti validi:

1. **Bug nei moduli:**
   - Ipotesi: Il bug è nel modulo 17 o 81 (p∨q), il bug è un errore numerico (r), e il modulo 81 non ha errori numerici (r→¬q).
   - Conclusione: Il bug è nel modulo 17 (p).
   - Passaggi:
     - Da r→¬q e r (Modus Ponens) segue ¬q.
     - Da p∨q e ¬q (Disgiunzione) segue p.
2. **Squadra di football:**

- Ipotesi: Se i Chargers hanno un buon linebacker (p), possono battere i Broncos (q), i Jets (r) e i Dolphins (s).
- Conclusione: Possono battere Jets e Dolphins ($r \wedge s$).
- Passaggi:
  - Usando Modus Ponens e Sillogismo Ipotetico, si dimostra r e s.
  - Applicando la congiunzione, si ottiene $r \wedge s$.

Errori comuni:

1. **Fallacia dell'affermazione della conclusione:**

   $p \rightarrow q$, $q \not\Rightarrow p$ (Sapere che q è vero non implica che p sia vero.)

2. **Fallacia della negazione della premessa:**

   $p \rightarrow q, \neg p \not\Rightarrow \neg q$

1.5 - Quantifiers

- **Funzioni proposizionali**:
  - Una funzione proposizionale P(x) è una frase che dipende da una variabile x. Diventa una proposizione (cioè vera o falsa) quando x assume un valore dalla "dominio del discorso" D.
  - Esempio: P(n): "n è un numero dispari" con D=Z+.
- **Quantificatore universale ($\forall$)**:
  - Espressione: $\forall x, P(x)$ ("per ogni x, P(x)").
  - È vera se P(x) è vera per tutti i valori di x nel dominio D.
  - Un controesempio è un valore di x che rende P(x) falso, dimostrando che $\forall x, P(x)$ è falso.
- **Quantificatore esistenziale ($\exists$)**:
  - Espressione: $\exists x, P(x)$ ("esiste almeno un x tale che P(x)").
  - È vera se almeno un x in D rende P(x) vera.

- È falsa se P(x) è falso per ogni x in D.

- **Esempi e applicazioni**:
  - Universale: $\forall x \in R, x^2 \geq 0$ (vero perché il quadrato di un numero reale è sempre non negativo).
  - Esistenziale: $\exists x \in R, x^2 + 1 = 2$ (vero perché esiste almeno un x, ad esempio x=1).
- **Verifica tramite pseudocodice**:
  - Per $\forall x, P(x)$, si verifica ogni elemento del dominio e si ritorna **falso** al primo controesempio.
  - Per $\exists x, P(x)$, si interrompe la ricerca al primo valore vero.
- **Variabili libere e vincolate**:
  - Una variabile x è libera in P(x) (non è legata a un quantificatore).
  - Quando x è vincolata ($\forall x$ o $\exists x$), la frase diventa una proposizione con un valore di verità.

1.6 - Nested quantifiers

I quantificatori annidati si riferiscono a espressioni che utilizzano più di un quantificatore, come "per ogni x" seguito da "per ogni y", o "esiste x" seguito da "per ogni y". Questi quantificatori sono utilizzati per esprimere affermazioni su insiemi che coinvolgono più variabili.

Esempi:

1. **La somma di due numeri reali positivi è positiva**:
   - Tradotto simbolicamente: $\forall x \forall y ((x > 0) \land (y > 0) \rightarrow (x + y > 0))$
   - Questo afferma che, per ogni coppia di numeri reali x e y, se entrambi sono positivi, la loro somma è positiva.
2. **Restituire una proposizione simbolica**:
   - Ad esempio, l'affermazione “Ogni m ha un n tale che m < n” può essere scritta come $\forall m \exists n (m < n)$. Questo significa che, per ogni m,

esiste almeno un n tale che m < n (ad esempio, non esiste l'intero più grande).

3. **"Everybody loves somebody" (Tutti amano qualcuno)**:
   - Tradotto simbolicamente: $\forall x \exists y\ L(x, y)$, dove L(x, y) significa “x ama y”.
   - In parole, per ogni persona x, esiste almeno una persona y che x ama.

Quantificatori Universali ed Esistenziali

- **$\forall x \forall y\ P(x, y)$**: Questo è vero se per ogni x e per ogni y nella loro rispettiva dominio, P(x, y) è vero. Ad esempio, $\forall x \forall y\ ((x > 0) \wedge (y > 0) \rightarrow (x + y > 0))$ è vero, poiché per ogni coppia di numeri reali positivi, la somma è positiva.
- **$\forall x \exists y\ P(x, y)$**: Questo è vero se, per ogni x, esiste almeno un y tale che P(x, y) è vero. Un esempio è $\forall x \exists y(x + y = 0)$, che è vero perché, per ogni numero reale x, possiamo trovare un numero y (y = -x) tale che x + y = 0.

Negazione di Quantificatori

Per negare un'affermazione che contiene quantificatori, si applicano le leggi di De Morgan:

- La negazione di **$\forall x \exists y\ P(x, y)$** è **$\exists x \forall y\ \neg P(x, y)$**, che significa che esiste almeno un x per il quale P(x, y) è falso per ogni y.

Esempio:

- La negazione di $\forall x \exists y(x + y = 0)$ è $\exists x \forall y(x + y \neq 0)$, cioè esiste almeno un x tale che per ogni y, x + y non è mai uguale a 0.

Chapter 2 - Proofs

2.1 - Mathematical systems, direct proofs and counterexemples

Sistemi Matematici:

Un sistema matematico è costituito da assiomi, definizioni e termini indefiniti.

- Gli assiomi sono considerati veri
- Le definizioni creano nuovi concetti a partire da quelli esistenti.
- I teoremi sono derivati da questi assiomi e definizioni
- Esempi includono la geometria euclidea e il sistema dei numeri reali.

Tipi di Teoremi:

- **Teoremi:** Proposizioni che sono dimostrate vere.
- **Lemmi:** Teoremi utili che potrebbero non essere interessanti di per sé, ma aiutano a dimostrare altri teoremi.
- **Corollari:** Teoremi che derivano facilmente da altri teoremi.

Prove Dirette:

Una prova diretta dimostra una proposizione assumendo che le ipotesi siano vere e mostrando che la conclusione segue logicamente.
La proposizione da dimostrare ha spesso la forma: "Per tutti $x_1, x_2, \ldots, x_n$, se p($x_1, x_2, \ldots, x_n$), allora q($x_1, x_2, \ldots, x_n$)."
Se p($x_1, x_2, \ldots, x_n$) è vero, la prova prosegue mostrando che anche q($x_1, x_2, \ldots, x_n$) è vero.

Esempi di Prove Dirette:

- **Numeri Pari e Dispari:** Le definizioni di numeri pari e dispari aiutano a dimostrare la proposizione "se m è dispari e n è pari, allora m + n è dispari."
- **Operazioni su Insiemi:** Una prova diretta può essere utilizzata anche per dimostrare relazioni tra insiemi, come "X ∩ (Y − Z) = (X ∩ Y) − (X ∩ Z)."
- **Definizione di Minimo:** Usando la funzione minimo, una prova può dimostrare che se d=min(d1,d2) e x≤d, allora x≤d1 e x≤d2.

2.2 - More methods of proof

- **Dimostrazione per contraddizione:**

- Si assume che l'ipotesi p sia vera e la conclusione q sia falsa (¬q).
- Usando p e ¬q insieme a assiomi, definizioni, teoremi precedenti e regole di inferenza, si arriva a una contraddizione ($r \wedge \neg r$).
- Concludiamo che q deve essere vera.
- Questo tipo di dimostrazione è anche chiamato indiretta, poiché si giunge a una contraddizione e si conclude che q è vero.

- **Dimostrazione per contrapposizione:**
  - Se si prova $p \to q$ per contraddizione, si ottiene $\neg q \to \neg p$, che è equivalente a $p \to q$.
  - Questo approccio è chiamato dimostrazione per contrapposizione.
- **Dimostrazione per casi:**
  - Quando l'ipotesi si divide naturalmente in vari casi, si usa la dimostrazione per casi.
  - Si dimostra che, se p1 o p2 (i vari casi) sono veri, allora la conclusione q è vera per ciascun caso
  - La dimostrazione per casi è giustificata dal fatto che le implicazioni nei vari casi sono equivalenti a una singola implicazione.
- **Dimostrazioni di equivalenza:**
  - Per le proposizioni del tipo "p se e solo se q", si dimostra che $p \to q$ e $q \to p$ sono veri.
  - Questo dimostra che p è equivalente a q.
- **Dimostrazioni di esistenza:**
  - Una **prova di esistenza** serve a dimostrare che esiste almeno un elemento che soddisfa una certa proprietà, cioè $\exists x P(x)$.
  - Un modo per provare $\exists x P(x)$ è mostrare un elemento a del dominio che rende P(a) vero.

Esempi:

1. **Esempio 2.2.12**

Enunciato: Esiste un numero reale x tale che a<x<b, dove a e b sono numeri reali con a<b.

Prova: Si prende il numero x=$(a+b)/2$, che è il punto medio tra a e b. Chiaramente, a<x<b.

2. **Prove costruttive vs non costruttive**:
   - Una **prova costruttiva** di ∃xP(x) mostra direttamente un elemento a che soddisfa P(a), come negli esempi sopra.
   - Una **prova non costruttiva** di ∃xP(x) non mostra esplicitamente un tale elemento, ma lo dimostra indirettamente, ad esempio usando una prova per contraddizione.

2.3 - Resolution proofs

La **risoluzione** è una tecnica di prova proposta da J. A. Robinson nel 1965. Si basa su una regola semplice:

- Se p∨q e ¬p∨r sono veri, allora q∨r è vero.

Clausola (Clauses)

Una **clausola** è un'espressione formata da termini separati da "oppure" (∨), dove ogni termine è una variabile o la negazione di una variabile.

Esempi:

- a∨b∨¬c∨d è una clausola.
- xy∨w∨¬z**non** è una clausola (poiché xy è una combinazione di variabili, non una singola variabile).
- p→q **non** è una clausola (poiché i termini sono separati da →).

Prova Diretta tramite Risoluzione

La **prova diretta** con risoluzione si procede applicando ripetutamente la regola di risoluzione fino a derivare la conclusione. Durante l'applicazione della regola, p deve essere una singola variabile, ma q e r possono essere espressioni complesse.

Esempio 2.3.4:
Dimostrare:

1. $a \vee b$
2. $\neg a \vee c$
3. $\neg c \vee d \therefore b \vee d$

Prova:

- Applicando la regola di risoluzione a 1 e 2 otteniamo $b \vee c$.
- Applicando la regola di risoluzione a 3 e $b \vee c$ otteniamo la conclusione $b \vee d$

Casi Speciali della Risoluzione

- Se $p \vee q$ e $\neg p$ sono veri, allora q è vero.
- Se p e $\neg p \vee r$ sono veri, allora r è vero.

Esempio 2.3.5:
Dimostrare:

1. a
2. $\neg a \vee c$
3. $\neg c \vee d \therefore d$

Prova:

- Applicando la regola di risoluzione a 1 e 2 otteniamo c.
- Applicando la regola di risoluzione a 3 e c otteniamo d.

Sostituzione di Ipotesi Non Clausole

Se un'ipotesi non è una clausola, si deve sostituirla con un'espressione equivalente che sia una clausola, usando leggi come le leggi di De Morgan. Ad esempio, $\neg(a \vee b)$diventa $\neg a \vee \neg b$

Esempio 2.3.6:
Dimostrare:

1. $a \vee \neg bc$

2. $\neg(a \vee d) \therefore \neg b$

Prova:

- Utilizzando la legge di distribuzione $(a \vee bc \equiv (a \vee b)(a \vee c))$, sostituire la prima ipotesi con $a \vee \neg b$ e $a \vee c$.
- Applicando la risoluzione alle ipotesi risultanti, otteniamo la conclusione $\neg b$.

Combinazione con la Prova per Contraddizione

La risoluzione può essere combinata con la prova per contraddizione. Si nega la conclusione e si aggiungono le clausole corrispondenti alla negazione della conclusione agli assunti. Applicando la risoluzione, si cerca una contraddizione.

Esempio 2.3.7:
Ripetere la prova dell'Esempio 2.3.4 utilizzando la risoluzione con la prova per contraddizione.

Prova:

- Si nega la conclusione $(b \vee d)$ e si ottiene $\neg b \vee \neg d$
- Aggiungendo le clausole $\neg b$ e $\neg d$ agli assunti, si applica la risoluzione fino a derivare una contraddizione, completando la prova.

Suggerimenti per la Risoluzione

1. Sostituire le ipotesi o la conclusione che non sono clausole con clausole equivalenti.
2. Applicare la risoluzione ripetutamente per derivare la conclusione.
3. La risoluzione può essere combinata con la prova per contraddizione.

2.4 - Mathematical induction

Per dimostrare che una dichiarazione S(n) è vera per tutti gli interi positivi n, usiamo i seguenti due passaggi:

- **Passaggio Base:** Dimostrare che la dichiarazione è vera per il primo valore, tipicamente n=1. Questo verifica che il caso base sia corretto.

- **Passaggio Induttivo:** Assumere che la dichiarazione sia vera per un arbitrario n=k. Questa è chiamata l'Ipotesi Induttiva. Poi, dimostrare che la dichiarazione è vera per n=k+1. Questo mostra che la verità della dichiarazione per n=k implica la sua verità per n=k+1.

Se entrambi i passaggi vengono completati con successo, la dichiarazione è provata per tutti gli interi positivi n.

Esempio 1: Somma dei primi n interi

La somma dei primi n interi Sn=1+2+...+n può essere provata soddisfare la formula:

$Sn = \frac{n(n+1)}{2}$

Passaggio Base: Per n=1, verifichiamo che la formula dia $S1 = \frac{1(1+1)}{2} = 1$, che è vero.

Passaggio Induttivo: Supponiamo che la formula sia valida per n=k. Mostriamo che è valida per n=k+1. Usando l'assunzione e semplificando, verifichiamo che la formula è valida per k+1.

Esempio 2: Disuguaglianza del fattoriale

Dobbiamo dimostrare che $n! \geq 2^{n-1}$ per tutti n≥1n.

Passaggio Base: Per $n = 1, 1! = 1 \geq 2^{1-1} = 1$, che è vero.

Passaggio Induttivo: Supponiamo che $k! \geq 2^{k-1}$ per un qualche k≥1. Dobbiamo provare che $(k + 1)! \geq 2^k$. Questo implica esprimere (k+1)! in termini di k! e applicare l'ipotesi induttiva.

Esempio 3: Somma geometrica

Dobbiamo dimostrare che la somma di una serie geometrica è:

$a + ar + ar^2 + \cdots + ar^n = \frac{a(r^{n+1}-1)}{r-1}$ per r≠1

Passaggio Base: Per n=0, la formula si semplifica a a, che corrisponde alla somma della serie.

Passaggio Induttivo: Supponiamo che la formula sia valida per nnn, poi dimostriamo che è valida per n+1.

Esempio 4: Divisibilità per 4

Dobbiamo dimostrare che 5n−1 è divisibile per 4 per tutti n≥1n \geq 1n≥1.

Passaggio Base: Per n=1, 5(1)−1=45(1) - 1 = 45(1)−1=4, che è divisibile per 4.

Passaggio Induttivo: Supponiamo che 5k−1sia divisibile per 4. Dobbiamo mostrare che 5(k+1)−1 è anche divisibile per 4. Usando l'assunzione, esprimiamo 5(k+1)−1 in termini di 5k−1 e verifichiamo che il risultato è divisibile per 4.

Esempio 5: Cardinalità dell'insieme delle potenze

Dobbiamo dimostrare che l'insieme delle potenze di un insieme X con n elementi ha 2^n sottoinsiemi.

Passaggio Base: Per n=0, l'insieme X è vuoto, e il suo insieme delle potenze ha 1 sottoinsieme, che è 2^0

Passaggio Induttivo: Supponiamo che il risultato sia valido per un insieme di dimensione n. Mostriamo che per un insieme di dimensione n+1, l'insieme delle potenze contiene 2^{n+1}sottoinsiemi, dividendo i sottoinsiemi in quelli che contengono un particolare elemento e quelli che non lo contengono.

2.5 - Strong form of induction and the Well-ordering property

Il **Principio di Induzione Forte** è una versione estesa della **matematica induzione** che permette di assumere la veridicità di tutte le affermazioni precedenti, non solo di quella immediatamente precedente. Questo può rendere il passo induttivo più semplice in alcuni casi. Formalmente, se vogliamo dimostrare che una proposizione S(n) è vera per ogni intero $n \geq n_o$, dobbiamo:

1. Verificare che $S(n_o)$ è vera (Passo Base).
2. Supporre che S(k) sia vera per tutti gli interi k con $n_o \leq k < n$.
3. Usare questa assunzione per dimostrare che S(n) è vera.

Questo approccio è utile quando il passo induttivo richiede l'assunzione di molteplici casi precedenti, come negli esempi che seguono.

Esempi:

1. **Posta con francobolli da 2 e 5 centesimi:** Si dimostra che è possibile ottenere qualsiasi importo di almeno 4 centesimi utilizzando solo francobolli da 2 e 5 centesimi. La dimostrazione usa l'induzione forte, assumendo la veridicità delle affermazioni precedenti per ottenere quella successiva.
2. **Sequenze ricorsive:** In un esempio in cui una sequenza è definita ricorsivamente, si usa l'induzione forte per dimostrare che $c_n < 2^n$ per ogni $n \geq 1$.
3. **Moltiplicazione con parentesi:** Si dimostra che, indipendentemente da come vengono inserite le parentesi in una moltiplicazione di n numeri, sono necessarie esattamente n - 1 moltiplicazioni.

Proprietà di Ben Ordinamento: Ogni insieme non vuoto di numeri interi non negativi ha un elemento minimo. Questa proprietà è usata per dimostrare il **Teorema del Quotiente e del Resto**, che afferma che quando si divide un numero n per un divisore positivo d, esistono un quoziente q e un resto r che soddisfano $0 \leq r < d$. Inoltre, q e r sono unici.

In generale, l'induzione forte è utile quando ci sono più casi da considerare per provare una dichiarazione per un intero n.

Chapter 3 - Functions Sequences and Relations

3.1 - Functions

1. Algoritmo di Luhn

- Utilizzato per verificare errori nei numeri delle carte di credito.
- Procedura:
    1. Raddoppia ogni altra cifra (partendo da destra, escluso il check digit).
    2. Somma le cifre dei risultati (ad esempio, 18 diventa 1+8).
    3. Calcola la somma totale.

4. Il check digit è determinato come $[10 - (S mod 10)] mod 10$, dove S è la somma calcolata.

- Esempi mostrano come questo algoritmo rileva errori comuni, come:
  - La modifica di un singolo numero.
  - La trasposizione di due cifre adiacenti.

2. Definizione di Funzione

- Una **funzione** è una relazione tra due insiemi, X (dominio) e Y (codominio), che assegna a ogni elemento di X un unico elemento in Y.
- Rappresentazioni:
  - Insiemi di coppie ordinate (x,y).
  - Diagrammi a frecce.
  - Notazione f(x).

3. Esempi di Funzioni

- Funzioni semplici come $f(x) = x^2$ o $R(x) = \frac{1}{x}$.
- Controllo se un insieme è una funzione:
  - Ogni elemento del dominio deve avere un'unica immagine nel codominio.

4. Uso del Modulo (\mod)

- Determinazione del resto nella divisione tra interi.
- Applicazioni:
  - Algoritmo di Luhn: calcolo del check digit.
  - Calcolo dei giorni della settimana (365mod7=1).
  - **Hash Functions**: gestione delle collisioni in memoria.

5. Hash Functions

- Trasformano dati in numeri per determinare la posizione in memoria.
- Esempio: h(n) = n mod 11.

- Affrontano problemi come le **collisioni**, quando due numeri si mappano sulla stessa posizione.

Floor e Ceiling

- **Floor** (pavimento, ⌊x⌋): Il massimo intero ≤x.
- **Ceiling** (soffitto, ⌈x⌉): Il minimo intero ≥x
- Esempi:
  - ⌊8.3⌋=8, ⌈9.1⌉=10
  - ⌊−8.7⌋=−9, ⌈−11.3⌉=−11

Applicazioni

- Funzioni pratiche come la calcolazione delle tariffe postali si basano su queste funzioni per arrotondare pesi (esempio con P(w)=92+20·⌈w−1⌉).

Definizioni principali

1. **Iniettività (One-to-one)**:
   - Una funzione f è iniettiva se per ogni $x1 \neq x2$ si ha $f(x1) \neq f(x2)$.
   - Esempio: $f(n) = 2n + 1$ è iniettiva.
   - Controesempio: $f(n) = 2n - n^2$ non è iniettiva poiché $f(2) = f(4)$.
2. **Suriettività (Onto)**:
   - Una funzione f è suriettiva se ogni elemento del codominio Y ha almeno una controimmagine in X.
   - Esempio: $f(x) = \frac{1}{x^2} da X = R \setminus 0 a Y = R+$ è suriettiva.
   - Controesempio: $f(n) = 2n - 1$ non è suriettiva poiché i numeri pari non appartengono al codominio.
3. **Biezione**:
   - Una funzione è biettiva se è sia iniettiva sia suriettiva.

- Esempio: La funzione $f(x) = 2x$ da R a R+ è biettiva. La sua inversa è $f^{-1}(y) = log_2(y)$.

Composizione di funzioni

- La composizione f∘g combina due funzioni $f : Y \to Z$ e $g : X \to Y$ in $f \circ g(x) = f(g(x))$.
- Esempio: Se $g(x) = a$ e $f(a) = y$, allora $f \circ g(x) = y$.

Teoremi

1. **Quoziente e resto**:
   - Per interi n e d>0, $n = dq + r$ con $0 \leq r < d$. Il quoziente $q = \lfloor n/d \rfloor$, e il resto è$r = n - dq$.
   - Applicazione: Calcolo di n mod d.
2. **Inversa di una funzione biettiva**:
   - Una funzione biettiva f ha una inversa $f - 1$, che si ottiene invertendo i ruoli di dominio e codominio.

3.2 - Sequences and string

1. **Definizione di sequenza**:

- Una sequenza è una funzione il cui dominio è un sottoinsieme di numeri interi.
- Il termine "indice" si riferisce alla posizione dell'elemento nella sequenza.
- Le sequenze possono essere finite o infinite.

2. **Esempi di sequenze**:

- Sequenze definite da regole matematiche (es. $sn = 2n$).
- Sequenze definite da parole o caratteri (es. lettere della parola "digital").
- Sequenze definite in intervalli specifici.

3. **Proprietà delle sequenze**:

- Una sequenza è **crescente** se ogni elemento successivo è maggiore del precedente.
- È **decrescente** se ogni elemento successivo è minore.
- Una sequenza è **non decrescente** se gli elementi successivi sono maggiori o uguali, e **non crescente** se sono minori o uguali.

4. **Sotto-sequenze**:

    - Si ottengono selezionando un sottoinsieme di termini di una sequenza originale, mantenendo l'ordine.
    - Ad esempio, dalla sequenza a, a, b, c, q, b, c è una sotto-sequenza.

5. **Operazioni su sequenze**:

    - Somma (Σ) e prodotto (Π) degli elementi di una sequenza in un intervallo specifico.
    - È possibile cambiare gli indici e i limiti nella notazione sommatoria o produttoria.

6. **Esempi specifici**:

- Calcolo di termini specifici sostituendo indici nella formula della sequenza.
- Uso di relazioni ricorsive per definire sequenze.
- Riconoscimento di proprietà come monotonia o formazione di sotto-sequenze.

7. **Applicazioni pratiche**:

- La tabella iniziale mostra il costo del taxi come esempio di sequenza finita.

- Le sequenze possono rappresentare modelli discreti in vari contesti matematici e applicativi.

3.3 - Relations

Una **relazione** tra due insiemi può essere vista come un insieme di coppie ordinate. Formalmente, una relazione R da un insieme X a un insieme Y è un sottoinsieme del prodotto cartesiano X×Y. Se $(x, y) \in R$, si dice che xxx è in relazione con y, indicato come xRy.

Definizione di Relazione

- Una relazione è una collezione di coppie ordinate.
- Se $X = Y$, la relazione è detta **binaria** su X.

Esempi di Relazioni

1. **Tabella di relazioni**: Elenca quali elementi di X sono in relazione con quali elementi di Y. Es: "Bill è iscritto a Computer Science e Arte."
2. **Relazione definita da una regola**: Si specificano i criteri per cui (x,y) appartiene a R. Es: "x divide y."

Rappresentazione di Relazioni

- **Tabella**: Una matrice o elenco che mostra le coppie in R.
- **Digrafi**: Grafi diretti con nodi che rappresentano gli elementi e frecce per ogni coppia (x, y) in R. Gli **anelli** rappresentano (x, x).

Proprietà delle Relazioni

Le relazioni possono avere alcune proprietà caratteristiche:

1. **Riflessiva**

   R è riflessiva se $(x, x) \in R(x, x)$ per ogni $x \in X$.

   Es: La relazione $x \leq y$ è riflessiva perché ogni $x \leq x$.

2. **Simmetrica**

R è simmetrica se $(x, y) \in R \implies (y, x) \in R$.

Es: Una relazione che collega "amici reciproci" è simmetrica.

3. **Antisimmetrica**

   R è antisimmetrica se $(x, y) \in R$ e $(y, x) \in R \implies x = y$.

   Es: La relazione $x \leq y$ è antisimmetrica.

4. **Transitiva**

   R è transitiva se $(x, y) \in R$ e $(y, z) \in R \implies (x, z) \in R$.

   Es: La relazione $x \leq y$

Confronti tra Proprietà

- **Simmetria e Antisimmetria**: Non sono mutuamente esclusive; una relazione può essere sia simmetrica che antisimmetrica.
- **Riflessività e Transitività**: Relazioni riflessive e transitive formano spesso strutture matematiche ben definite, come relazioni di ordine parziale.

Applicazioni

- **Funzioni**: Sono un caso particolare di relazioni, con l'ulteriore vincolo che ogni elemento del dominio è associato a uno e un solo elemento del codominio.
- **Grafi e Teoria dei Grafi**: I digrafi sono strumenti utili per visualizzare e analizzare relazioni.

3.4 - Equivalence relations

Partizione e Relazioni

Un insieme X può essere suddiviso in sottoinsiemi non vuoti chiamati **partizioni**, tali che ogni elemento di X appartiene esattamente a un sottoinsieme. Ad esempio, un insieme di 10 palline colorate può essere partizionato in base al colore, ottenendo tre sottoinsiemi R, B, G. Una partizione S di X permette di

definire una relazione R, in cui xRy significa che x e y appartengono allo stesso sottoinsieme di S.

Proprietà della Relazione

Se R è definita da una partizione, allora è:

- **Riflessiva**: ogni elemento è in relazione con se stesso.
- **Simmetrica**: se xRy, allora yRx.
- **Transitiva**: se xRy e yRz, allora xRz.

Tali relazioni sono chiamate **relazioni di equivalenza**.

Definizione Formale

Una relazione su un insieme X è detta di equivalenza se soddisfa le proprietà di riflessività, simmetria e transitività.

Classi di Equivalenza

Dati una relazione di equivalenza R e un elemento $a \in X$, la **classe di equivalenza** di a, denotata $[a]$, è l'insieme di tutti gli elementi $x \in X$ tali che xRa. L'insieme delle classi di equivalenza forma una partizione di X.

Esempi

1. **Partizioni di numeri**: Una relazione definita come “avere lo stesso resto quando divisi per 3” su $\{1, 2, ..., 10\}$ genera classi di equivalenza: $\{1, 4, 7, 10\}$, $\{2, 5, 8\}$, $\{3, 6, 9\}$.
2. **Relazione su colori**: Se R è la relazione "avere lo stesso colore", le palline di uno stesso colore formano una classe di equivalenza.

Teorema

Se R è una relazione di equivalenza su un insieme finito X, e ogni classe di equivalenza ha r elementi, il numero totale di classi è $\mid X \mid /r$.

Applicazioni

Le relazioni di equivalenza permettono di semplificare problemi suddividendo un insieme in gruppi distinti di elementi equivalenti. Questi gruppi (classi di equivalenza) possono essere utilizzati in diversi contesti, come la suddivisione per colori, resti, o altre proprietà condivise.

3.5 - Matrices of relations

Una matrice è un modo utile per rappresentare una relazione R tra due insiemi X e Y. Le righe rappresentano gli elementi di X, le colonne quelli di Y, e l'elemento in posizione (x, y) vale 1 se xRy, 0 altrimenti. L'ordine degli elementi nei due insiemi influenza la rappresentazione della matrice.

Caratteristiche principali delle matrici di relazioni:

1. **Riflessione**:

   Una relazione è riflessiva se e solo se tutti gli elementi sulla diagonale principale della matrice sono 1.

2. **Simmetria**:

   Una relazione è simmetrica se e solo se la matrice è simmetrica rispetto alla diagonale principale.

3. **Antisimmetria**:

   Una relazione è antisimmetrica se non esistono i, j tali che $A[i, j] = 1$, a meno che $i = j$.

4. **Composizione di relazioni**:

   Se R_1 è una relazione da X a Y e R_2 è una relazione da Y a Z, il prodotto matriciale $A_1 A_2$ rappresenta la composizione $R2 \circ R1$. Per ottenere la matrice della composizione, si sostituiscono i valori diversi da zero con 1.

5. **Transitività**:

   Per verificare se una relazione è transitiva, si calcola il quadrato della matrice $A2$. La relazione è transitiva se, per ogni elemento non nullo in $A2$, il corrispondente elemento in A è anch'esso non nullo.

Esempi:

- La matrice di una relazione riflessiva avrà tutti 1 sulla diagonale principale.
- Una matrice simmetrica è identica rispetto alla sua trasposta.
- Se il prodotto matriciale genera valori che non corrispondono a 1 nella matrice originale, la relazione non è transitiva.

Consigli per la risoluzione dei problemi:

- La matrice è un'alternativa utile per rappresentare una relazione.
- Analizza la diagonale principale per verificare la riflessione.
- Controlla la simmetria per identificare relazioni simmetriche.
- Usa il prodotto matriciale per comporre relazioni e testare la transitività.

3.6 - Relational Databases

Relazioni n-arie

Una relazione binaria ha due colonne, ma è possibile estendere il concetto a relazioni con un numero arbitrario di colonne, chiamate relazioni n-arie. Ad esempio, una tabella con 4 colonne rappresenta una relazione 4-aria, come mostrato nella Tabella 3.6.1, che elenca ID, nomi, posizioni e età dei giocatori.

TABLE 3.6.1 ■ PLAYER

ID Number	*Name*	*Position*	*Age*
22012	Johnsonbaugh	c	22
93831	Glover	of	24
58199	Battey	p	18
84341	Cage	c	30
01180	Homer	1b	37
26710	Score	p	22
61049	Johnsonbaugh	of	30
39826	Singleton	2b	31

Attributi e chiavi

Le colonne di una relazione n-aria sono chiamate **attributi**, ognuno associato a un dominio (insieme di valori validi).

Una **chiave** è un attributo (o una combinazione di attributi) che identifica univocamente una tupla. Ad esempio, l'attributo "ID Number" è una chiave nella Tabella 3.6.1, ma "Name" non lo è, poiché due giocatori possono avere lo stesso nome.

Database e gestione

Un database è una raccolta di record gestiti da un sistema di gestione database (DBMS), che consente di archiviare e interrogare grandi quantità di dati. Il modello relazionale, introdotto da E.F. Codd, si basa sul concetto di relazioni n-arie.

Operatori relazionali

1. **Selezione**: Filtra le righe in base a una condizione.

   Esempio: Per selezionare i giocatori con posizione "c" nella Tabella 3.6.1:

   `PLAYER [Position = c]`

   Risultato: {(22012, Johnsonbaugh, c, 22), (84341, Cage, c, 30)}.

2. **Proiezione**: Seleziona colonne specifiche eliminando duplicati.

   Esempio: Per ottenere i nomi e le posizioni:

   `PLAYER [Name, Position]`

   Risultato: {(Johnsonbaugh, c), (Glover, of), ...}.

3. **Join**: Combina righe da due tabelle in base a una condizione.

   Esempio: Collegare la Tabella 3.6.1 con la Tabella 3.6.2 sulla base dell'uguaglianza tra "ID Number" e "PID".

TABLE 3.6.2 ■ ASSIGNMENT

PID	*Team*
39826	Blue Sox
26710	Mutts
58199	Jackalopes
01180	Mutts

   Risultato (Tabella 3.6.3): Colonne combinate, con informazioni sui team di ogni giocatore.

TABLE 3.6.3 ■ PLAYER [ID Number = PID] ASSIGNMENT

ID Number	*Name*	*Position*	*Age*	*Team*
58199	Battey	p	18	Jackalopes
01180	Homer	1b	37	Mutts
26710	Score	p	22	Mutts
39826	Singleton	2b	31	Blue Sox

Esempi di query

1. **Trovare i nomi dei giocatori che appartengono a un team**:
   - Effettuare un join delle tabelle (Tabella 3.6.1 e Tabella 3.6.2).
   - Proiettare sull'attributo "Name".

     Risultato: {Battey, Homer, Score, Singleton}.
2. **Trovare i nomi dei giocatori del team "Mutts"**:
   - Selezionare dalla Tabella 3.6.2 le righe con "Team = Mutts".
   - Effettuare un join con la Tabella 3.6.1.
   - Proiettare sull'attributo "Name".

     Risultato: {Homer, Score}.

Chapter 5 - Introduction to number theory

5.1 - Divisors

- **Definizione di Divisibilità**

  Dividere: Si dice che d divide n(denotato $d|n$) se esiste un intero q tale che $n = dq$. In questo caso, d è un divisore o fattore di n.

  Esempio: $3 \mid 21$ perché $21 = 3 \times 7$, dove 7 è il quoziente.

- **Teorema del Quoziente e del Resto**

  Il teorema garantisce l'esistenza di interi unici q e r tali che $n = dq + r$, con $0 \leq r < d$.

  Se $r = 0$, allora d divide n.

- **Proprietà dei Divisori (Teorema 5.1.3)**

  Se $d \mid m$ e $d \mid n$, allora $d \mid (m+n)$ e $d \mid (m-n)$.

- **Numeri Primi e Composti**

  Primo: Un numero maggiore di 1 che non ha divisori diversi da 1 e da se stesso (ad esempio, 23 è primo).

  Composto: Un numero maggiore di 1 che non è primo (ad esempio, 34 è composto perché è divisibile per 17).

  Test di primalità: Per verificare se un numero 1 è primo, basta controllare la divisibilità per gli interi da 2 a n. Questo riduce significativamente il numero di verifiche.

- **Teorema 5.1.7 (Test di Primalità)**

  Un intero positivo $n > 1$ è composto se e solo se ha un divisore d tale che $2 \leq d \leq \sqrt{n}$.

  Questo riduce il numero di verifiche di divisibilità necessarie per verificare se un numero è primo.

- **Algoritmo per il Test di Primalità (Algoritmo 5.1.8)**

  L'algoritmo verifica la divisibilità di n per tutti gli interi d tali che $2 \leq d \leq \sqrt{n}$.

  Se viene trovato un divisore d, restituisce d, indicando che n è composto; altrimenti, restituisce 0, indicando che n è primo.

- **Teorema Fondamentale dell'Aritmetica**

  Teorema 5.1.11: Ogni intero maggiore di 1 può essere fattorizzato in modo unico in numeri primi, ignorando l'ordine dei fattori.

- **Teorema 5.1.12 (Infinità dei Numeri Primi)**

  Il numero di numeri primi è infinito. Questo è dimostrato considerando il prodotto di tutti i numeri primi minori o uguali a un dato primo p, aggiungendo 1, e mostrando che il risultato è o divisibile per un nuovo primo maggiore di p, o è esso stesso primo.

- **Massimo Comune Divisore (MCD)**

  Il MCD di due interi m e n è il più grande intero che divide sia m che n.

Esempio: Il MCD di 30 e 105 è 15, poiché i loro divisori comuni sono $1, 3, 5e15$, e 15 è il più grande.

- **Trova il MCD tramite la Fattorizzazione Prima (Teorema 5.1.17)**

  Data la fattorizzazione prima di m e n, il MCD si trova prendendo l'esponente minimo di ciascun fattore primo comune a entrambe le fattorizzazioni.

- **Esempio di Calcolo del MCD**

  Per $m = 30 = 2 \times 3 \times 5$ e $n = 105 = 3 \times 5 \times 7$, il MCD è $3 \times 5 = 15$.

5.2 - Representation of integers and integers algorithms

Sistemi Numerici

1. **Sistema Decimale** (Base 10):
   - Usa 10 simboli ($0 - 9$). La posizione di ciascun simbolo è significativa: da destra a sinistra, il primo simbolo rappresenta il numero di unità (10^0), il secondo il numero di decine (10^1), il terzo il numero di centinaia (10^2), e così via.
2. **Sistema Binario** (Base 2):
   - Usa solo 0 e 1. Ogni simbolo rappresenta una potenza di 2. Ad esempio, $1011012 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4510$.
3. **Sistema Esadecimale** (Base 16):
   - Usa 16 simboli ($0 - 9, A - F$, dove $A = 10, B = 11, ..., F = 15$). Ogni simbolo rappresenta una potenza di 16. Ad esempio, $B4F_{16} = 11 \times 16^2 + 4 \times 16^1 + 15 \times 16^0 = 2895_{10}$.

Algoritmi di Conversione

1. **Conversione da Base b a Decimale** (Algoritmo 5.2.3):

- L'algoritmo somma i prodotti dei simboli del numero per le potenze di b. Il tempo di esecuzione è $O(n)$, dove n è la lunghezza del numero.

2. **Esempio di Conversione**:
   - Il numero binario `1011012` viene convertito in decimale come $32 + 8 + 4 + 1 = 45_{10}$.
3. **Conversione da Decimale a Binario**:
   - Questo processo si effettua tramite divisioni successive per 2, registrando i resti. Ogni divisione successiva fornisce i bit in ordine inverso. Ad esempio, per il numero 91, si ottiene la rappresentazione binaria 1011011_2.

Implicazioni per gli Algoritmi

L'algoritmo di determinazione della primalità (discusso in seguito) utilizza la rappresentazione binaria e il numero di bit necessari per rappresentare un numero n. Poiché il numero di bit cresce come $1 + lg(n)$, si deduce che alcuni algoritmi (come quello per determinare se un numero è primo) non sono polinomiali in relazione alla dimensione dell'input.

In sintesi, questa parte esplora i diversi sistemi numerici, la loro rappresentazione in bit e come effettuare le conversioni tra di essi, oltre a discutere la complessità di alcuni algoritmi in base alla lunghezza dell'input.

Esempio 5.2.6 - Decimale a Binario

Per convertire il numero decimale 130 in binario, si eseguono divisioni successive per 2, registrando i resti:

1. $130 \div 21$ resto $= 0$(bit delle 1)
2. $65 \div 2$ resto $= 1$ (bit delle 2)
3. $32 \div 2$ resto $= 0$ (bit delle 4)
4. $16 \div 2$ resto $= 0$ (bit delle 8)
5. $8 \div 2$ resto $= 0$ (bit delle 16)
6. $4 \div 2$ resto $= 0$(bit delle 32)

7. $2 \div 2$ resto $= 0$ (bit delle 64)
8. $1 \div 2$ resto $= 1$ (bit delle 128)

La rappresentazione binaria di 130 è quindi 10000010.

Algoritmo 5.2.7 - Conversione di un intero decimale in una base b

L'algoritmo che converte un numero decimale in una base arbitraria b funziona come segue:

```
dec_to_base_b(m, b, c, n):
    n = -1
    while m > 0:
        n = n + 1
        c[n] = m % b
        m = m // b
```

In questo algoritmo, si divide successivamente il numero m per b, registrando i resti (le cifre nella nuova base).

Esempio 5.2.8 - Decimale a Binario con Algoritmo

Per convertire 11 in binario utilizzando l'algoritmo:

1. Dividi 11 per 2, ottieni 5 (resto 1), imposta il primo bit.
2. Dividi 5 per 2, ottieni 2 (resto 1), imposta il secondo bit.
3. Dividi 2 per 2, ottieni 1 (resto 0), imposta il terzo bit.
4. Dividi 1 per 2, ottieni 0 (resto 1), imposta il quarto bit.

Il numero 11 in binario è quindi `1011`.

Esempio 5.2.9 - Decimale a Esadecimale

Per convertire il numero decimale 20385 in esadecimale, si eseguono divisioni per 16, registrando i resti:

1. $20385 \div 16$ resto = 1 (posto delle unità)

2. $1274 \div 16$ resto = 10 (posto delle 16)
3. $79 \div 16$ resto = 15 (posto delle 256)
4. $4 \div 16$ resto = 4 (posto delle 4096)

La rappresentazione esadecimale di 20385 è quindi $4FA1$.

Esempio 5.2.10 - Somma Binaria

Per sommare i numeri binari 10011011 e 1011011, si esegue l'addizione da destra a sinistra, con il riporto. L'addizione risulta essere:

```
  10011011
+ 01011011
-----------
  11110110
```

Esempio 5.2.13 - Somma Esadecimale

Per sommare $84F$ e $42EA$ in esadecimale, si sommano i numeri partendo dalla colonna di destra:

1. Somma di F (15 in decimale) e A (10 in decimale) = 25(in decimale) = 19 (in esadecimale), quindi scriviamo 9 e riportiamo 1.
2. Aggiungendo $1, 4$ e E (14 in decimale), otteniamo 13(in esadecimale).

Il risultato finale è $4B39$.

Algoritmo 5.2.12 - Aggiunta di Numeri Binari

Questo algoritmo somma due numeri binari bit per bit, considerando anche il riporto da un'operazione precedente. La somma dei bit viene effettuata come segue:

```
binary_addition(b, b', n, s):
    carry = 0
```

```
    for i = 0 to n:
        s[i] = (b[i] + b'[i] + carry) % 2
        carry = (b[i] + b'[i] + carry) // 2
```

Algoritmo 5.2.16 - Esponenziazione per Quadratura Ripetuta

Questo algoritmo calcola a^n (potenza di a elevata a n) utilizzando la tecnica della "quadratura ripetuta", che riduce il numero di moltiplicazioni necessarie:

```
_via_repeated_squaring(a, n):
    result = 1
    x = a
    while n > 0:
        if n % 2 == 1:
            result = result * x
        x = x * x
        n = n // 2
    return result
```

Questo metodo riduce il numero di moltiplicazioni rispetto al metodo tradizionale che utilizza $n - 1$ moltiplicazioni.

Teorema 5.2.17 - Proprietà del Modulo

Questo teorema dice che a^b mod z=$[(a \text{ mod } z)(b \text{ mod } z)]$ mod z. È utile per mantenere i numeri relativamente piccoli durante i calcoli di esponenti grandi, come mostrato nell'Esempio 5.2.18.

5.3 - The euclidean algorythm

Nella Sezione 5.1, abbiamo discusso alcuni metodi per calcolare il massimo comune divisore di due numeri interi che si sono rivelati inefficienti. L'algoritmo euclideo è un algoritmo antico, famoso ed efficiente per trovare il massimo comune divisore di due numeri interi.

L'algoritmo euclideo si basa sul fatto che se $r = a$ mod b, allora

$gcd(a, b) = gcd(b, r)$.

Prima di dimostrare questa formula, mostriamo come l'algoritmo euclideo la utilizzi per trovare il massimo comune divisore.

Esempio 5.3.1 Poiché 105 mod $30 = 15$, per la formula sopra,

$gcd(105, 30) = gcd(30, 15)$.

Poiché 30 mod $15 = 0$, per la stessa formula,

$gcd(30, 15) = gcd(15, 0)$.

Per ispezione, $gcd(15, 0) = 15$. Quindi,

$gcd(105, 30) = gcd(30, 15) = gcd(15, 0) = 15$.

Ora dimostriamo la formula sopra.

Teorema 5.3.2 Se a è un intero non negativo, b è un intero positivo, e $r = a$ mod b, allora

$gcd(a, b) = gcd(b, r)$.

Prova Per il teorema del quoziente e resto, esistono q e r tali che

$a = bq + r0 \leq r < b$.

Mostriamo che l'insieme dei divisori comuni di a e b è uguale all'insieme dei divisori comuni di b e r, provando così il teorema.

Sia c un divisore comune di a e b. Per il Teorema 5.1.3(c), c ∣ bq. Poiché $c \mid a$ e $c \mid bq$, per il Teorema 5.1.3(b), $c \mid a - bq (= r)$. Quindi, c è un divisore comune di b e r.

Viceversa, se ccc è un divisore comune di b e r, allora $c \mid bq$ e $c \mid bq + r (= a)$, quindi c è un divisore comune di a e b. Quindi l'insieme dei divisori comuni di a e b è uguale all'insieme dei divisori comuni di b e r. Pertanto, $gcd(a, b) = gcd(b, r)$.

Successivamente, enunciamo formalmente l'algoritmo euclideo come Algoritmo 5.3.3.

Algoritmo 5.3.3 Algoritmo Euclideo

Questo algoritmo trova il massimo comune divisore dei numeri interi non negativi a e b, dove non entrambi sono zero.

Input: a e b (interi non negativi, non entrambi zero)

Output: Massimo comune divisore di aaa e bbb

1. gcd(a, b) {
2. // rendi a il più grande
3. if (a < b)
4. swap(a, b)
5. while (b ≠ 0) {
6. r = a mod b
7. a = b
8. b = r
9. }
10. return a
11. }

5.3 - the RSA public-Key cryptosystem

La crittologia studia i sistemi per comunicazioni sicure. In un crittosistema, un messaggio viene cifrato dal mittente e decifrato dal destinatario, con l'obiettivo di proteggere il contenuto dagli utenti non autorizzati. I crittosistemi sono essenziali per governi, aziende e individui, ad esempio per proteggere transazioni finanziarie o dati sensibili.

Nei sistemi più semplici, mittente e destinatario condividono una chiave privata per trasformare i caratteri del messaggio. Tuttavia, questi metodi sono vulnerabili, poiché chi intercetta un messaggio potrebbe analizzare la frequenza dei caratteri o delle combinazioni.

Sistema RSA

L'RSA, inventato da Rivest, Shamir e Adleman, è un crittosistema a chiave pubblica: ogni utente rende pubblica una chiave di cifratura e mantiene privata una chiave di decifratura. Per inviare un messaggio, il mittente utilizza la chiave pubblica del destinatario per cifrare il messaggio. Il destinatario usa la propria chiave privata per decifrare il messaggio.

Funzionamento:

1. Si scelgono due numeri primi p e q e si calcola $z = p \cdot q$.
2. Si calcola $\phi = (p-1)(q-1)$.
3. Si sceglie un numero n tale che $gcd(n, \phi) = 1$, rendendo la coppia z, n pubblica.
4. Si calcola un numero s, tale che $n \cdot s$ mod $\phi = 1$, mantenendolo segreto.
5. Il mittente cifra il messaggio come$c = an$ mod z, dove a è il messaggio convertito in un numero.
6. Il destinatario decifra il messaggio calcolando cs mod

z, ottenendo il messaggio originale a.

Esempio:

Con $p = 23, q = 31, z = 713, \phi = 660$, e $n = 29$, la chiave privata è $s = 569$. Per cifrare $a = 572$, si calcola $c = 572_{29} mod\ 713 = 113$. Il destinatario decifra 113^{569} mod $713 = 572$.

Sicurezza dell'RSA:

La sicurezza si basa sulla difficoltà di fattorizzare grandi numeri z (ad esempio, prodotti di numeri primi con almeno 1024 bit). Attualmente, non esistono algoritmi efficienti per questa operazione o per il calcolo delle radici modulari. Tuttavia, metodi alternativi, come attacchi basati sul tempo di decifratura, possono rappresentare una minaccia. Per questo motivo, sono state adottate contromisure per garantire la sicurezza.

L'RSA è oggi ampiamente utilizzato per transazioni sicure su internet, come acquisti online e comunicazioni riservate.

Chapter 11 - Boolean algebras and Combinatorial circuits

11.1 - Combinatorial circuits

Un **circuito combinatorio** è un tipo di circuito digitale il cui output è determinato unicamente dalla combinazione attuale degli input, senza dipendere da stati precedenti. Non possiede memoria, a differenza dei circuiti sequenziali (trattati nel capitolo successivo).

Tipi di Porte Logiche

1. **AND**: Produce `1` solo se tutti gli input sono `1`. Simbolo: $x_1 \wedge x_2$.
2. **OR**: Produce `1` se almeno uno degli input è `1`. Simbolo: $x_1 \vee x_2$.
3. **NOT**: Inverte l'input (`1` diventa `0`, `0` diventa `1`). Simbolo: $\neg x$.

Tabelle Logiche

Le tabelle logiche descrivono i risultati degli output per tutte le possibili combinazioni degli input. Ad esempio:

- AND: $x_1 \wedge x_2$:
  - $0 \wedge 0 = 0$
  - $0 \wedge 1 = 0$
  - $1 \wedge 0 = 0$
  - $1 \wedge 1 = 1.$
- OR: $x_1 \vee x_2$:
  - $0 \vee 0 = 0$
  - $0 \vee 1 = 1$
  - $1 \vee 0 = 1$
  - $1 \vee 1 = 1.$
- NOT: $\neg x$:

- $\neg 0 = 1$
- $\neg 1 = 0$.

Circuiti e Tabelle Logiche

- I circuiti combinatori possono essere rappresentati graficamente o con **espressioni booleane** che utilizzano gli operatori $\wedge$ (AND), $\vee$ (OR), e $\neg$ (NOT).
- **Esempio**: Per il circuito con espressione $y = \neg((x_1 \wedge x_2) \vee x_3)$, si calcola l'output seguendo il flusso del circuito:
  - $x_1 \wedge x_2$: AND tra x1 e x2.
  - $(x_1 \wedge x_2) \vee x_3$: OR tra il risultato precedente e x3.
  - $\neg((x_1 \wedge x_2) \vee x_3)$: NOT dell'output.

Composizione di Circuiti

I circuiti combinatori possono essere interconnessi per creare circuiti più complessi. Ogni circuito è descritto da una propria espressione booleana, che può essere combinata con altre per rappresentare il comportamento dell'intero sistema.

Esempio Avanzato

Per l'espressione booleana $(x_1 \wedge (x_2 \vee x_3)) \vee x_2$:

1. Si costruisce il circuito che calcola $x_2 \vee x_3$.
2. Il risultato è *ANDato* con x1: $x_1 \wedge (x_2 \vee x_3)$.
3. Questo output è poi *ORato* con x2: $(x_1 \wedge (x_2 \vee x_3)) \vee x_2$.
4. Si genera la tabella logica corrispondente.

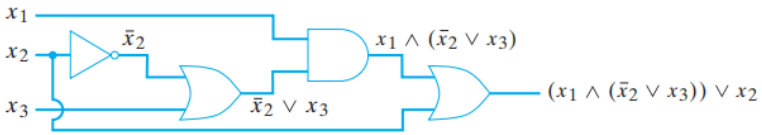


Figure 11.1.11 The combinatorial circuit corresponding to the Boolean expression $(x_1 \wedge (\bar{x}_2 \vee x_3)) \vee x_2$.

x_1	x_2	x_3	$(x_1 \wedge (\bar{x}_2 \vee x_3)) \vee x_2$
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	0

11.2 - Properties of combinatorial circuits

Definizione di base:

- Si utilizzano gli operatori binari $\wedge$ (AND), $\vee$ (OR) e l'operatore unario (NOT) sul set $Z_2 = 0, 1$, rappresentando le operazioni logiche fondamentali.

Teorema 11.2.1: Le operazioni $\wedge$, $\vee$ e soddisfano le seguenti proprietà:

1. **Leggi associative:**
   - $(a \vee b) \vee c = a \vee (b \vee c)$
   - $(a \wedge b) \wedge c = a \wedge (b \wedge c)$

     $\forall a, b, c \in Z_2.$
2. **Leggi commutative:**
   - $a \vee b = b \vee a.$
   - $a \wedge b = b \wedge a$
3. **Leggi distributive:**
   - $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c).$
   - $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c).$
4. **Leggi identità:**

- $a \vee 0 = a$
- $a \wedge 1 = a.$

5. **Leggi del complemento:**
   - $a \vee \neg a = 1$
   - $a \wedge \neg a = 0.$

Dimostrazione: Si verifica ogni proprietà testando tutti i possibili valori di $a, b, c \in Z_2$.

Equivalenza dei circuiti combinatori

1. **Circuiti equivalenti:** Due circuiti combinatori sono equivalenti se producono la stessa uscita per ogni insieme di ingressi.
2. **Relazione tra espressioni booleane e circuiti:** Due circuiti sono equivalenti se le loro espressioni booleane sono uguali (cioè forniscono lo stesso valore per ogni combinazione di ingressi).
3. **Ottimizzazione:** Circuiti equivalenti possono avere un diverso numero di porte logiche; è preferibile minimizzare il numero di porte per ridurre i costi.

Esempi

1. **Distributività:**
   - Si dimostra che $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ verificando tutte le combinazioni possibili di $a, b, cinZ_2$.
2. **Equivalenza tra espressioni:**
   - Si verifica che$\neg(x \vee y) = \neg x \wedge \neg y$ costruendo una tabella della verità.
3. **Equivalenza dei circuiti:**
   - Due circuiti diversi (ad esempio quelli nelle Figure 11.2.4 e 11.2.5) possono avere tabelle di verità identiche e quindi essere equivalenti.

11.3 - Boolean algebras

Definizione 11.3.1: Algebra di Boole

Un'algebra di Boole B è definita come un insieme S con:

- Due elementi distinti 0 e 1.
- Due operatori binari + e ·.
- Un operatore unario ¬.

Le seguenti leggi devono essere soddisfatte:

1. **Leggi associative**:
   - $(x + y) + z = x + (y + z)$
   - $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.
2. **Leggi commutative**:
   - $x + y = y + x$
   - $x \cdot y = y \cdot x$.
1. **Leggi distributive**:
   - $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
   - $x + (y \cdot z) = (x + y) \cdot (x + z)$
2. **Leggi di identità**:
   - $x + 0 = x$
   - $x \cdot 1 = x$
3. **Leggi dei complementi**:
   - $x + \neg x = 1$
   - $x \cdot \neg x = 0$

Un esempio è l'insieme {0,1} con gli operatori logici OR (+) e AND (·).

Proprietà derivate

- **Teorema 11.3.4**: Il complemento ¬x è unico.
- **Teorema 11.3.6**: Proprietà addizionali, tra cui:

- **Leggi idempotenti**:
  - $x + x = x$
  - $x \cdot x = x$
- **Leggi dei limiti**:.
  - $x + 1 = 1$
  - $x \cdot 0 = 0$
- **Leggi di assorbimento**:
  - $x + xy = x$
  - $x(x + y) = x$
- **Leggi di De Morgan**:
  - $(x + y)^{\neg} = \neg x \cdot \neg y$
  - $(x \cdot y)^{\neg} = \neg x + \neg y$

Esempi

1. **Insiemi**: Se S è l'insieme delle parti $P(U)$ di un insieme universale U, allora le operazioni unione (+), intersezione (·) e complemento (¬) formano un'algebra di Boole.
2. **Dualità**: Ogni legge di un'algebra di Boole ha il suo "dual", ottenuto scambiando 0 con 1, + con ·, e viceversa.

Dualità e Teoremi

- **Teorema 11.3.10**: Il duale di un teorema di un'algebra di Boole è anch'esso un teorema.
- Esempio: La dualità delle leggi idempotenti: x+x=x (sommatoria) e x·x=x (prodotto).

11.4 - Boolean functions and synthesis of circuits

Circuiti combinatori e tabelle di verità

Un circuito combinatorio esegue un'operazione specifica basata sui valori degli ingressi. Per progettare un circuito, si parte da una tabella di verità che definisce la relazione tra ingressi e uscite. Ad esempio, per la funzione XOR ($x_1 \oplus x_2$), la tabella specifica i valori di uscita per ogni combinazione di ingressi.

Definizione di funzione booleana

Una funzione booleana è una relazione $f(x_1, \ldots, x_n)$ rappresentabile con un'espressione booleana. Ad esempio, $f(x_1, x_2, x_3) = x_1 \wedge (x_2 \vee x_3)$ è una funzione booleana.

Costruzione di funzioni booleane

Data una tabella di verità, si può rappresentare la funzione come una combinazione di termini in cui l'uscita è 1. Ogni riga con uscita 1 è rappresentata da un termine che usa gli ingressi con gli operatori logici appropriati (AND e NOT). Questi termini sono poi uniti con l'operatore OR. Questa rappresentazione è chiamata **forma normale disgiuntiva (DNF)**.

Mintermini

Un minterm è una combinazione booleana in cui ogni variabile compare esattamente una volta (o nella forma normale o negata). Ogni minterm corrisponde a una riga della tabella di verità con uscita 1.

Teorema

Ogni funzione booleana può essere rappresentata come una somma logica (OR) di mintermini corrispondenti alle righe in cui la funzione vale 1.

Forma normale congiuntiva (CNF)

Esiste una rappresentazione duale della DNF chiamata forma normale congiuntiva, in cui la funzione è rappresentata come un prodotto logico (AND) di maxtermini. Un maxterm è una combinazione booleana in cui ogni variabile compare almeno una volta (nella forma normale o negata).

Esempio: XOR

La funzione XOR ($x_1 \oplus x_2$) può essere rappresentata come:

$$x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

Il circuito combinatorio corrispondente utilizza AND, OR e NOT.

11.5 - Applications

Circuiti combinatori e porte logiche

1. **Definizione di porta logica**: Una porta logica è una funzione da Z_2^n a Z_2 che può essere utilizzata per costruire circuiti combinatori. Esempi includono le porte **AND**, **OR** e **NOT**.
2. **Completezza funzionale**: Un insieme di porte è detto **funzionalmente completo** se può implementare qualsiasi funzione booleana. Ad esempio, le porte **AND**, **OR** e **NOT** insieme sono complete. Tuttavia, anche gli insiemi {**AND, NOT**} o {**OR, NOT**} sono completi.
3. **Porte NAND**: Una singola porta **NAND** (not AND) è sufficiente per costruire qualsiasi funzione booleana, rendendola un set funzionalmente completo.
4. **Circuiti minimizzati**:
   - **Minimizzazione**: I circuiti possono essere semplificati riducendo il numero di porte utilizzate, sfruttando leggi e proprietà dell'algebra booleana.
   - **Esempio**: Un circuito per $f(x, y, z) = xz \vee yz$ può essere semplificato da 9 a 3 porte.
5. **Circuiti utili**:
   - **Mezzo sommatore (Half Adder)**: Calcola la somma binaria di due bit (s) e il bit di riporto (c).
   - **Sommatore completo (Full Adder)**: Somma tre bit (due bit più un riporto) producendo un risultato a due bit (cs).
   - **Sommatore a 3 bit**: Utilizzando mezzi sommatore e sommatore completo, è possibile costruire un circuito che somma due numeri binari a 3 bit.
6. **Applicazioni pratiche**: I circuiti combinatori trovano utilizzo nei computer moderni per sommare numeri binari e in altre operazioni logiche. I progressi tecnologici hanno portato alla miniaturizzazione dei componenti (circuiti integrati), ma l'algebra booleana resta essenziale nella progettazione.

Chapter 12 - Automata Grammars and Languages

12.1 - Sequential cirquits and finite-state machines

Operazioni nei Circuiti Sequenziali

I circuiti sequenziali funzionano in intervalli di tempo discreti, dove lo stato del sistema cambia solo in momenti specifici $t = 0, 1, \dots$ Il comportamento dipende sia dallo stato attuale che dall'input.

- **Ritardo Unitario**: Un ritardo unitario accetta un bit x_t in ingresso al tempo t e restituisce $x_t - 1$, il bit ricevuto al tempo $t - 1$.

Sommatore Seriale

Un sommatore seriale accetta due numeri binari come input, uno per volta, e produce la loro somma. Esempio:

Per $x = 010$ e $y = 011$, il circuito calcola $z = 101$.

Macchina a Stati Finiti (FSM)

Un'FSM è un modello astratto con memoria interna e si definisce come M= (I,O,S,f,g,σ):

1. **Input** (I): Simboli in ingresso.
2. **Output** (O): Simboli in uscita.
3. **Stati** (S): Stati possibili del sistema.
4. **Funzione di Transizione** (f): Determina il prossimo stato.
5. **Funzione di Output** (g): Determina l'output.
6. **Stato Iniziale** (σ).

Esempio:

- Stati: σ_0, σ_1.
- Transizioni definite da una tabella o un diagramma (grafico con archi etichettati input/output).

Esempi di FSM

1. **Sommatore Seriale come FSM**: Accetta coppie di bit in ingresso. Gli stati (Carry e No Carry) determinano il comportamento. Il diagramma rappresenta tutte le possibili transizioni tra gli stati.
2. **Flip-Flop SR**: È un componente digitale base che memorizza 1 bit. È modellato come FSM con due stati ("S ultimo uguale a 1" e "R ultimo uguale a 1"). Il flip-flop ricorda quale dei due segnali, SSS o RRR, è stato impostato per ultimo

---

12.2 - Finite-state automata

Un automa a stati finiti è un tipo particolare di macchina a stati finiti, di interesse per la sua relazione con i linguaggi.

Definizione

Un automa a stati finiti A=(I,O,S,f,g,σ) è una macchina a stati finiti in cui:

- L'insieme dei simboli di output O è {0, 1},
- Lo stato corrente determina l'ultimo output. Gli stati in cui l'ultimo output è 1 sono detti **stati accettanti**.

Esempi

1. **Esempio 12.2.2:** Dato un'automa definito da una tabella di transizione, il diagramma degli stati mostra che gli stati accettanti sono σ_1 e σ_2.
2. **Esempio 12.2.3:** Un automa a stati finiti può essere rappresentato come una macchina a stati finiti con etichette sugli archi che indicano gli output.

Caratterizzazione alternativa

Un automa a stati finiti può essere definito come A=(I,S,f,A,σ), dove:

- I: insieme di simboli di input.
- S: insieme degli stati.
- f: funzione di transizione.
- A: insieme degli stati accettanti.

- σ: stato iniziale.

Esempi di accettazione

1. Una stringa α è accettata se esiste un percorso che termina in uno stato accettante.
    - **Esempio 12.2.6:** La stringa "abaa" è accettata dall'automa di un diagramma specifico.
    - **Esempio 12.2.7:** La stringa "abbabba" non è accettata perché termina in uno stato non accettante.

Progettazione di automi

1. **Esempio 12.2.8:** Progettare un automa che accetta solo stringhe senza "a". Usa due stati: uno iniziale e accettante (NA) e uno per "a trovato" (A).
2. **Esempio 12.2.9:** Progettare un automa che accetta stringhe con un numero dispari di "a". Usa due stati: uno per un numero pari (E) e uno per un numero dispari (O).

Algoritmo di accettazione

Un automa può essere tradotto in un algoritmo per decidere se una stringa è accettata:

- Inizializza lo stato iniziale.
- Per ogni simbolo della stringa, aggiorna lo stato.
- Se lo stato finale è accettante, restituisci "Accetta", altrimenti "Rifiuta".

Equivalenza tra automi

Due automi si dicono equivalenti se accettano esattamente le stesse stringhe. Questa relazione è un'**equivalenza** e gli automi equivalenti appartengono alla stessa classe di equivalenza.

Conclusione: Gli automi a stati finiti forniscono un modello matematico per riconoscere linguaggi e possono essere rappresentati con diagrammi, tabelle o algoritmi.

12.3 - Languages and grammar

Definizione di Linguaggio

- Il **Merriam-Webster's Collegiate Dictionary** definisce il linguaggio come:"Le parole, la loro pronuncia e i metodi di combinazione che una comunità usa e comprende".
- **Lingue naturali**: Complesse e difficili da caratterizzare completamente.
- **Lingue formali**: Più semplici e completamente specificabili; usate per modellare lingue naturali o comunicare con i computer.

12.3.1 Lingue Formali

Definizione 12.3.1

Sia A un insieme finito. Una **lingua formale** L su A è un sottoinsieme di A^*, l'insieme di tutte le stringhe su A.

Esempio 12.3.2

- $A = a, b$
- L: L'insieme di stringhe su A che contengono un numero dispari di a.
- Questa lingua è accettata dall'automa a stati finiti mostrato nella **Figura 12.2.7**.

12.3.2 Grammatiche

Definizione 12.3.3

Una **grammatica di struttura di frase** G è definita come:

1. N: Insieme finito di simboli non terminali.
2. T: Insieme finito di simboli terminali ($N \cap T = \emptyset$).
3. P: Insieme finito di produzioni:

$$P \subseteq [(N \cup T)^* - T^*] \times (N \cup T)^*$$

4. σ: Simbolo di partenza ($\sigma \in N$).

Si scrive $G = (N, T, P, \sigma)$.

Una produzione $(A, B) \in P(A, B)$ viene scritta come $A \to B$.

- A: Deve contenere almeno un simbolo non terminale.
- B: Può contenere qualsiasi combinazione di simboli terminali e non terminali.

Esempio 12.3.4

Grammatica $G = (N, T, P, \sigma)$:

- $N = \sigma, S$
- T={a, b}
- P={$\sigma \to b\sigma$,$\sigma \to aS$,$S \to bS$,$S \to b$}

12.3.3 Derivazione di Stringhe

Definizione 12.3.5

Data $G = (N, T, P, \sigma)$:

- **Derivazione diretta**:Se $\alpha \to \beta \in P$ e $x\alpha y \in (N \cup T)^*$, allora $x\beta y$ è direttamente derivabile da $x\alpha y$, scritto $x\alpha y \Rightarrow x\beta y$.
- **Derivazione generale**:Se $\alpha_1, \alpha_2, ..., \alpha_n \in (N \cup T)^*$, e α_{i+1} è direttamente derivabile da α_i, allora α_n è derivabile da α_1, scritto $\alpha_1 \Rightarrow \alpha_n$.
- **Linguaggio generato**:L'insieme $L(G)$ contiene tutte le stringhe su T derivabili da σ.

Esempio 12.3.6

Grammatica G come in **Esempio 12.3.4**:

1. La stringa abSbb è derivabile da aSbb, scritto:
   aSbb⇒abSbb
2. La stringa bbab è derivabile da σ, scritto:
   σ⇒bbab

Derivazione:
σ⇒bσ⇒bbσ⇒bbaS⇒bbab

12.3.4 Forma Normale di Backus-Naur (BNF)

Sintassi BNF

- Simboli non terminali: Racchiusi tra < e >.
- Produzione: Scritta come $S ::= T$.
- Produzioni multiple: Combinate con $|$, letto come "oppure".

Esempio 12.3.7

Grammatica per numeri interi:

- **Simboli non terminali**:
  <digit>,<integer>,<signed integer>,<unsigned integer>
- **Simboli terminali**:
  {0,1,2,...,9,+,−}
- **Produzioni**:

$$< digit >::= 0 \mid 1 \mid ... \mid 9$$

$$< integer >::=< signed\ integer > \mid < unsigned\ integer >$$

$$< signed\ integer >::= + < unsigned\ integer > \mid - < unsigned\ integer >$$

$$< unsigned\ integer >::=< digit > \mid < digit >< unsigned\ integer >$$

12.3.5 Classificazione delle Grammatiche

Definizione 12.3.8

Classificazione in base alle produzioni:

1. **Grammatica sensibile al contesto (Tipo 1)**:

   $$\alpha A\beta \to \alpha\delta\beta$$

Con $\alpha, \beta \in (N \cup T)^*, A \in N, \delta \neq \lambda$.

1. **Grammatica libera dal contesto (Tipo 2)**:

   $A \to \delta$
   Con
   $A \in N, \delta \in (N \cup T)^*$.

2. **Grammatica regolare (Tipo 3)**:

   $A \to a \mid aB \mid \lambda$
   Con
   $A, B \in N, a \in T$.

12.4 - Nondeterministic finite-state automata

Questa sezione illustra la relazione tra automi a stati finiti e grammatiche regolari, dimostrando che entrambi descrivono linguaggi regolari. Viene anche introdotto il concetto di automa a stati finiti non deterministico (NFA).

Conversione da automa a grammatica regolare

Un automa a stati finiti può essere trasformato in una grammatica regolare:

1. **Simboli terminali**: gli input dell'automa.
2. **Simboli non terminali**: gli stati dell'automa.
3. **Produzioni**: ogni arco dell'automa diventa una regola della grammatica. Ad esempio, un arco da S a S′ con etichetta x diventa $S \to xS\prime$.
4. **Produzioni speciali**: se uno stato è accettante, si aggiunge una regola che lo permette di terminare con la stringa vuota λ.

Esempio:

- L'automa di figura 12.2.7 genera la grammatica:

  $E \to bE, E \to aO, O \to aE, O \to bO, O \to \lambda$
  Questa grammatica genera lo stesso linguaggio accettato dall'automa.

Automata a stati finiti non deterministici (NFA)

Un NFA differisce da un automa deterministico (DFA) per il fatto che, per uno stato e un input specifico:

- Può esistere più di uno stato successivo (non determinismo).
- Non è richiesto che ogni input abbia un arco associato.

Definizione di un NFA:

1. **Simboli di input** (I): insieme finito di simboli.
2. **Stati** (S): insieme finito di stati.
3. **Funzione di transizione** (f): associa ogni coppia stato-simbolo a un sottoinsieme di stati (P(S)).
4. **Stati accettanti** (A): sottoinsieme degli stati.
5. **Stato iniziale** (σ).

Esempio:

- Consideriamo la grammatica $\sigma \to b\sigma, \sigma \to aC, C \to bC, C \to b$. Questa genera un NFA in cui:
  - Lo stato iniziale è σ.
  - C ha transizioni non deterministiche sull'input b (può rimanere in C o andare in F, uno stato accettante).

Accettazione di stringhe in un NFA

Una stringa è accettata da un NFA se:

1. Esiste almeno un percorso nello schema degli stati che rappresenta la stringa.
2. Il percorso termina in uno stato accettante.

Esempi:

- La stringa bbabb è accettata dall'NFA della figura 12.4.2, poiché esiste almeno un percorso che termina in uno stato accettante.
- La stringa abba non è accettata dall'NFA della figura 12.4.3, poiché nessun percorso termina in uno stato accettante.

Equivalenza tra NFA e grammatiche regolari

Un NFA può essere costruito a partire da una grammatica regolare seguendo un procedimento simile a quello descritto sopra. La relazione è bidirezionale: ogni NFA può essere trasformato in una grammatica regolare e viceversa.

Confronto tra DFA e NFA

Anche se sembra che un NFA sia più generale di un DFA, si può sempre convertire un NFA in un DFA equivalente. Questo sarà trattato nella prossima sezione.

12.5 - Relationship between languages and automa

La sezione 12.5 esplora le relazioni tra grammatiche regolari e automi a stati finiti (AF), dimostrando che ogni grammatica regolare può essere convertita in un automa a stati finiti deterministico equivalente (Teorema 12.5.4). Questo risultato si basa sulla conversione da automi a stati finiti non deterministici (NFA) a deterministici (DFA).

Esempi

- **Esempio 12.5.1**: Conversione di un NFA in un DFA equivalente. Vengono costruiti gli stati del DFA come sottoinsiemi degli stati del NFA e definite le transizioni basate sulle transizioni originali. Dopo aver eliminato gli stati irraggiungibili, si ottiene un DFA semplificato (Figura 12.5.2).
- **Esempio 12.5.2**: Applicazione del metodo a un altro NFA (Figura 12.5.3).

Dimostrazione (Teorema 12.5.3)

Si dimostra formalmente che un DFA costruito dai sottoinsiemi degli stati di un NFA accetta esattamente gli stessi linguaggi. La dimostrazione procede mostrando:

1. Se una stringa è accettata dal NFA, allora è accettata anche dal DFA equivalente.
2. Viceversa, ogni stringa accettata dal DFA corrisponde a una stringa accettata dal NFA.

Conclusione (Teorema 12.5.4)

Un linguaggio L è regolare se e solo se esiste un automa a stati finiti che accetta esattamente le stringhe di L.

Applicazioni

- **Esempio 12.5.5**: Dato un linguaggio generato da una grammatica regolare, si costruisce un DFA equivalente.
- **Esempio 12.5.6**: Dimostrazione che il linguaggio $L = a^n b^n \mid n \geq 1$ non è regolare, sfruttando il principio del "pumping lemma".
- **Esempio 12.5.7**: Costruzione di un automa che accetta il linguaggio inverso LR di un automa dato. Si invertono le frecce delle transizioni, scambiando stato iniziale e accettazione.
- **Esempio 12.5.8**: Estensione del metodo per L^R ad automi con più stati accettanti. Si introduce uno stato accettante unico e si applica la procedura.