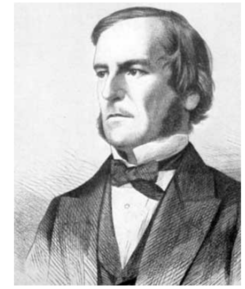
POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Algebra Booleana e Funzioni Logiche

Paolo Camurati

L’Algebra di Boole



- È stata introdotta nel 1854 da George Boole per analizzare algebricamente problemi di calcolo proposizionale (al fine di studiare le leggi del pensiero)
- È stata poi utilizzata come fondamento per la logica formale che costituisce la base per il ragionamento scientifico
- È un modello matematico usato per esprimere condizioni logiche e per la sintesi e l’analisi di sistemi digitali
- L’Algebra di Boole è un modello astratto: esistono tante Algebre di Boole!

Le Algebre di Boole

Un’Algebra di Boole è una struttura algebrica composta da:

- un insieme di elementi *B* che include almeno 0 e 1
- due operazioni binarie (= che operano su 2 operandi) $\{+, \cdot\}$
- un’operazione unaria (= che opera su 1 operando) $\{'\}$

che soddisfano gli assiomi di Huntingdon (1904):

1. *B* contiene almeno due elementi differenti a e b con $a \neq b$
2. *Chiusura:* $\forall$ a,b $\in$ B
   - (i) $a + b \in B$
   - (ii) $a \cdot b \in B$
   - *(iii)* $a' \in B$

3. *Commutatività*: $\forall$ a,b $\in$ B
    - (i) $a + b = b + a$
    - (ii) $a \cdot b = b \cdot a$
4. *Identità*: esistono 2 valori 0, 1 $\in B$ tali che
    - (i) $a + 0 = a$
    - (ii) $a \cdot 1 = a$
5. *Distributività*:
    - (i) $a + (b \cdot c) = (a + b) \cdot (a + c)$
    - (ii) $a \cdot (b + c) = a \cdot b + a \cdot c$
6. *Complemento*:
    - (i) $a + a' = 1$
    - (ii) $a \cdot a' = 0$

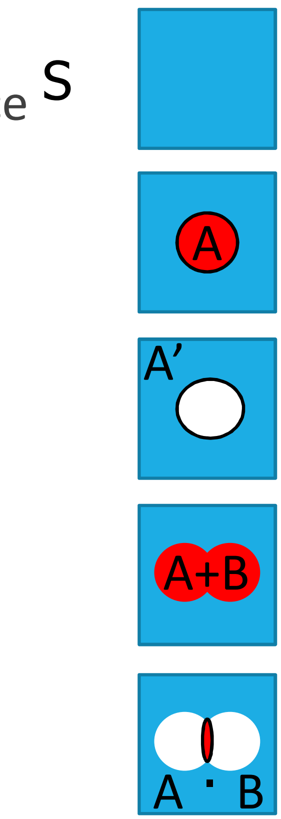
Esempi di Algebre Booleane

Specificando gli elementi di B e le operazioni si definisce un'interpretazione e si ottiene un modello di algebra.

Algebra degli insiemi:

Sia dato un insieme universo S. L'insieme delle parti di S (powerset) è l'insieme dei sottoinsiemi di S, inclusi S stesso e l'insieme vuoto ∅.

- B è l'insieme delle parti di S
- ' è l'operazione di complemento rispetto all'insieme universo S
- + è l'operazione di unione insiemistica
- · è l'operazione di intersezione insiemistica



Algebra della commutazione (**Switching Algebra**)
Claude Shannon (1938) ha introdotto l’Algebra della Commutazione per descrivere le proprietà dei circuiti elettrici con interruttori.

- $B = \{0, 1\}$
- operazioni binarie: OR (+), AND (·)
- operazione unaria: NOT (‘ o ¯ o ~ o sopralineatura ‾).

L’Algebra della commutazione (nel seguito identificata con abuso di notazione come Algebra Booleana):
- serve per esprimere le condizioni logiche nei linguaggi di programmazione
- è la base per progettare e analizzare sistemi digitali.

Interruttori e Algebra della Commutazione

Ingressi:

- 1 corrisponde a interruttore chiuso
- 0 corrisponde a interruttore aperto

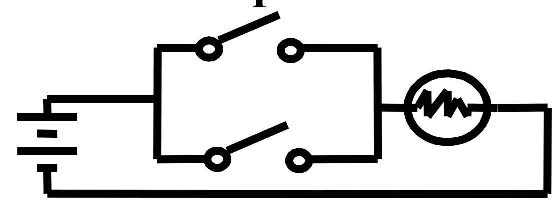
Uscita:

- 1 corrisponde a lampadina accesa
- 0 corrisponde a lampadina spenta

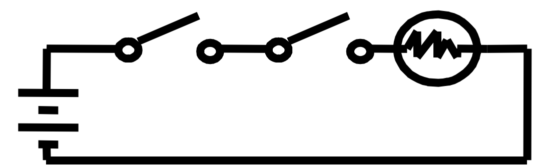
Il NOT usa un interruttore tale che:

- 1 è l'interruttore aperto
- 0 è l'interruttore chiuso

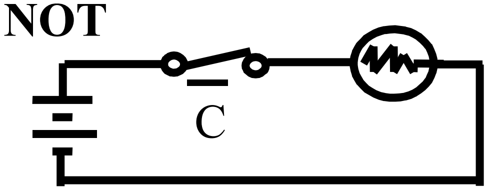
Interruttori in parallelo => OR



Interruttori in serie => AND



Interruttore normalmente chiuso => NOT



Variabili Booleane

- I due valori (binari) di B hanno vari nomi:
  - True/False
  - On/Off
  - Yes/No
  - 1/0
- Usiamo 1 e 0
- Variabile Booleana: variabile che assume valori in B
- Esempi di identificatori di variabili: A, B, x, Y, y, z

Esempi di notazione

Date le variabili binarie A, B, x, Y, y, z:

- $Y = A \cdot B$ si legge come "Y è uguale a A AND B."
- $z = x + y$ si legge come "z è uguale a x OR y."
- $X = A'$ si legge come "X è uguale a NOT A."

Nota: l'affermazione:

- 1 + 1 = 2 (che si legge "1 più 1 è uguale a 2") vale quando + è l'operatore della somma aritmetica
- 1 + 1 = 1 (che si legge "1 or 1 è uguale a 1") vale quando + è l'operatore OR dell'Algebra Booleana

Teoremi fondamentali

Idempotenza:	$a + a = a$	$a \cdot a = a$
Elemento nullo:	$a + 1 = 1$	$a \cdot 0 = 0$
Assorbimento:	$a + (a \cdot b) = a$	$a \cdot (a + b) = a$
	$a \cdot (a' + b) = a \cdot b$	$a + (a' \cdot b) = a + b$
Involuzione:	$(a')' = a$	
Associatività:	$a + (b + c) = (a + b) + c$	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
Legge di De Morgan:	$(a + b)' = a' \cdot b'$	$(a \cdot b)' = a' + b'$
Combinazione:	$(a \cdot b) + (a \cdot b') = a$	$(a + b) \cdot (a + b') = a$
Consenso:	$(a \cdot b) + (a' \cdot c) + (b \cdot c) = (a \cdot b) + (a' \cdot c)$	
	$(a + b) \cdot (a' + c) \cdot (b + c) = (a + b) \cdot (a' + c)$	
Fattorizzazione:	$(a + b) \cdot (a' + c) = (a \cdot c) + (a' \cdot b)$	
	$(a \cdot b) + (a' \cdot c) = (a + c) \cdot (a' + b)$	

Proprietà

- Se non c'è ambiguità si può omettere il simbolo ·
- Si usa il simbolo ' per NOT
- Il duale di un'espressione algebrica si ottiene scambiando + e · e scambiando 0 e 1.
- I teoremi compaiono in coppie duali. Quando in una riga c'è un solo teorema, esso si dice auto-duale, cioè espressione duale = espressione originale.
- I teoremi fondamentali sono dimostrabili a partire dai postulati di Huntingdon

Dimostrazioni

Teorema dell'assorbimento: $a + (a' \cdot b) = a + b$

- Induzione perfetta (tutti i casi)

a	a'	b	a + a'b	a + b
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1

- Usando i postulati e teoremi già dimostrati

$a + a'b = (a + a') \cdot (a + b)$ distribuitività

$(a + a') \cdot (a + b) = 1 \cdot (a + b)$ complemento

$1 \cdot (a + b) = a + b$ identità

Teorema del consenso: ab + a’c + bc = ab + a’c

- Usando i postulati e teoremi già dimostrati

ab + a’c + bc	= ab + a’c + 1·(bc)	identità
ab + a’c + 1·(bc)	= ab + a’c + (a + a’)·(bc)	complemento
ab + a’c + (a + a’)·(bc)	= ab + a’c + abc + a’bc	distributività
ab + a’c + abc + a’bc	= ab + abc + a’c + a’bc	commutatività
ab + abc + a’c + a’bc	= ab + a’c	assorbimento

Definizione degli operatori Booleani

AND	OR	NOT
$0 \cdot 0 = 0$	$0 + 0 = 0$	$0' = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$1' = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	

Espressioni Booleane

Le espressioni Booleane sono tutte e solo quelle formate mediante l'uso di:

- costanti 0, 1 $\in$ B
- variabili Booleane x1, x2, ..., xn
- operatori AND, OR, NOT

applicando le seguenti regole:

- una costante è un'espressione
- una variabile è un'espressione
- se *g* e *h* sono espressioni Booleane, allora lo sono anche $g + h$, $g \cdot h$ e g'

Precedenze degli operatori Booleani

L'ordine di precedenza degli operatori Booleani è:

1. parentesi
2. NOT
3. AND
4. OR

Conseguenza: le parentesi appaiono attorno alle espressioni OR

Esempio: $A(B + C)(C + D')$

Funzioni Booleane

Dato B={0,1} si dice funzione logica o Booleana f di n variabili Booleane x1, x2, ..., xn , la funzione:

$$f(x1, x2, ..., xn): B^n \rightarrow B$$

che a ogni possibile assegnazione di valori Booleani alle variabili x1, x2, ..., xn faccia corrispondere un valore *0* o *1*.

B^n è il prodotto cartesiano $\underbrace{B \times B \times .... B}_{\text{n volte}}$.

Quante sono le possibili funzioni di n variabili Booleane?

- n variabili possono assumere $k = 2^n$ possibili configurazioni, sono le disposizioni ripetute di 2 elementi presi a n a n
- per ogni configurazione la funzione assume valore 0 oppure 1 (m=2)
- il numero di funzioni n variabili Booleane è pari al numero di disposizioni ripetute di m elementi presi a k a k e vale

$$D'_{m,k} = m^k = 2^{2^n}$$

Con n=1 le funzioni possibili sono 4:

- $f(x) = 0$ uscita costante 0
- $f(x) = 1$ uscita costante 1
- $f(x) = x$ trasferimento di ingresso su uscita
- $f(x) = x'$ trasferimento di ingresso negato su uscita

Con n= 2 le funzioni possibili sono 16:

- $f(x,y) = 0$ costante 0
- $f(x,y) = 1$ costante 1
- $f(x,y) = x$ trasferimento
- $f(x,y) = y$ trasferimento
- $f(x,y) = x'$ trasferimento e NOT
- $f(x,y) = y'$ trasferimento e NOT
- $f(x,y) = xy$ AND
- $f(x,y) = (xy)'$ **NAND**
- $f(x,y) = x+y$ OR
- $f(x,y) = (x+y)'$ **NOR**
- $f(x,y) = x'+y$ implicazione $x \rightarrow y$
- $f(x,y) = x+y'$ implicazione $y \rightarrow x$
- $f(x,y) = xy'$ inibizione di y
- $f(x,y) = x'y$ inibizione di x
- $f(x,y) = x \oplus y$ **EXOR**
- $f(x,y) = (x \oplus y)'$ **EXNOR**

NAND, NOR, EXOR, EXNOR: funzioni Booleane non elementari.

x	y	0	1	x	y	x'	y'	xy	x↑y
0	0	0	1	0	0	1	1	0	1
0	1	0	1	0	1	1	0	0	1
1	0	0	1	1	0	0	1	0	1
1	1	0	1	1	1	0	0	1	0

x	y	x+y	x↓y	x → y	y → x	xy'	x'y	x⊕y	(x⊕y)'
0	0	0	1	1	1	0	0	0	1
0	1	1	0	1	0	0	1	1	0
1	0	1	0	0	1	1	0	1	0
1	1	1	0	1	1	0	0	0	1

Rappresentazione delle funzioni Booleane

Formalismi di rappresentazione:

- espressioni
- tabelle di verità
- diagrammi circuitali.

Un formalismo di rappresentazione si dice *canonico* se, date due funzioni arbitrarie f e g queste sono uguali se e solo se è uguale la loro rappresentazione.

Le tabelle di verità sono rappresentazioni canoniche, non così le espressioni e i diagrammi circuitali.

Tabelle di verità

La *tabella di verità* di una funzione Booleana f è elenco tabulare dei valori di f per tutte le possibili combinazioni di valore dei suoi argomenti. Se la funzione f è di n variabili, la tabella avrà 2^n righe e n +1 colonne.

AND		
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR		
x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
x	x’
0	1
1	0

NAND e NOR

NAND e NOR sono importanti per la realizzazione di circuiti digitali mediante porte logiche. Da un punto di vista tecnologico è più facile realizzare porte NAND e NOR che AND, OR e NOT

Per NAND e NOR non vale l'associatività.

NAND		
x	y	(xy)'
0	0	1
0	1	1
1	0	1
1	1	0

NOR		
x	y	(x+y)'
0	0	1
0	1	0
1	0	0
1	1	0

Simboli che si possono trovare in letteratura: NAND ↑, NOR ↓

EXOR e EXNOR

EXOR è l'operatore differenza $x \neq y = xy' + x'y$

EXNOR è l'operatore uguaglianza $x \equiv y = xy + x'y'$

EXOR		
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

EXNOR		
x	y	$(x \oplus y)'$
0	0	1
0	1	0
1	0	0
1	1	1

Simboli che si possono trovare in letteratura:
EXOR $\oplus$, EXNOR $\equiv$.

Implicazione

Esprime condizioni del tipo

se c'è il sole, usciamo $x \rightarrow y$ x implica y

Analisi dei casi:

- x=1 e y=1: $x \rightarrow y$ è vera (vale 1) INTUITIVO!
- x=1 e y=0: $x \rightarrow y$ è falsa (vale 0) INTUITIVO!
- x=0 e y=0 o y=1: $x \rightarrow y$ è vera (vale 1) CONTROINTUITIVO!

Se l'ipotesi è falsa , posso dire quello che voglio sulla tesi!

Espressione Booleana:

$x \rightarrow y = xy + x'$ assorbimento

$= x' + y$

L'implicazione non esiste come porta logica elementare.

Insiemi funzionalmente completi

Un insieme di operatori è funzionalmente completo se è ogni funzione Booleana può essere descritta da un'espressione che usa solo gli operatori di quell'insieme.

Insiemi funzionalmente completi:

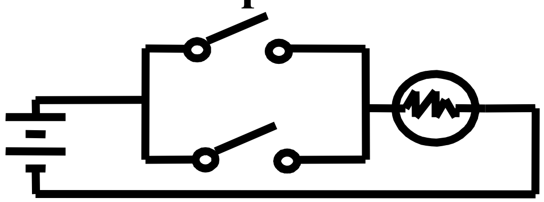
- AND, OR, NOT
- AND, NOT: infatti $x + y = (x'y')'$ (De Morgan)
- OR, NOT: infatti $xy = (x'+y')'$ (De Morgan)
- NAND: infatti $x' = (x \cdot x)' = x \uparrow x$, $xy = ((x \cdot y)')' = (x \uparrow y)' = (x \uparrow y) \uparrow (x \uparrow y)$
- NOR: infatti $x' = (x+x)' = x \downarrow x$, $x+y = ((x+y)')' = (x \downarrow y)' = (x \downarrow y) \downarrow (x \downarrow y)$

La completezza funzionale di NAND e NOR ha importanti conseguenze sulla realizzazione dei circuiti digitali.

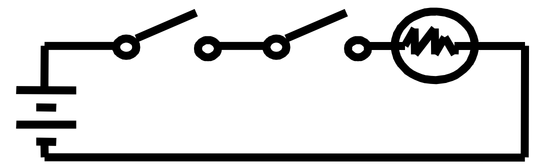
Le Porte Logiche

Gli operatori dell'Algebra Booleana possono essere realizzati fisicamente mediante circuiti elettronici che operano come interruttori opportunamente connessi.

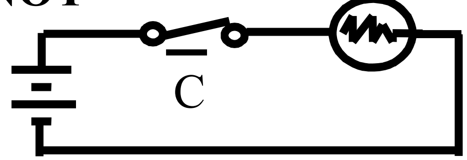
Interruttori in parallelo => OR



Interruttori in serie => AND



Interruttore normalmente chiuso => NOT



Nei primi computer gli interruttori erano azionati da campi magnetici prodotti in spire chiamate *relay*. Gli interruttori aprivano/chiudevano il flusso di corrente.

In seguito i *vacuum tubes* che aprivano/chiudevano il flusso di corrente elettronicamente hanno sostituito i relay.

Oggi si usano i *transistor* come interruttori elettronici per aprire/chiudere i flussi di corrente.

Le porte logiche (*logic gates*) sono dispositivi elettronici che realizzano fisicamente gli operatori Booleani AND, OR, NOT, NAND, NOR, EXOR, EXNOR.

Simboli delle Porte Logiche

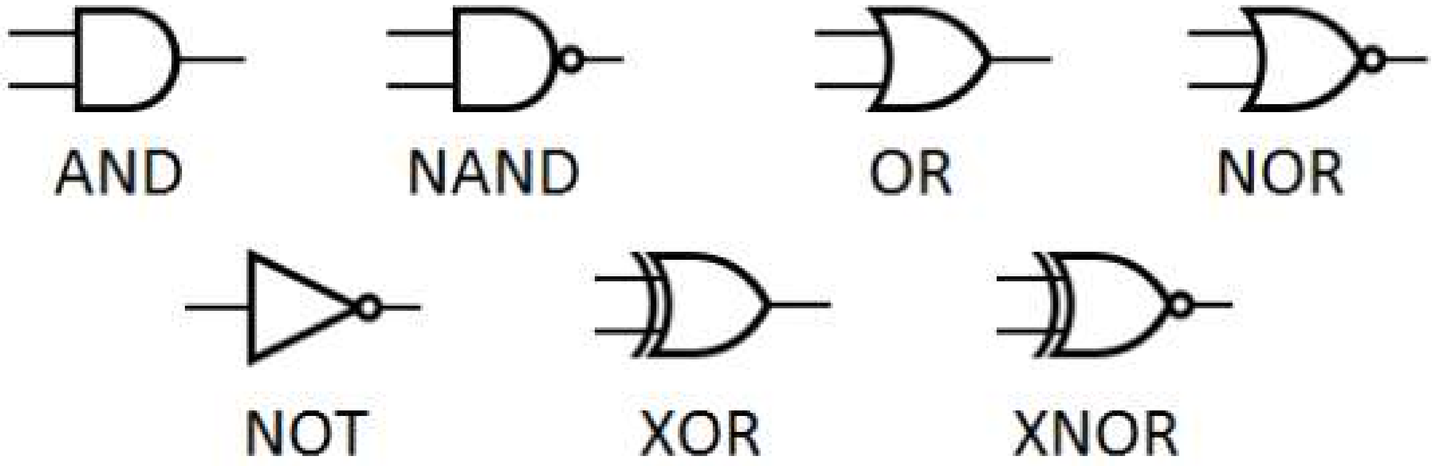


Diagramma logico

- Un diagramma logico (logic diagram) è una interconnessione di porte logiche mediante fili.
- Espressioni Booleane, tabelle di verità e diagrammi logici descrivono la stessa funzione!
- Le tabelle di verità sono uniche, non così le espressioni e i diagrammi logici. Questo permette flessibilità nell'implementazione delle funzioni.

Esempio

Funzione Booleana

f(x, y, z) = x + y’z

Diagramma logico

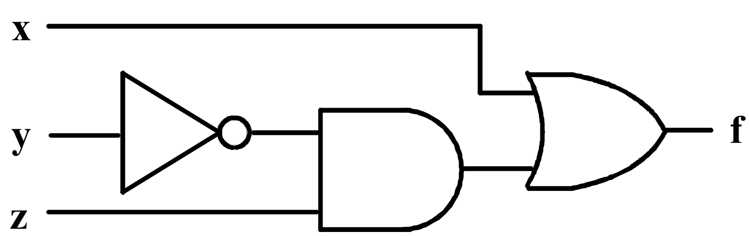


Tabella di verità

f			
x	y	z	x+y’z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Le Reti Logiche

Dato un insieme di simboli in ingresso codificato su n bit ($n \geq 0$) ed un insieme di simboli in uscita codificato su m bit ($m>0$), una rete logica è un sistma di eleborazione hardware che accetta una sequenza di simboli in ingresso e la trasforma in una sequenza di simboli in uscita.

La rete si dice combinatoria se il simbolo in uscita dipende unicamente dal simbolo in ingresso a quello stesso tempo. La rete dunque non ha memoria.

La rete si dice sequenziale se il simbolo in uscita dipende sia dal simbolo in ingresso al momento, sia da quelli precedentemente arrivati. La rete ha memoria. Lo stato memorizza quello che è rilevante della sequenza di ingresso ai fini dell'evoluzione nel tempo della rete. Il numero di stati nelle reti sequenziali è finito.

Rete combinatoria: il sommatore

Nel caso si vogliano sommare 2 numeri su n bit, quale sarebbe la dimensione della tabella di verità? ***$2^{(n+n)}$***

- *n= 2:* $2^4 = 16$ righe
- *n= 4:* $2^8 = 256$ righe
- *n= 8:* $2^{16} = 65536$ righe
- *n= 16:* $2^{32} = 4$ miliardi di righe.

Conclusione: serve un metodo alternativo, che imiti il procedimento manuale della somma:

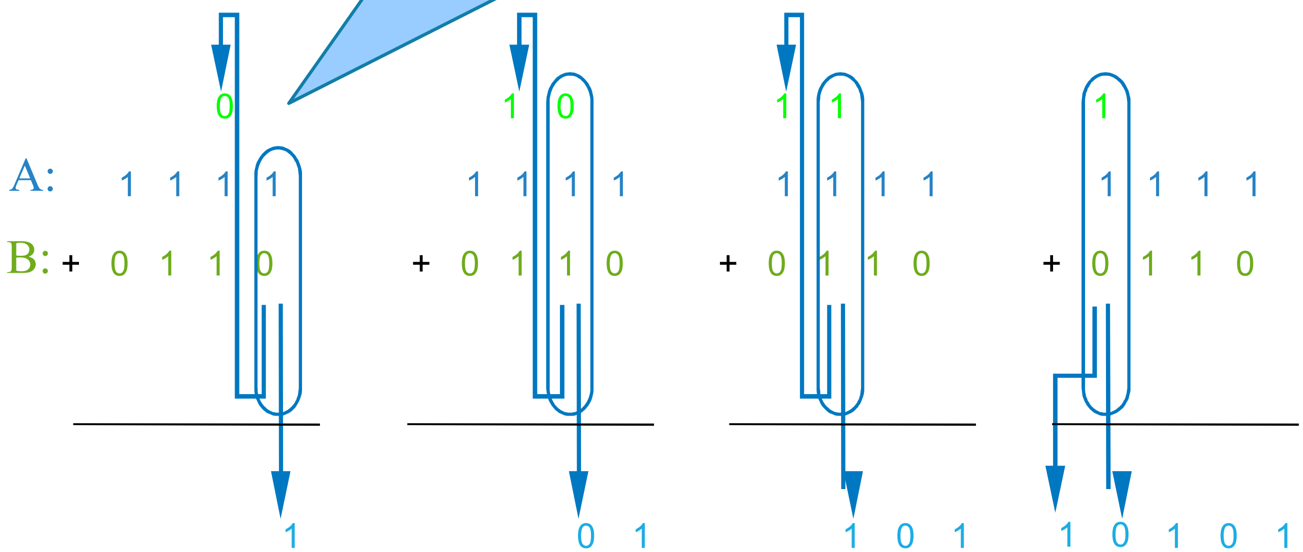
- mettere i numeri in colonna (bit dello stesso peso)
- procedere colonna per colonna da quella più a destra
- calcolare la somma della colonna e il riporto che viene propagato alla colonna successiva

Informatica Unità T1 Somma binaria

- Regole base:

0	+	0	=	0	
0	+	1	=	1	
1	+	0	=	1	
1	+	1	=	0	(carry = 1)

il riporto in ingresso è 0 per definizione nella colonna più a destra



Per sommare le colonne si progettano mediante la tabella di verità i seguenti blocchi funzionali:

- *Half-Adder:* ingressi X e Y su 1 bit, uscite riporto Cout (carry out) e somma S su 1 bit (per la colonna più a destra)
- *Full-Adder:* ingressi X, Y e Cin (carry in) su 1 bit, uscite riporto Cout e somma S su 1 bit (per le alter colonne)

e li si compone strutturalmente per realizzare un

- *Ripple Carry Adder*: blocco funzionale per la somma di una coppia di interi X e Y su n bit con Cin su 1 bit.

Half-adder

Operazioni:

X	0	0	1	1
+	+	+	+	+
Y	0	1	0	1
Cout S	0 0	0 1	0 1	1 0

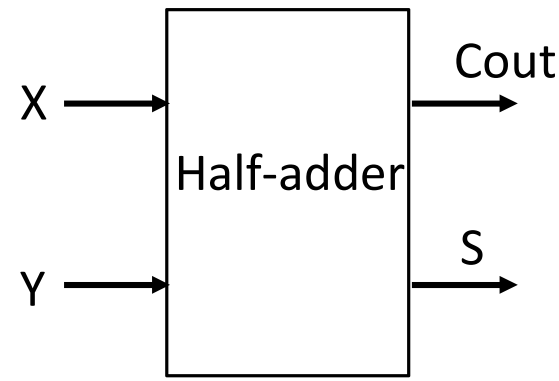


Tabella di verità:

Half-adder			
X	Y	Cout	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Funzione Booleana

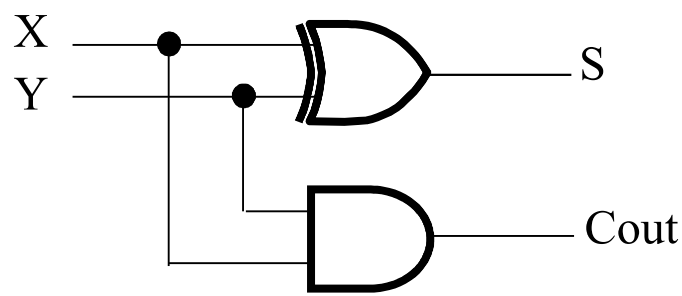
S vale 1 se e solo se (X = 0 AND Y = 1) OR (X = 1 AND Y = 0)

$$S = XY' + X'Y = X \oplus Y$$

Cout vale 1 se e solo se X = 1 AND Y = 1

$$Cout = XY$$

Diagramma logico



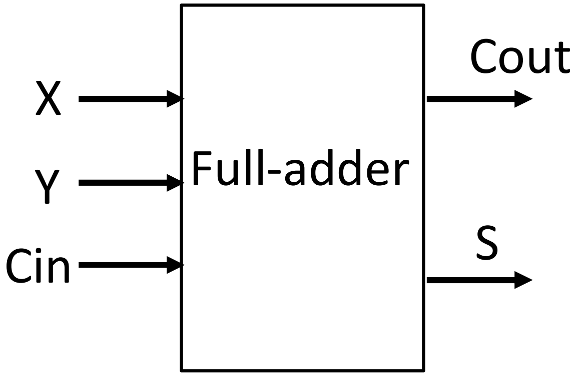
Full-adder

Operazioni:

Cin	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
+	+	+	+	+	+	+	+	+
X	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
+	+	+	+	+	+	+	+	+
Y	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
Cout S	**0 0**	**0 1**	**0 1**	**1 0**	**0 1**	**1 0**	**1 0**	**1 1**

Tabella di verità:

Full-adder				
X	Y	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Funzione Booleana

S vale 1 se e solo se (Cin = 1 AND X = 0 AND Y = 0) OR
(Cin = 0 AND X = 0 AND Y = 1) OR
(Cin = 0 AND X = 1 AND Y = 0) OR
(Cin = 1 AND X = 1 AND Y = 1)

$$S = CinX'Y' + Cin'X'Y + Cin'XY' + CinXY$$

L'espressione può essere manipolata mediante le regole dell'Algebra Booleana:

$S = CinX'Y' + Cin'X'Y + Cin'XY' + CinXY$	distributività
$= Cin(XY + X'Y') + Cin'(X'Y + XY')$	definizione di EXOR/EXNOR
$= Cin(X \oplus Y)' + Cin'(X \oplus Y)$	definizione di EXOR
$= Cin \oplus (X \oplus Y)$	

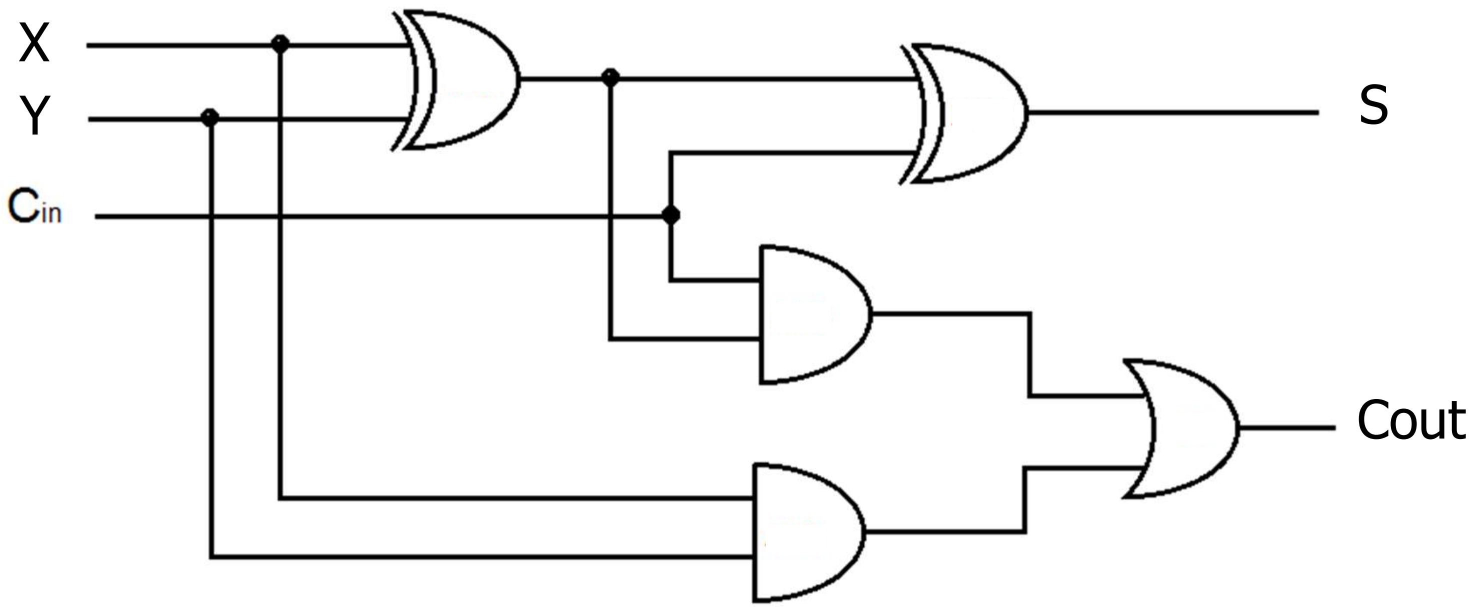
Cout vale 1 se e solo se (Cin = 1 AND X = 0 AND Y = 1) OR
(Cin = 1 AND X = 1 AND Y = 0) OR
(Cin = 1 AND X = 1 AND Y = 1) OR
(Cin = 0 AND X = 1 AND Y = 1) OR

$$Cout = CinX'Y + CinXY' + CinXY + Cin'XY$$

L'espressione può essere manipolata mediante le regole dell'Algebra Booleana:

$$
\begin{aligned}
Cout &= CinX'Y + CinXY' + CinXY + Cin'XY \\
&= Cin(X'Y + XY') + XY(Cin + Cin') && \text{distributività} \\
&= Cin(X \oplus Y) + XY(1) && \text{EXOR/complemento} \\
&= Cin(X \oplus Y) + XY && \text{identità}
\end{aligned}
$$

Diagramma logico



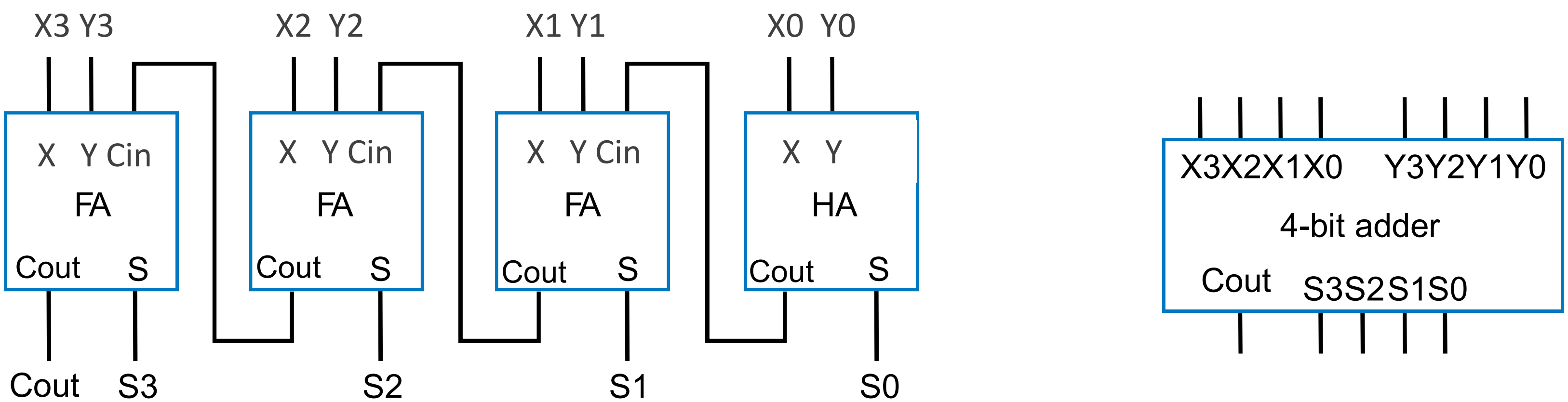
Sommatori binari

- Per sommare operandi su più bit, “raggruppiamo” assieme i segnali logici in vettori e usiamo blocchi funzionali che lavorano su vettori
- Esempio: <u>4-bit ripple carry adder:</u> somma i vettori X(3:0) e Y(3:0) per ottenere un vettore somma S(3:0)
- Il carry out dalla cella i diventa il carry in della cella i + 1

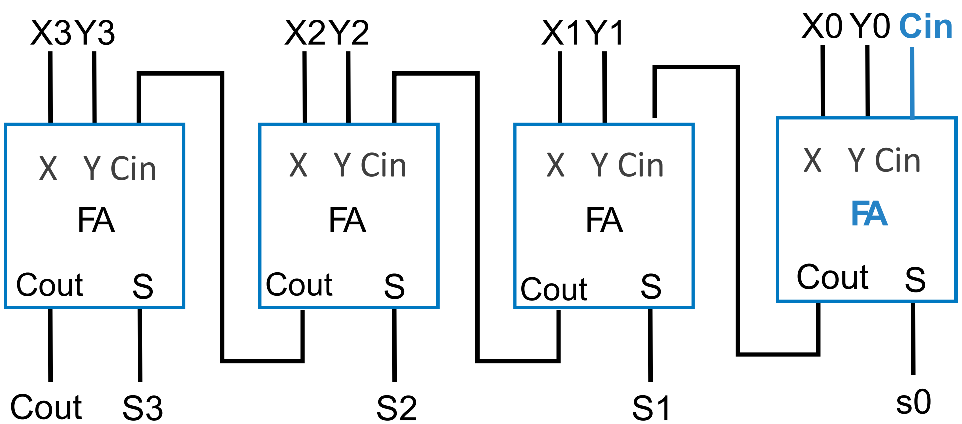
Descrizione	Pedice 3 2 1 0	Nome
Cin	0 1 1 0	C_i
Augendo	1 0 1 1	X_i
Addendo	<u>0 0 1 1</u>	Y_i
Somma	1 1 1 0	S_i
Riporto	0 0 1 1	C_{i+1}

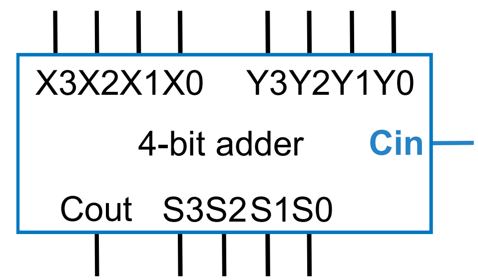
4-bit Ripple Carry adder

- Un Ripple Carry Adder a 4 bit è composto da 1 Half-adder e 3 Full Adder da 1 bit
- Ripple carry: il riporto si propaga da destra a sinistra attraverso i blocchi



- Sostituendo l'Half-adder con un Full-adder si realizza la somma
  X + Y + Cin
- Utile per la connessione in cascata di più sommatori





- Connettendo in cascata k sommatori da 4 bit si ottengono sommatori da 4k bit
- Esempio: sommatore da 8 bit ottenuto come cascata di k=2 sommatori da 4 bit: