Circuiti Combinatori

Programma delle prossime lezioni

Funzioni booleane

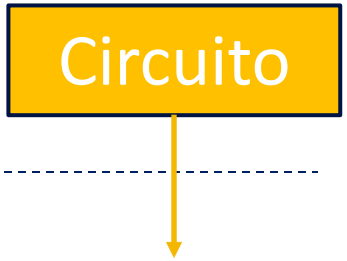
Algebra di Boole

Semplificazione logica

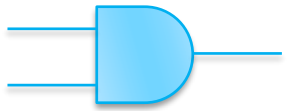
Notazione

- **Dominio**
  - L'insieme contiene solo due elementi: $B = \{0, 1\}$
- **Variabili**
  - Useremo variabili per denotare gli elementi, indicate con le lettere *x, y, z*, oppure anche *a, b, c, …*
- **Per le funzioni indicheremo**
  - $f(x_1, \ldots, x_n) : B^n \longrightarrow B$
  - Funzione scalare di *n* variabili
  - Ogni variabile prende valore 0 / 1, risultato vale solo 0 / 1
- **Funzioni vettoriali**
  - $\mathbf{F}(x_1, \ldots, x_n) : B^n \longrightarrow B^m$
  - Le tratteremo come *m* funzioni indipendenti di *n* variabili ciascuna

Operatori



- **Rappresentano i connettivi logici fondamentali**
- **AND(x, y) : $B^2 \rightarrow B$**
  - Indicato come: $x \bullet y$, xy
  - Il risultato di AND vale 1 se e solo se entrambe le variabili assumono il valore 1
  - Chiamata anche congiunzione, intersezione, greatest lower bound



- **OR(x, y) : $B^2 \rightarrow B$**
  - Indicato come: $x + y$
  - Il risultato di OR vale 1 se e solo se almeno una delle due variabili assume il valore 1
  - Chiamata anche disgiunzione, unione, least upper bound

- **NOT(x) : $B \rightarrow B$**
  - Indicato come: $\overline{x}$ x' $\sim x$
  - Il risultato di NOT vale 1 se x vale 0, e vale 0 se x vale 1
  - Chiamata anche complementazione o negazione

Tabella della verità

- **Funzione definita *enumerando* i valori che essa assume in corrispondenza a ciascuna combinazione di ingressi**
  - Invece di un "grafico" rappresentiamo la funzione tramite una tabella, detta ***tabella della verità*** della funzione

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

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

x	NOT
0	1
1	0

Altri esempi

- **Confronto**
  - $f(x, y)$ vale 1 solo se $x = y$

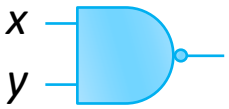
x	y	Conf
0	0	1
0	1	0
1	0	0
1	1	1

- **Confronto multiplo**
  - $f(a, b, c)$ vale 1 solo se $a = b$ e $b \neq c$

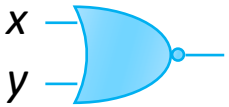
a	*b*	*c*	*f(a, b, c)*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Altri operatori a 2 variabili

- **NAND**
  - $(x \bullet y)'$



- **NOR**
  - $(x + y)'$



x	y	NAND	NOR	XOR	XNOR
0	0	1	1	0	1
0	1	1	0	1	0
1	0	1	0	1	0
1	1	0	0	0	1

- **EXOR – EXNOR**
  - $x \oplus y = xy' + x'y$
  - $(x \oplus y)' = x'y' + xy$

Dominio delle funzioni

- **Il numero delle possibili combinazioni dei valori delle variabili è finito**
  - Con *n* variabili e 2 soli valori (0 e 1) sono possibili solo 2^n diverse combinazioni
- **Esempio**
  - B^2 = { (0, 0), (0, 1), (1, 0), (1, 1) }
  - B^3 = { (0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1),
    (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1) }

Dimensione delle tabelle

- **Esempi: *n* è il numero degli ingressi**
  - $n = 4$ $2^{4-3} = 2^1$ = 2 byte
  - $n = 8$ $2^{8-3} = 2^5$ = 32 byte
  - $n = 16$ $2^{16-3} = 2^{13}$ = 8 kbyte
  - $n = 32$ $2^{32-3} = 2^{29}$ = 512 Mbyte
  - $n = 36$ $2^{36-3} = 2^{33}$ = 8 Gbyte
- **Tabelle quindi difficilmente manipolabili**
  - Funzioni di decine di variabili sono normali (un sommatore a 16 bit ha 32 + 1 ingressi)
  - Occorre trovare una rappresentazione delle funzioni molto più compatta

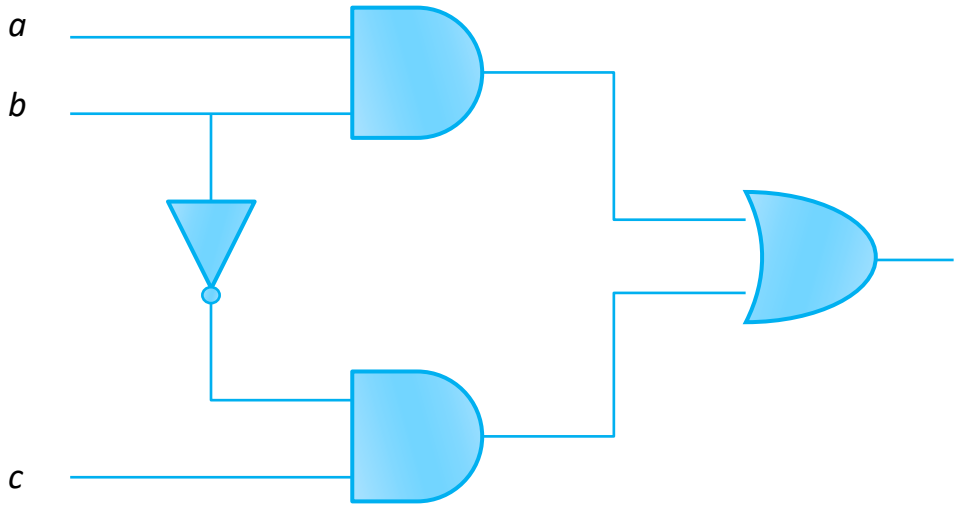
Espressioni

- **Una funzione può essere anche rappresentata come un'espressione facente uso degli operatori già definiti**
  - $f(a, b, c) = ab + b'c$
  - Utilizziamo gli operatori AND, OR e NOT
  - Convenzionalmente diamo precedenza alle operazione AND rispetto alla OR

a	*b*	*c*	*f(a, b, c)*
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Circuito

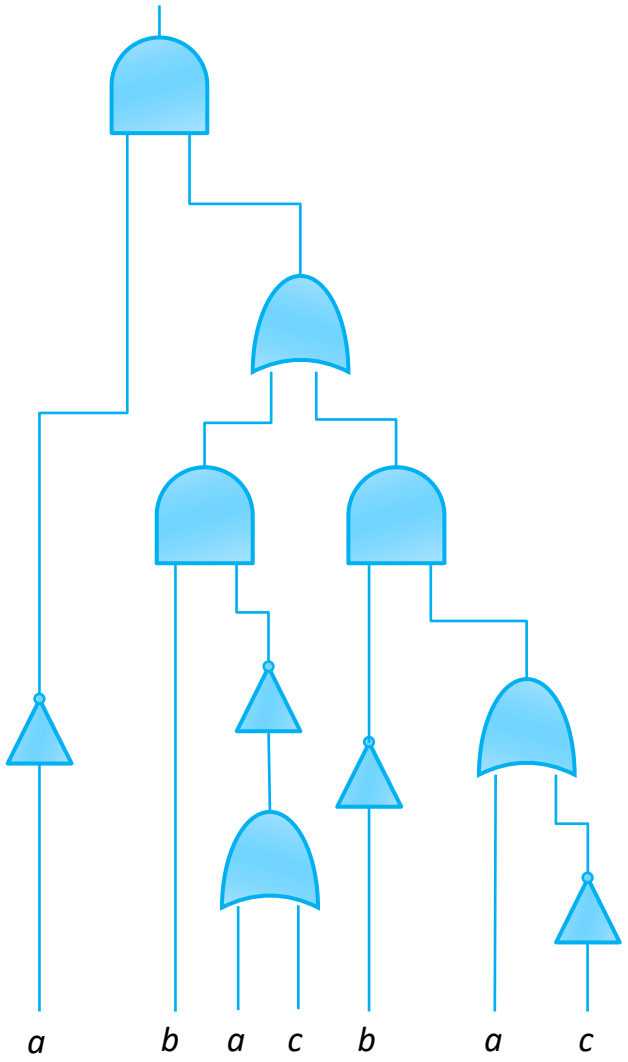
- **L'espressione può essere realizzata combinando i circuiti delle porte logiche**
  - $f(a, b, c) = ab + b'c$



a	*b*	*c*	*f(a, b, c)*
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

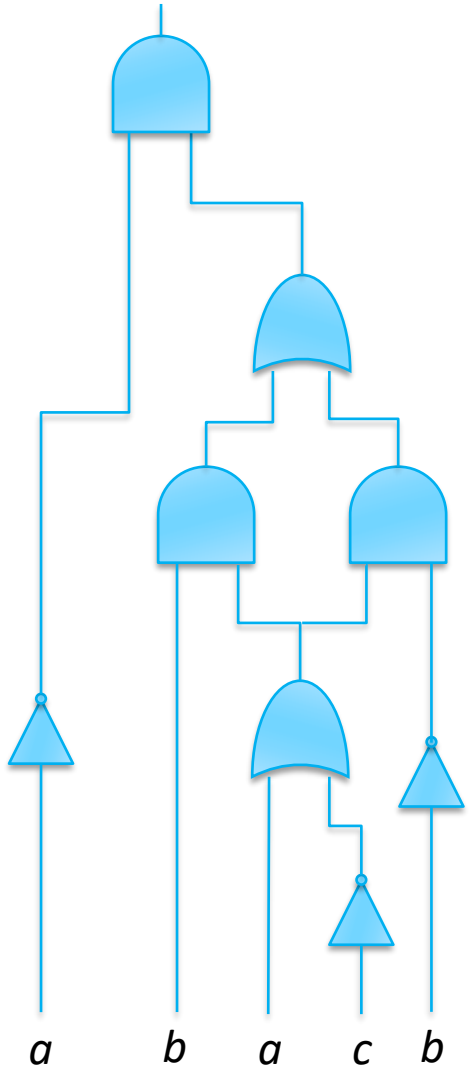
$f(a, b, c) = a' (b(a + c)' + b'(a + c'))$

- **Operatore più esterno (l'ultimo)**
  - $f(a, b, c) = a' \bullet (\ldots)$
  - Porta AND
- **Operatore successivo**
  - $f(a, b, c) = a' \bullet (\ldots + \ldots)$
  - Porta OR
- **Successivo ancora**
  - $f(a, b, c) = a' \bullet (b \bullet (\ldots)' + b' \bullet (\ldots))$
  - Due porte AND
  - Complementazioni
- **Infine**
  - $f(a, b, c) = a' (b(a + c)' + b'(a + c'))$
  - Due porte OR



Esempio: dal circuito all'espressione

- **Considerate il circuito a destra**
  - Attenzione che è un po' diverso da prima
- **Partendo dall'uscita**
  - $f(a, b, c) = a' \bullet (\ldots)$
  - $f(a, b, c) = a' \bullet (\ldots + \ldots)$
  - $f(a, b, c) = a' \bullet (b \bullet (\ldots) + (\ldots) \bullet b')$
  - $f(a, b, c) = a' (b(a + c') + (a + c')b')$



Espressioni

- **Diverse espressioni possono rappresentare la medesima funzione**
  - Per l'esempio precedente:
  - $f(a, b, c) = ab + b'c$
  - $f(a, b, c) = a'b'c + ab'c + abc' + abc$
  - Verificate che le due espressioni abbiano la stessa tabella
- **La dimensione delle espressioni è meno influenzata dal numero di variabili**
  - L'espressione dipende da cosa la funzione calcola, non dal numero di variabili
  - Difficile determinare una dimensione a priori

a	*b*	*c*	*f(a, b, c)*
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Uso delle funzioni logiche

A che servono le funzioni logiche???

- **Possiamo usare una funzione logica per esprimere le finalità di un sistema**
- **Esempio**
  - Abbassare le tende se c'è il sole o se si preme un comando
  - A meno che non ci sia troppo vento
- **Variabili**
  - **sole**: 0 se non c'è, 1 se c'è
  - **comando:** 0 non attivo, 1 attivo
  - **vento:** 0 non c'è vento, 1 c'è troppo vento
  - **tende:** 0 alzate, 1 abbassate

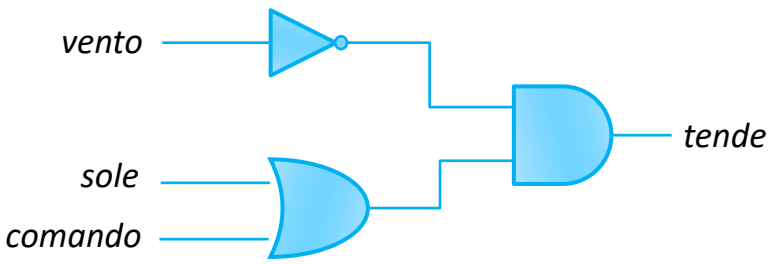
Tabella della verità

sole	*comando*	*vento*	*tende*
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

- **Abbassa solo se non c'è vento, e solo se c'è il sole oppure se è stato dato il comando**

Funzione corrispondente

- Se c'è il vento, le tende devono stare alzate (tende = 0), quindi occorre mettere il complemento di vento a fattore con AND
  - tende = vento' · …
- Le tende devono essere abbassate (tende = 1) se c'è il sole (sole = 1) oppure (inclusivo) se viene dato il comando (comando = 1)
  - tende = … · (sole + comando)
- Complessivamente
  - tende = vento' · (sole + comando)

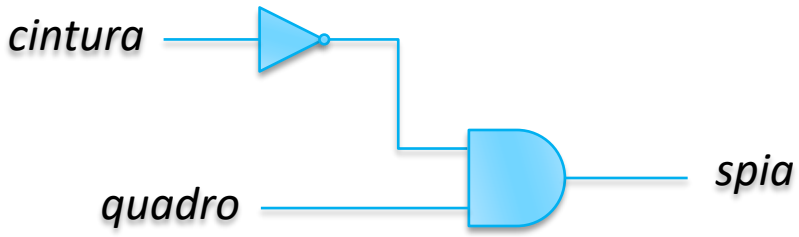


Esempio di progetto

- **Realizzazione di spia della cintura di sicurezza**
  - Si realizzi un dispositivo per automobili che accenda una spia qualora l'automobile sia accesa e la cintura di sicurezza *non* sia allacciata
  - Derivare la funzione logica che rappresenta la specifica
  - Dalla funzione, disegnare il circuito corrispondente

Esempio di progetto

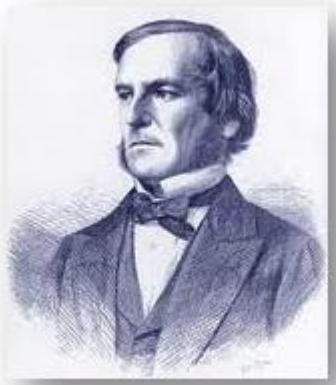
- **Realizzazione di spia della cintura di sicurezza**
  - Si realizzi un dispositivo per automobili che accenda una spia qualora l'automobile sia accesa e la cintura di sicurezza *non* sia allacciata
  - **Variabili** del circuito
    - **spia** (uscita): 0 spenta, 1 accesa
    - **quadro** (ingresso): 0 spento, 1 acceso
    - **cintura** (ingresso): 0 slacciata, 1 allacciata
  - **spia = quadro • cintura'**



Take away

- Ogni sistema segue una certa logica di funzionamento
- Possiamo esprimere tale logica tramite delle funzioni booleane
- Le funzioni possono essere rappresentate da delle espressioni dalle quali è facile ricavare un circuito

Obiettivo: manipolare funzioni complesse e gestire il funzionamento di una rete nel tempo



Algebra di Boole

Come manipolo le espressioni?

Manipolazioni algebriche

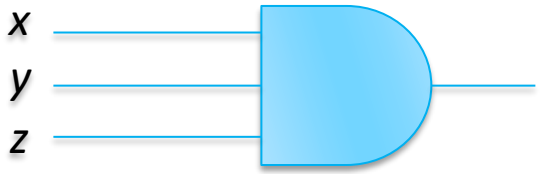
- **Le espressioni possono essere manipolate grazie alle proprietà delle operazioni fondamentali**
  - Ogni proprietà esprime una **equivalenza**: *due espressioni sono equivalenti se rappresentano la medesima funzione*
  - Usiamo le proprietà per derivare una nuova espressione equivalente a quella data
- **Esempio: Idempotenza**
  - $x + x = x$
  - $x \bullet x = x$
- **Come si dimostra?**
  - Basta provare tutte le combinazioni di ingresso (induzione completa)
  - In altre parole, si confronta la tabella della verità dell'espressione a sinistra con quella dell'espressione a destra
  - Se sono uguali allora le espressioni sono equivalenti

Assiomi dell'algebra booleana

- **Identità**
  - $x + 0 = x$
  - $y \bullet 1 = y$
- **Commutativa**
  - $x + y = y + x$
  - $x \bullet y = y \bullet x$
- **Distributiva**
  - $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$
  - $x + (y \bullet z) = (x + y) \bullet (x + z)$
- **Complementazione**
  - $x + x' = 1$
  - $x \bullet x' = 0$

Teoremi dell'algebra booleana

- **Proprietà associativa**
  - $x + (y + z) = (x + y) + z$
  - $x\ (y\ z) = (x\ y)\ z$
  - Questa proprietà ci permette di definire in maniera univoca le operazioni AND ed OR di più di 2 variabili
  - Analogamente si hanno porte logiche a più ingressi



Teoremi dell'algebra booleana

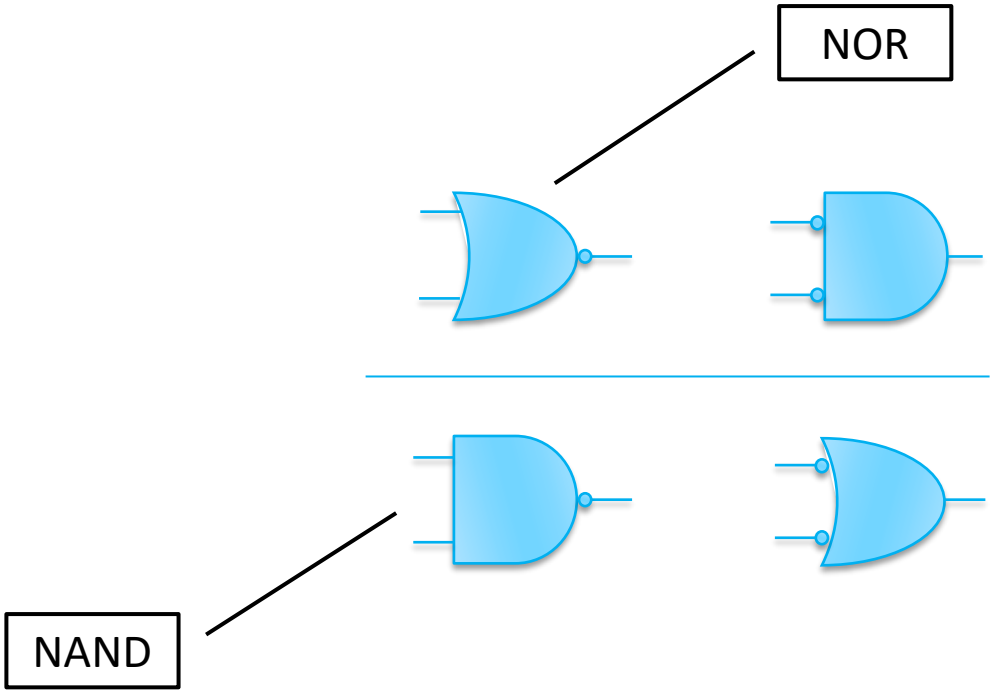
- **Legge dell'elemento nullo**
  - $x + 1 = 1$
  - $x \cdot 0 = 0$
- **Involuzione**
  - $(x')' = x$
- **Idempotenza**
  - $x + x = x$
  - $x \cdot x = x$
- **Assorbimento**
  - $x + xy = x$
  - $x(x + y) = x$
  - Dimostrazione

$$x + xy = x1 + xy = x(1 + y) = x1 = x$$

- **Semplificazione**
  - $x + x'y = x + y$
  - $x(x' + y) = xy$
- **Adiacenza**
  - $xy + xy' = x$
  - $(x + y)(x + y') = x$

Teoremi dell'algebra booleana

- **Legge di De Morgan**
  - $(x + y)' = x' \bullet y'$ $\quad (x' + y')' = x \bullet y$
  - $(x \bullet y)' = x' + y'$ $\quad (x' \bullet y')' = x + y$
  - Questa legge trasforma una OR in una AND, e viceversa



Manipolare espressioni

- **Le proprietà ci consentono di ridurre la dimensione delle espressioni, e quindi dei circuiti**
- **Ma come trovo una prima espressione se ho solo una tabella della verità?**

Semplifico il problema

a	*b*	*c*	*f*(*a*, *b*, *c*)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

a	*b*	*c*	*f*(0, *b*, *c*)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0

a	*b*	*c*	*f*(1, *b*, *c*)
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Spezzo in 2

$f(0, b, c) = b' + c'$

$f(1, b, c) = c$

Come ricombinarle?

Aggiungo condizione

a	b	c	f(a, b, c)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

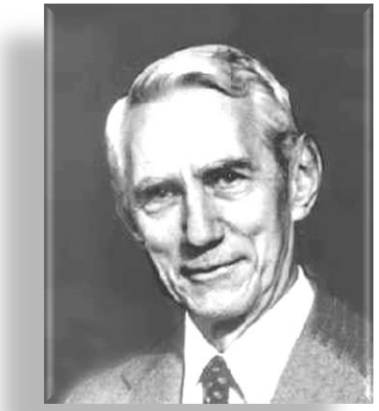
a	b	c	f(a, b, c)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

a	b	c	f(a, b, c)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Spezzo in 2 $a'(b' + c')$ ac

$$f(a, b, c) = ac + a'(b' + c')$$

Teorema di espansione di Shannon



- **Sia *f* una funzione di *n* variabili. Allora**
  - $f(x_1, \ldots, x_n) = x_1 \cdot f(1, x_2, \ldots, x_n) + x'_1 \cdot f(0, x_2, \ldots, x_n)$
- **La variabile x_1 seleziona una delle due forme della funzione**
  - Quella in cui si sostituisce 1 quando $x_1 = 1$
  - Quella in cui si sostituisce 0 quando $x_1 = 0$
- **Riduzione delle variabili**
  - Notare come $f(1, x_2, \ldots, x_n)$ e $f(0, x_2, \ldots, x_n)$ non dipendano più da x_1

Esempio

- **Sia $f(a, b, c) = a'(b' + c') + c(a + b')$**
  - $f(a, b, c) = a \bullet f(1, b, c) + a' \bullet f(0, b, c)$
    $= a \bullet [1'(b' + c') + c(1 + b')] + a' \bullet [0'(b' + c') + c(0 + b')]$
    $= a \bullet [0(b' + c') + c(1)] + a' \bullet [1(b' + c') + c(b')]$
    $= a \bullet [c] + a' \bullet [(b' + c') + cb']$
    $= a \bullet [c] + a' \bullet [b' + c' + cb']$
    $= ac + a'(b' + c' + cb')$
- **Si noti**
  - $f(1, b, c) = c$
    $f(0, b, c) = b' + c' + cb'$

Applicazione ricorsiva

- **Le funzioni $f(1, x_2,\ldots, x_n)$ e $f(0, x_2,\ldots, x_n)$ possono a loro volta essere espanse sulla variabile x_2**
  - $f(x_1, \ldots, x_n) = x_1 \cdot f(1, x_2,\ldots, x_n) + x'_1 \cdot f(0, x_2,\ldots, x_n)$
  - $f(x_1, \ldots, x_n) = x_1 \cdot [x_2 \cdot f(1, 1, x_3,\ldots, x_n) + x'_2 \cdot f(1, 0, x_3,\ldots, x_n)] + x'_1 \cdot [x_2 \cdot f(0, 1, x_3,\ldots, x_n) + x'_2 \cdot f(0, 0, x_3,\ldots, x_n)]$
  - $f(x_1, \ldots, x_n) = x_1x_2 \bullet f(1, 1, x_3,\ldots, x_n) + x_1x'_2 \bullet f(1, 0, x_3,\ldots, x_n) + x'_1x_2 \bullet f(0, 1, x_3,\ldots, x_n) + x'_1x'_2 \bullet f(0, 0, x_3,\ldots, x_n)$
- **Osservare la struttura**
  - Un prodotto di variabili moltiplica la funzione valutata sulle variabili che assumono valore 1 se in forma affermata, e 0 se in forma negata

Espansione completa

La funzione *f* valutata in un punto è un preciso valore (0 o 1), che possiamo ottenere dalla tabella della verità

- **Espandendo su tutte le variabili**
  - $f(x_1, \ldots, x_n) = x_1x_2x_3 \cdots x_{n-1}x_n \cdot f(1, 1, 1, \ldots, 1, 1) +$
    $+ x_1x_2x_3 \cdots x_{n-1}x'_n \cdot f(1, 1, 1, \ldots, 1, 0) +$
    $+ x_1x_2x_3 \cdots x'_{n-1}x_n \cdot f(1, 1, 1, \ldots, 0, 1) +$
    $+ x_1x_2x_3 \cdots x'_{n-1}x'_n \cdot f(1, 1, 1, \ldots, 0, 0) +$
    $\ldots$
    $+ x'_1x_2x_3 \cdots x_{n-1}x_n \cdot f(0, 1, 1, \ldots, 1, 1) +$
    $+ x'_1x_2x_3 \cdots x_{n-1}x'_n \cdot f(0, 1, 1, \ldots, 1, 0) +$
    $\ldots$
    $+ x'_1x'_2x'_3 \cdots x_{n-1}x_n \cdot f(0, 0, 0, \ldots, 1, 1) +$
    $+ x'_1x'_2x'_3 \cdots x_{n-1}x'_n \cdot f(0, 0, 0, \ldots, 1, 0) +$
    $+ x'_1x'_2x'_3 \cdots x'_{n-1}x_n \cdot f(0, 0, 0, \ldots, 0, 1) +$
    $+ x'_1x'_2x'_3 \cdots x'_{n-1}x'_n \cdot f(0$

Il termine a prodotto è presente se e solo se la funzione in quel punto vale 1

Espansione completa

- **Espandendo su tutte le variabili**
  - $f(x_1, \ldots, x_n) = x_1x_2x_3 \cdots x_{n-1}x_n \cdot f(1, 1, 1,\ldots, 1, 1) +$
    $+ x_1x_2x_3 \cdots x_{n-1}x'_n \cdot f(1, 1, 1,\ldots, 1, 0) +$
    $+ x_1x_2x_3 \cdots x'_{n-1}x_n \cdot f(1, 1, 1,\ldots, 0, 1) +$
    $+ x_1x_2x_3 \cdots x'_{n-1}x'_n \cdot f(1, 1, 1,\ldots, 0, 0) +$
    $\ldots$
    $+ x'_1x_2x_3 \cdots x_{n-1}x_n \cdot f(0, 1, 1,\ldots, 1, 1) +$
    $+ x'_1x_2x_3 \cdots x_{n-1}x'_n \cdot f(0, 1, 1,\ldots, 1, 0) +$
    $\ldots$
    $+ x'_1x'_2x'_3 \cdots x_{n-1}x_n \cdot f(0, 0, 0,\ldots, 1, 1) +$
    $+ x'_1x'_2x'_3 \cdots x_{n-1}x'_n \cdot f(0, 0, 0,\ldots, 1, 0) +$

Si può dunque scrivere l'espansione completa di Shannon partendo dalla tabella della verità e guardando solo le righe dove la funzione vale 1

- Per ogni 1 si guarda il valore di tutte le variabili per quella riga
- Se alla variabile è assegnato 1 compare in forma affermata nel prodotto
- Se alla variabile è assegnato 0 compare in forma negata nel prodotto

Esempio

- $f(a, b, c) = a'(b' + c') + c(a + b')$
  - $f(a, b, c) = a'b'c' \bullet f(0, 0, 0) +$
    $+ a'b'c \bullet f(0, 0, 1) +$
    $+ a'bc' \bullet f(0, 1, 0) +$
    ~~$+ a'bc \bullet f(0, 1, 1) +$~~
    ~~$+ ab'c' \bullet f(1, 0, 0) +$~~
    $+ ab'c \bullet f(1, 0, 1) +$
    ~~$+ abc' \bullet f(1, 1, 0) +$~~
    $+ abc \bullet f(1, 1, 1)$
  - $f(a, b, c) = a'b'c' + a'b'c + a'bc' + ab'c + abc$

a	b	c	f(a, b, c)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- **Per casa: disegnate il circuito corrispondente**

Forme canoniche

- **L'espansione completa di una funzione è un'espressione univoca**
  - Perché i termini dell'espressione derivano dalla tabella della verità
- **Due funzioni uguali hanno quindi la stessa espansione completa**
- **L'espansione completa è quindi detta forma canonica**
  - In forma canonica, due espressioni sono uguali se e solo se le corrispondenti funzioni sono uguali
  - Non vale per espressioni normali: funzioni uguali possono essere in generale rappresentate da espressioni diverse
  - … ma hanno la stessa forma canonica

Conseguenze

- **Qualunque funzione booleana può essere espressa tramite gli operatori logici di base**
  - Basta esprimerla in forma canonica!
  - Risultato non ovvio: non vale, per esempio, in analisi
- **Bastano in realtà due soli operatori**
  - Per il teorema di De Morgan
  - $x + y = (x' \bullet y')'$ (OR espressa con sole AND e NOT)
  - $x \cdot y = (x' + y')'$ (AND espressa con sole OR e NOT)
  - Uno degli operatori di base è superfluo!

Seconda forma canonica

- **Il teorema di Shannon vale anche in forma duale**
  - $f(x_1, \ldots, x_n) = (x_1 + f(0, x_2, \ldots, x_n)) \cdot (x'_1 + f(1, x_2, \ldots, x_n))$
- **Espandendo su tutte le variabili**
  - $f(x_1, \ldots, x_n) = (x_1 + x_2 + \cdots + x_n + f(0, 0, \ldots, 0)) \cdot$
    $\cdot (x_1 + x_2 + \cdots + x'_n + f(0, 0, \ldots, 1)) \cdot$
    $\ldots$
    $\cdot (x'_1 + x'_2 + \cdots + x_n + f(1, 1, \ldots, 0)) \cdot$
    $\cdot (x'_1 + x'_2 + \cdots + x'_n + f(1, 1, \ldots, 1))$
- **Dualità**
  - In ogni termine, una variabile compare affermata se valutata in 0, negata se valutata in 1
  - Il termine viene eliminato se la funzione vale 1
  - Partendo dalla tabella, si scelgono le righe in cui la funzione vale 0

Esempio

- $f(a, b, c) = a'(b' + c') + c(a + b')$
  - $f(a, b, c) = (a + b + c + f(0, 0, 0)) \cdot$
    $\cdot (a + b + c' + f(0, 0, 1)) \cdot$
    $\cdot (a + b' + c + f(0, 1, 0)) \cdot$
    $\cdot (a + b' + c' + f(0, 1, 1)) \cdot$
    $\cdot (a' + b + c + f(1, 0, 0)) \cdot$
    $\cdot (a' + b + c' + f(1, 0, 1)) \cdot$
    $\cdot (a' + b' + c + f(1, 1, 0)) \cdot$
    $\cdot (a' + b' + c' + f(1, 1, 1))$

a	b	c	f(a, b, c)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

  - $f(a, b, c) = (a + b' + c') \cdot (a' + b + c) \cdot (a' + b' + c)$

- **Per casa: disegnate il circuito corrispondente**

SOP e POS

- **La prima forma canonica è anche detta *somma di prodotti***
  - O, in Inglese, Sum Of Products (SOP)
  - E' una grossa somma, in cui ogni termine è un prodotto di variabili affermate o negate
  - $f(a, b, c) = a'b'c' + a'b'c + a'bc' + ab'c + abc$

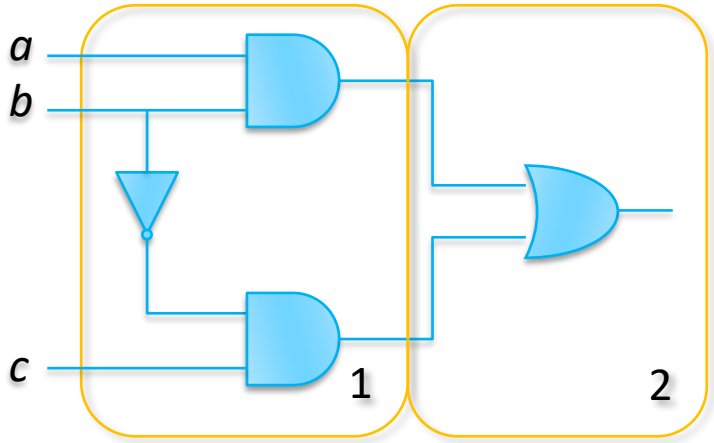
- **La seconda forma canonica è anche detta *prodotto di somme***
  - O, in Inglese, Product Of Sums (POS)
  - E' un grosso prodotto, in cui ogni termine è una somma di variabili affermate o negate
  - $f(a, b, c) = (a + b' + c') \bullet (a' + b + c) \bullet (a' + b' + c)$

Sintesi a due livelli

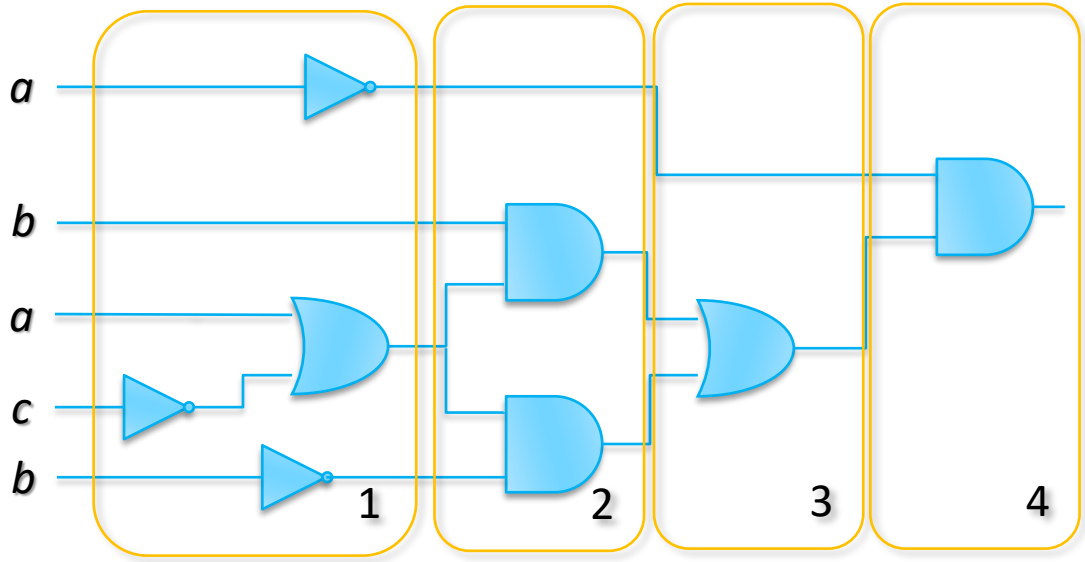
Come rendere la manipolazione sistematica

I livelli

Livello: massimo numero di porte logiche attraversate dall'ingresso all'uscita



$f(a, b, c) = ab + b'c$



$f(a, b, c) = a' (b(a + c') + (a + c')b')$

Sintesi a due livelli

- **Ci limiteremo a mano a realizzazioni a due livelli**
  - Teoricamente semplice da ottenere
  - In prima analisi, è anche il circuito più veloce
  - Talvolta però di grosse dimensioni (e.g., addizionatore)
  - Alla base dei metodi per realizzazioni multi-livello
- **Un'espressione a due livelli deve essere**
  - O una somma di prodotti
    (primo livello tutto di AND seguito da una OR)
  - Oppure un prodotto di somme
    (primo livello tutto di OR seguito da una AND)
  - (per la proprietà associativa)

Terminologia

- **Letterali**
  - Un letterale è una variabile in forma affermata o in forma negata
  - Esempi: x x' y' b a'
  - La funzione $f(a, b, c) = a'b'c' + a'b'c + a'bc' + ab'c + abc$ ha 3 variabili e 15 letterali
  - La funzione $f(a, b, c) = a'b' + a'c' + ac$ ha 3 variabili e 6 letterali
- **Misureremo la complessità di un'espressione dal suo numero di letterali**
  - Ogni letterale diventa l'ingresso di una porta logica
  - Quindi il numero di letterali è correlato con il numero di transistori del circuito

Terminologia

- **Prodotto fondamentale (minterm)**
  - Un prodotto in cui *ogni* variabile compare una volta come letterale
  - $a'b'c'$ $a'b'c$ $a'bc'$ $ab'c$ abc
- **Somma fondamentale (maxterm)**
  - Una somma in cui *ogni* variabile compare una volta come letterale
  - $(a + b' + c)$ $(a' + b + c')$ $(a' + b' + c)$ $(a + b + c)$
- **Le forme canoniche fanno uso di minterm e maxterm**
  - I minterm di una funzione corrispondono agli 1 della sua tabella della verità
  - I maxterm di una funzione corrispondono agli 0 della sua tabella della verità

Implicanti

- **Siano *f* e *g* funzioni di *n* variabili**
- **Si definisce *g* un *implicante* di *f* se e solo se, per qualunque assegnamento (x_1, …, x_n) alle variabili**
  - **Se $g(x_1, \ldots, x_n) = 1$, allora $f(x_1, \ldots, x_n) = 1$**
  - Quando *g* è un implicante di *f*, se *g* vale 1 in qualche punto, allora anche *f* vale 1, ma se *g* vale 0, allora *f* può essere indifferentemente 0 o 1
- **Esempio:**
  - Sia $f(x, y, z) = xy + z$
  - Le funzioni $g(x, y, z) = xy$ $g(x, y, z) = z$ $g(x, y, z) = y'z$ sono implicanti di *f*
  - La funzione $g(x, y, z) = x$ non è un implicante

Esempio

x	*y*	*z*	*f*(*x*, *y*, *z*)	*xy*	*z*	*y'z*	*x*
0	**0**	**0**	0	0	0	0	0
0	**0**	**1**	1	0	1	1	0
0	**1**	**0**	0	0	0	0	0
0	**1**	**1**	1	0	1	0	0
1	**0**	**0**	0	0	0	0	1
1	**0**	**1**	1	0	1	1	1
1	**1**	**0**	1	1	0	0	1
1	**1**	**1**	1	1	1	0	1

- Ogni implicante *copre* una parte della funzione
- Abbiamo fatto a pezzi la funzione $f(x, y, z) = xy + z$
  - $f(x, y, z) = y'z + z + xy$

Implicanti come mattoni

- **Un implicante rappresenta un "pezzo" di una funzione**
  - L'implicante può essere usato per rappresentare parte, ma non necessariamente tutti gli 1 di una funzione
  - Vari implicanti possono essere usati per rappresentare differenti pezzi di una funzione
  - I vari pezzi possono anche sovrapporsi
  - La funzione è data dall'OR di appropriati implicanti

Come trovare gli implicanti?

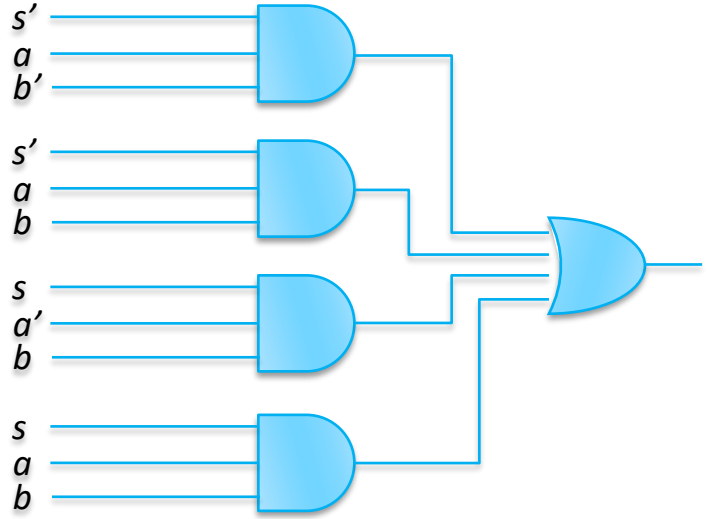
- **Partiamo dai minterm**
  - Dopo tutto i minterm sono implicanti!
  - Ognuno copre però soltanto un 1 della funzione
- **Cerchiamo di espanderli**
  - Usiamo le proprietà dell'algebra Booleana
  - Mettiamo assieme dei minterm in modo da formare termini più grossi (con meno letterali)
  - Ci limitiamo ad implicanti in forma di **prodotto di letterali**
    - ab' : ok
    - $ab' + a'b$: non ok

Esempio: il multiplexer 2 a 1

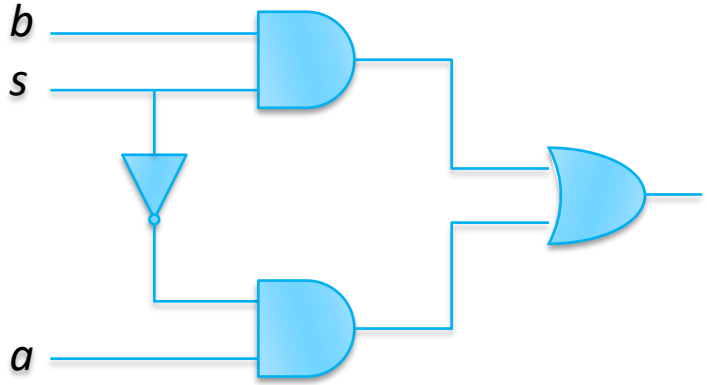
- **Si progetti un circuito che permetta di scegliere uno tra due segnali *a* e *b* tramite un comando *s***
  - Quando *s* = 0, si deve scegliere *a*
  - Quando *s* = 1, si deve scegliere *b*
- **Scriviamo la prima forma canonica**
  - *m = s'ab' + s'ab + sa'b + sab*
- **Applichiamo l'adiacenza logica**
  - *xy + xy' = x*
  - Supponiamo che *x* sia un gruppo di letterali, ed *y* una variabile
  - Sulla mappa corrispondono a righe in cui ***m* = 1** e che **differiscono per una sola variabile**
  - *m = s'ab' + s'ab + sa'b + sab*
  - *m = s'a + sb*
  - Ridotto da 12 a 4 letterali

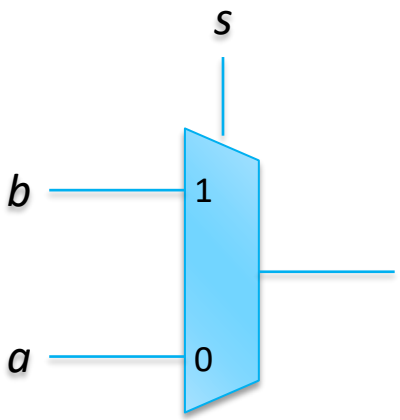
s	*a*	*b*	*m(a,b,s)*
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

Circuiti



$m = s'ab' + s'ab + sa'b + sab$





$m = s'a + sb$

Interpretazione geometrica

- **Nella mappa, chiamiamo term un gruppo di 1 di *f* che non sia un minterm**
  - Con un solo colpo prendiamo diversi 1
  - E' come scrivere la forma canonica, ma si tralasciano le variabili il cui valore cambia
  - *s'ab'* + *s'ab* diventa *s'a*
  - **Si possono scrivere direttamente!**
  - Facili da individuare quando le righe sono una vicina all'altra
  - Meno facile quando sono lontane
- **Trucco**
  - Disponiamo le righe in modo che cambi una sola variabile alla volta
  - In questo modo termini logicamente adiacenti sono vicini anche fisicamente sulla tabella

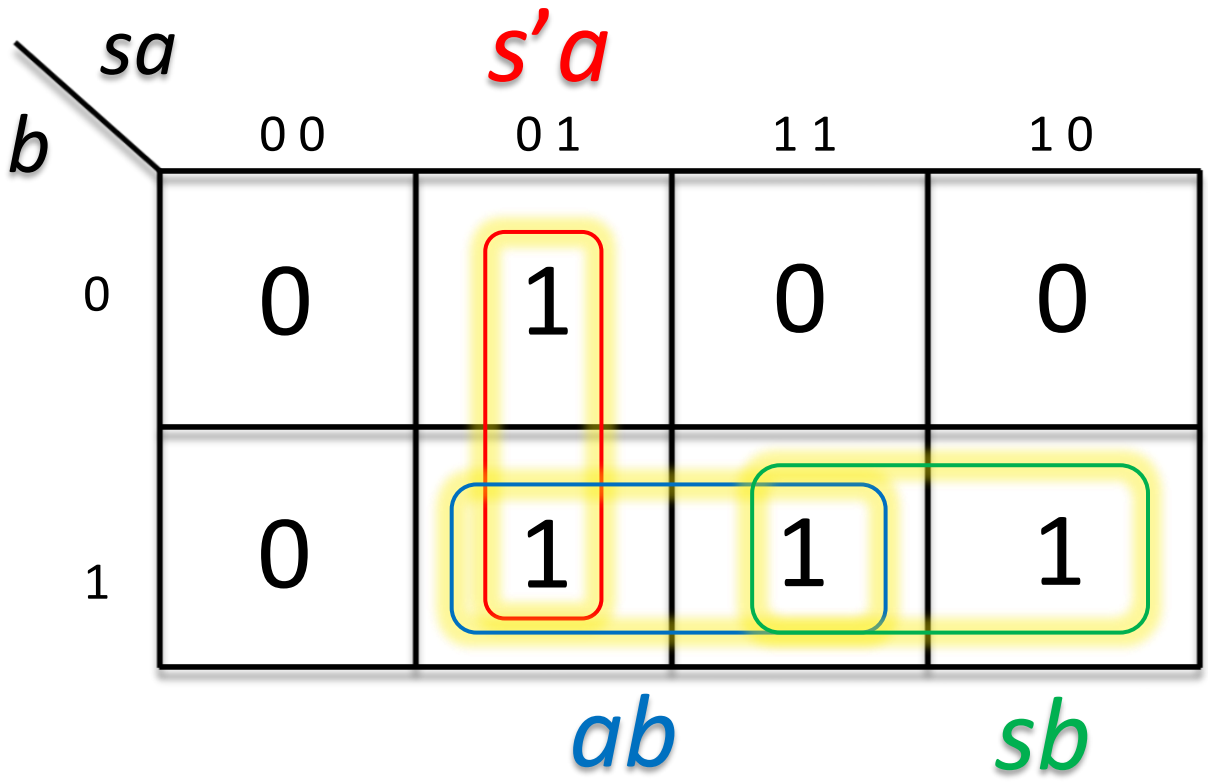
s	*a*	*b*	*f*(*a*, *b*, *s*)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Interpretazione geometrica

- **Modifichiamo l'ordine delle righe**
  - Invece del codice binario normale si usa il codice Gray
  - Tra una riga e la successiva cambia solamente una variabile
  - Adesso entrambi i termini sono vicini
- **Non funziona sempre**
  - Abbiamo una sola dimensione
  - Un termine risulta ancora spezzato
  - Dobbiamo sfruttare dimensioni aggiuntive

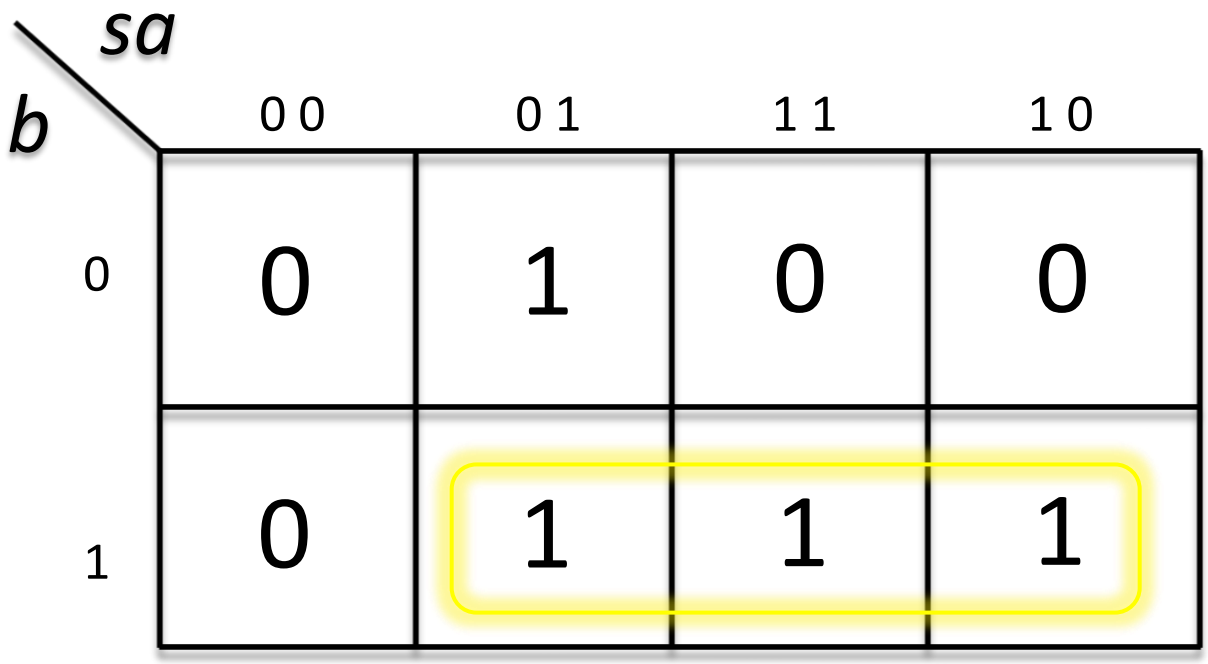
s	*a*	*b*	*f*(*a*, *b*, *s*)
0	0	0	**0**
0	0	1	**0**
0	1	1	**1**
0	1	0	**1**
1	1	0	**0**
1	1	1	**1**
1	0	1	**1**
1	0	0	**0**

La mappa di Karnaugh



- **La prima colonna è vicina all'ultima!**

La mappa di Karnaugh



- **E questo raggruppamento?**
  - Non è esprimibile tramite un solo prodotto
  - E' la somma dei due implicanti precedenti: *ab* + *sb*

Take away

- **Ci limitiamo a fare espressioni a due livelli**
  - Gli implicanti sono termini prodotto
  - Possiamo trovarli per adiacenza logica sull'espressione
- **Trasformiamo l'adiacenza logica in geometrica**
  - Occorre mettere vicine nello spazio le righe della tabella che sono logicamente vicine
- **Raggruppiamo gli 1 in termini**
  - Mettiamo assieme gli 1 che stanno vicini
  - Cerchiamo i raggruppamenti più grossi (hanno meno letterali)

Adiacenze a tre variabili: 2 uni

a \ *bc*	00	01	11	10
0	1	1	0	0
1	0	0	0	0

$a'b'$

a \ *bc*	00	01	11	10
0	0	0	0	0
1	1	1	0	0

ab'

a \ *bc*	00	01	11	10
0	0	1	1	0
1	0	0	0	0

$a'c$

a \ *bc*	00	01	11	10
0	0	0	0	0
1	0	1	1	0

ac

a \ *bc*	00	01	11	10
0	0	0	1	1
1	0	0	0	0

$a'b$

a \ *bc*	00	01	11	10
0	0	0	0	0
1	0	0	1	1

ab

Adiacenze a tre variabili: 2 uni

a \ *bc*	00	01	11	10
0	1	0	0	1
1	0	0	0	0

$a'c'$

a \ *bc*	00	01	11	10
0	0	0	0	0
1	1	0	0	1

ac'

a \ *bc*	00	01	11	10
0	1	0	0	0
1	1	0	0	0

$b'c'$

a \ *bc*	00	01	11	10
0	0	1	0	0
1	0	1	0	0

$b'c$

a \ *bc*	00	01	11	10
0	0	0	1	0
1	0	0	1	0

bc

a \ *bc*	00	01	11	10
0	0	0	0	1
1	0	0	0	1

bc'

Adiacenze a tre variabili: 4 uni

a \ *bc*	00	01	11	10
0	1	1	0	0
1	1	1	0	0

b'

a \ *bc*	00	01	11	10
0	0	1	1	0
1	0	1	1	0

c

a \ *bc*	00	01	11	10
0	0	0	1	1
1	0	0	1	1

b

a \ *bc*	00	01	11	10
0	1	0	0	1
1	1	0	0	1

c'

a \ *bc*	00	01	11	10
0	1	1	1	1
1	0	0	0	0

a'

a \ *bc*	00	01	11	10
0	0	0	0	0
1	1	1	1	1

a

Adiacenze a tre variabili: tutti o nessuno

a \ *bc*	00	01	11	10
0	1	1	1	1
1	1	1	1	1

1

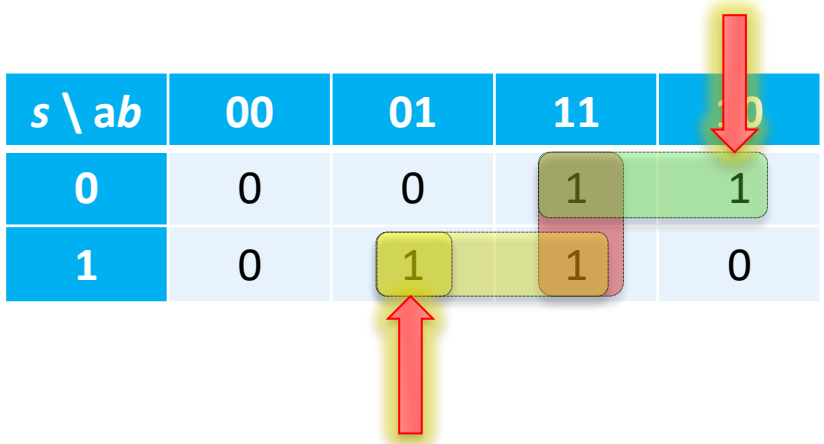
a \ *bc*	00	01	11	10
0	0	0	0	0
1	0	0	0	0

0

Esempi di semplificazione

Come scegliamo gli implicanti?

- **Alcuni implicanti sono primi**
  - Si dice che un implicante è primo quando **non è contenuto in un altro implicante**
  - Gli implicanti contenuti in altri implicanti si possono buttare via
  - Nel caso del multiplexer i minterm *non* sono primi (*sa'b* → *sb*)



s \ *ab*	00	01	11	10
0	0	0	1	1
1	0	1	1	0

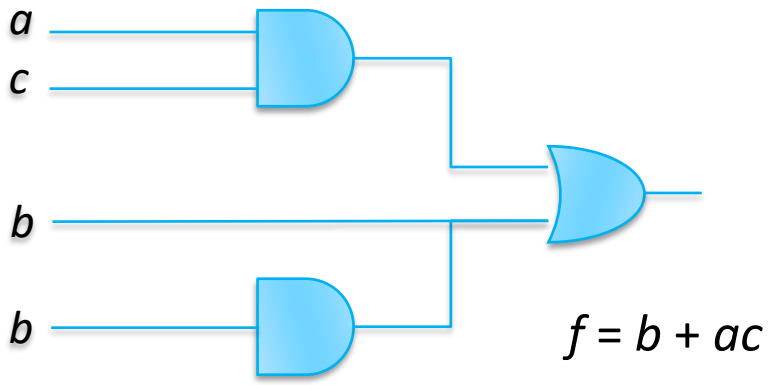
- Alcuni implicanti primi sono **ridondanti**
  - Per il multiplexer ci sono 3 implicanti primi
  - *s'a* *sb* *ab*
  - Uno dei tre non serve (*ab*) perché gli altri da soli già coprono la funzione
- Alcuni implicanti primi sono **essenziali**
  - Coprono un 1 non coperto da nessun altro implicante primo
  - Devono necessariamente comparire in qualunque copertura minima
  - *s'a* ed *sb* sono essenziali

Procedimento

- **Obiettivo: trovare il più piccolo insieme di implicanti primi che ricoprano tutta la funzione**
  - Primi perché altrimenti ci sarebbero implicanti con meno letterali
  - Insieme più piccolo per ridurre il numero di termini
  - Alcuni implicanti primi potrebbero essere esclusi (ridondanti)
- **Si scrive la tabella della verità in forma di mappa di Karnaugh**
  - Ricordare che la numerazione segue il codice Gray
- **Partendo dai minterm ci si espande per trovare gli implicanti primi**
  - Espandersi in tutte le direzioni per trovare le adiacenze
  - Quando non ci si può più espandere si è arrivati all'implicante primo
- **Si scrive l'espressione degli implicanti guardando le variabili che sono costanti**
  - Basta guardare le cifre scritte sulle righe e colonne della mappa
- **Si sceglie un insieme di implicanti non ridondante**
  - Prendiamo tutti gli implicanti primi essenziali
  - Si coprono i rimanenti 1 scegliendo un insieme piccolo (cosa non banale)

Esempio

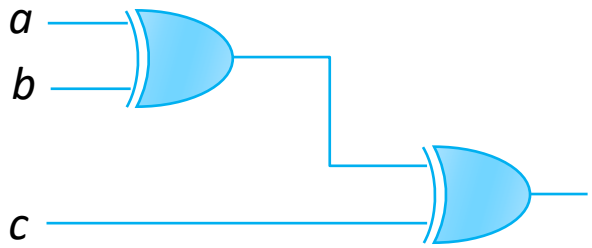
a \ *bc*	00	01	11	10
0	0	0	1	1
1	0	1	1	1



- **Implicanti primi**
  - *b* *ac*
  - Entrambi essenziali
  - *f* = *b* + *ac*
- **Nota bene**
  - Un 1 è coperto più volte (corrisponde al minterm *abc*)
  - Non è un problema, due porte AND saranno attive
  - Meglio che non usare *ab'c*
  - La porta AND a 1 ingresso si può togliere e collegare *b* direttamente alla porta OR

Esempio: scacchiera (odd function)

$a \setminus bc$	00	01	11	10
0	0	1	0	1
1	1	0	1	0



- **I minterm non sono espandibili**
  - Sono tutti implicanti primi
  - Mappa a scacchiera non minimizzabile
  - $f = ab'c' + a'b'c + abc + a'bc'$
- **Si tratta di una rete di EXOR**
  - $f = b'(ac' + a'c) + b(ac + a'c')$
  - $f = b'(ac' + a'c) + b(ac' + a'c)' = b'(a \oplus c) + b(a \oplus c)'$
  - $f = b \oplus (a \oplus c) = a \oplus b \oplus c$

$(ac + a'c') =$
$(ac + a'c')'' =$
$((ac)' (a'c')')' =$
$((a' + c') (a'' + c''))' =$
$((a' + c') (a + c))' =$
$(a'a + a'c + c'a + cc')' =$
$(a'c + c'a)' =$
$(ac' + a'c)'$

Esempio: conta uni

- **Si realizzi un circuito con 3 ingressi ed una uscita che indichi quanti ingressi sono a 1**
  - Occorre innanzi tutto codificare l'uscita
  - Per esempio la codifichiamo in binario
  - Vi sono da 0 a 3 possibili ingressi a 1, quindi 4 combinazioni (0, 1, 2 e 3)
  - Sono sufficienti due cifre binarie
- **La funzione da realizzare ha 2 uscite**
  - Le trattiamo come due funzioni indipendenti
  - Ma i loro valori devono essere tra loro collegati

Tabella della verità

a	*b*	*c*	s_1	s_0
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

- **Una colonna per ogni uscita**
  - Sono due funzioni diverse, ma le determiniamo contemporaneamente
  - Per la semplificazione le trattiamo separatamente

Mappe di Karnaugh

- **Due mappe, una per ciascuna uscita**
  - Le semplifichiamo separatamente
  - $s_1 = ac + bc + ab$
  - $s_0 = a \oplus b \oplus c$

a	b	c	s_1	s_0
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

s_1

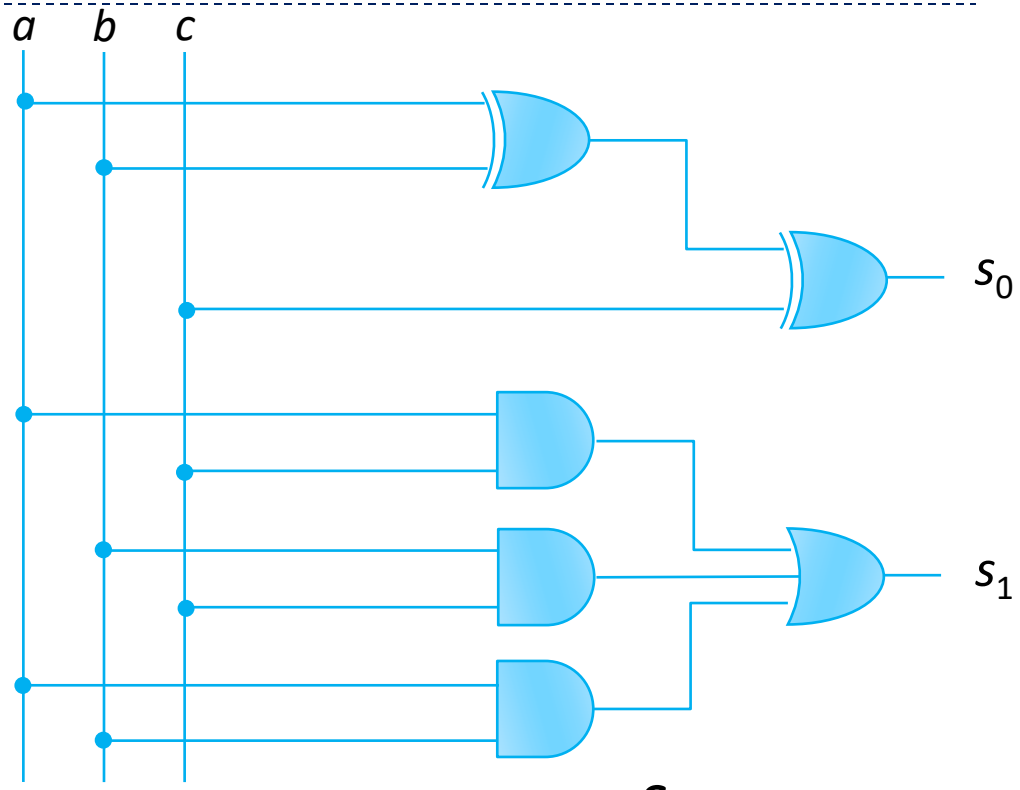
$a \setminus bc$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

s_0

$a \setminus bc$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Circuiti

- **Due circuiti**
  - $s_1 = ac + bc + ab$
  - $s_0 = a \oplus b \oplus c$



s_1

a \ bc	00	01	11	10
0	0	0	1	0
1	0	1	1	1

s_0

a \ bc	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Esempio: priority encoder

- **Come prima, ma questa volta vogliamo sapere l'indice dell'ingresso a 1 con valore più alto**
  - Diamo ad *a* indice 1, a *b* indice 2 e a *c* indice 3
  - Se *c* = 1, l'uscita deve valere 3
  - Se *c* = 0 e *b* = 1, l'uscita deve valere 2
  - Se *c* = 0, *b* = 0 e *a* = 1, l'uscita deve valere 1
  - Se sono tutti a 0, l'uscita vale 0
- **Utile per esempio per codificare le priorità degli interrupt**
  - Ci serve l'indice dell'interrupt attivo a priorità più alta

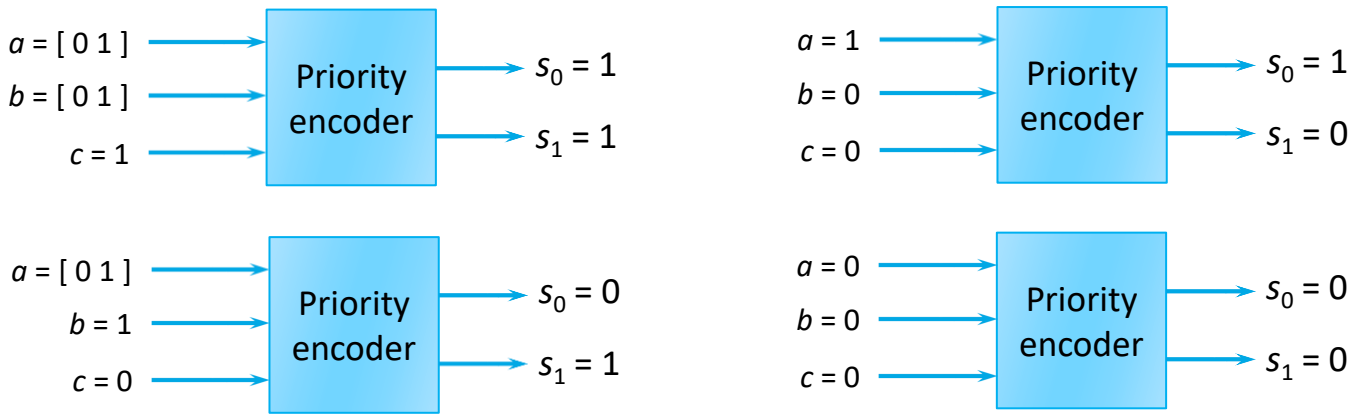


Tabella e mappe

- **Semplificando**
  - $s_1 = b + c$
  - $s_0 = ab' + c$

a	*b*	*c*	s_1	s_0
0	**0**	**0**	0	0
0	**0**	**1**	1	1
0	**1**	**0**	1	0
0	**1**	**1**	1	1
1	**0**	**0**	0	1
1	**0**	**1**	1	1
1	**1**	**0**	1	0
1	**1**	**1**	1	1

s_1

a \ *bc*	00	01	11	10
0	0	1	1	1
1	0	1	1	1

s_0

a \ *bc*	00	01	11	10
0	0	1	1	0
1	1	1	1	0

Lieve modifica ad s_0

- **Scambiamo le ultime due righe**
  - $s_1 = b + c$
  - Per s_0 ora la scelta è più complessa
  - Gli implicanti $a'c$ e ac' sono essenziali, quindi ci vogliono per forza
  - Il restante 1 si può coprire in due modi
  - $s_0 = a'c + ac' + ab'$
  - $s_0 = a'c + ac' + b'c$

a	b	c	s_1	s_0
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

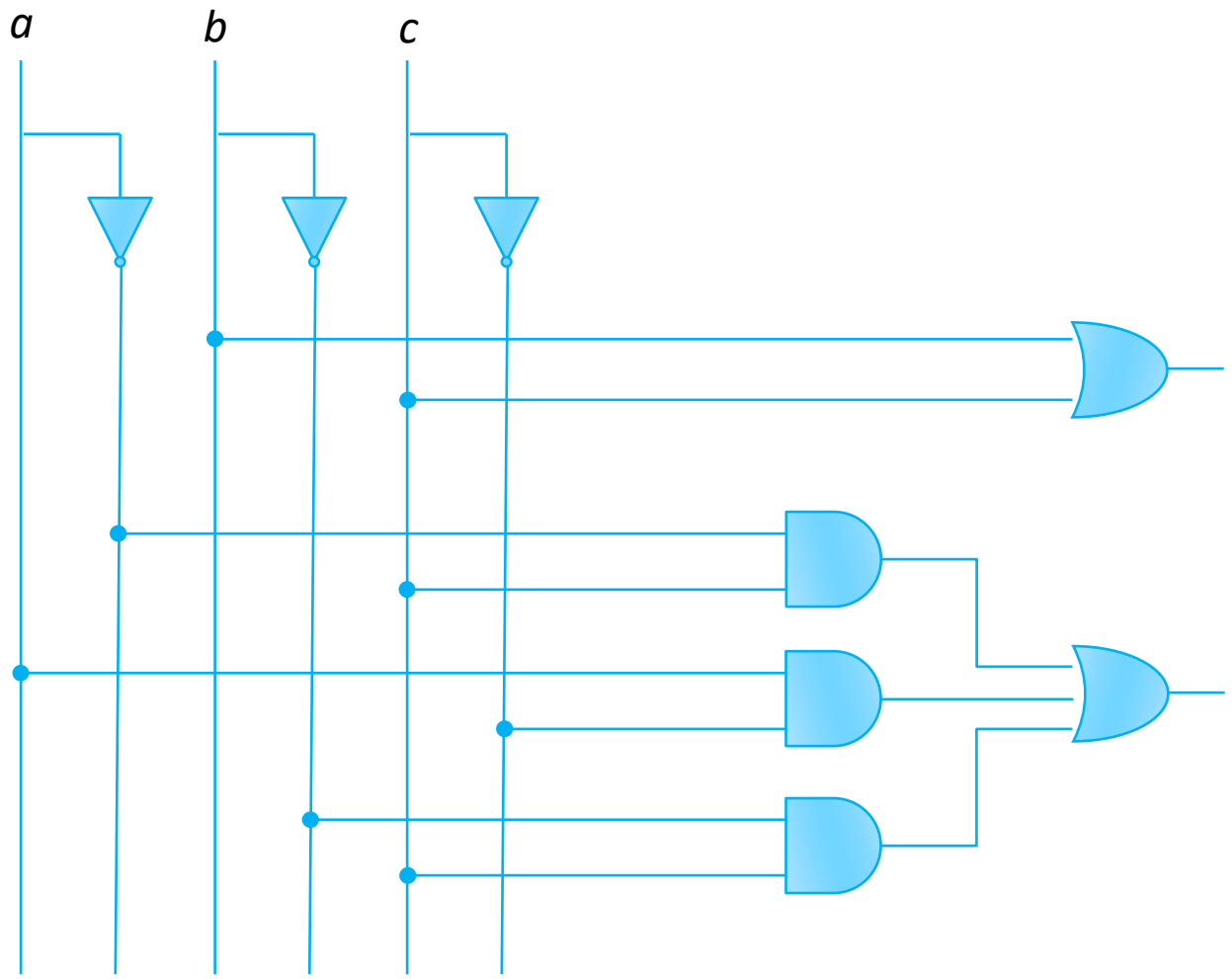
s_1

$a \setminus bc$	00	01	11	10
0	0	1	1	1
1	0	1	1	1

s_0

$a \setminus bc$	00	01	11	10
0	0	1	1	
1	1	1	0	1

Circuito



$s_1 = b + c$

$s_0 = a'c + ac' + b'c$

Take away

- **Avete visto come semplificare una funzione di 3 variabili**
  - Procedimento piuttosto semplice
  - Attenzione solo alla scelta degli implicanti
  - Con un po' di pratica si fa molto in fretta
- **Se ci sono più uscite le si trattano separatamente**
  - Ma la funzione vettoriale deve essere progettata nel suo complesso
  - Si potrebbe fare la minimizzazione congiunta
- **Facile costruire blocchetti di uso generico**
  - Multiplexer
  - Conta uni
  - Priority encoder

Mappe a 4 e più variabili

- **Le mappe di Karnaugh si possono scrivere anche a 4 e più ingressi**
  - Diventano più grosse
  - Trovare le adiacenze diventa un poco più complesso, soprattutto sopra le 5 variabili
- **Il procedimento rimane comunque sempre lo stesso**
  - Trovare gli implicanti primi
  - Tenere gli essenziali
  - Scegliere tra gli altri in modo da coprire la funzione

Mappe a 4 variabili

- **I minterm saranno ora espressioni di 4 letterali**
  - *ab'cd'* *a'bcd* *abc'd'* *abcd*
- **Gruppi di due 1 saranno termini a 3 letterali**
  - *ab'd'* *a'cd* *bc'd'* *abc*
- **Gruppi di quattro 1 saranno termini a 2 letterali**
  - *b'd'* *a'c* *c'd'* *ab*
- **Gruppi di otto 1 saranno termini a 1 letterale**
  - *b'* *c* *d'* *a*
- **La mappa adesso diventa quadrata**
  - Aggiungiamo una variabile alle righe e numeriamo sempre secondo il codice Gray

Adiacenze a 4 variabili: 4 uni

ab \ *cd*	00	01	11	10
00	1	1	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

b'c'

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	1	1	0	0
11	1	1	0	0
10	0	0	0	0

bc'

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

bd

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	1
10	0	0	1	1

ac

Adiacenze a 4 variabili: 4 uni

ab \ cd	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

$a'b'$

ab \ cd	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	0	0

$c'd$

ab \ cd	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	0	0

ab

ab \ cd	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$b'd'$

Adiacenze a 4 variabili: 8 uni

ab \ *cd*	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	0	0	0
10	0	0	0	0

a'

ab \ *cd*	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

d

ab \ *cd*	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

b'

ab \ *cd*	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

d'

Esempio

xy/zw	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

zw
xw
yw
y'w'
xy'
y'z

- **Identifichiamo gli implicanti primi**
  - Ci sono 6 implicanti primi, tutti da quattro uni
  - Ci sono molti implicanti da 2 uni, ma nessuno di essi è primo
  - Non ci sono implicanti da 8 uni
- **Troviamo quelli essenziali**
  - *y'w'* e *yw* sono essenziali
- **Me ne servono poi almeno altri due**
  - Uno tra *xy'* e *xw*
  - Uno tra *zw* e *y'z*
- **Per esempio**
  - $f = y'w' + yw + xw + zw$
  - $f = (y \oplus w)' + w(x + z)$

Da NON fare

xy/zw	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

- **<u>Metto implicanti NON primi</u>**
- **Per esempio**
  - $f = y'w' + yw + xy'w + x'y'zw$
- **Soluzione più costosa!!**
  - $xy'w$ è contenuto in xw o in xy'
  - $x'y'zw$ è contenuto in zw o in $y'z$
  - **Usate quello con meno letterali!!**
  - Non importa se si coprono degli 1 più volte

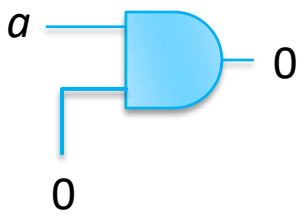
Esempio

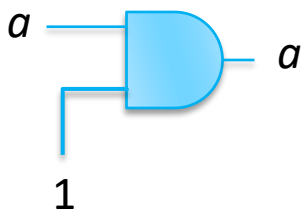
xy/zw	00	01	11	10
00	1	0	1	0
01	1	1	1	1
11	0	1	0	0
10	1	1	0	0

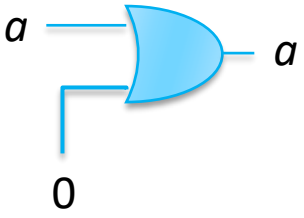
xz'w
yz'w
x'zw
x'y
y'z'w'
xy'z'
x'z'w'

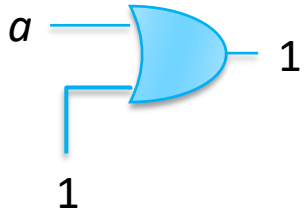
- **Identifichiamo gli implicanti primi**
  - Ci sono 6 implicanti primi da due uni
  - C'è un implicante da 4 uni
- **Troviamo quelli essenziali**
  - *x'zw* e *x'y* sono essenziali
- **Come copro i restanti uni?**
  - Posso usarne 3
  - Oppure solo 2
- **Ottengo**
  - $f = x'zw + x'y + xz'w + y'z'w'$
- **Due possibili insiemi non ridondanti**
  - Uno è più piccolo dell'altro
  - Un metodo è cercare di minimizzare le sovrapposizioni

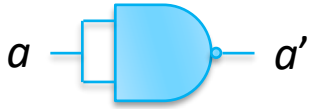
Uso delle “porte” logiche

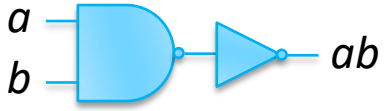


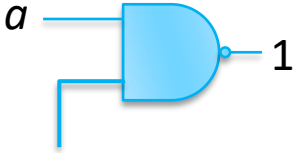


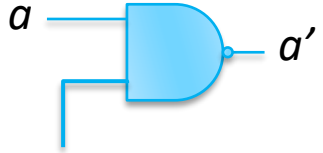


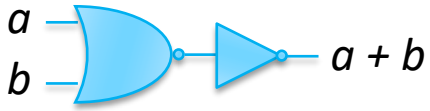


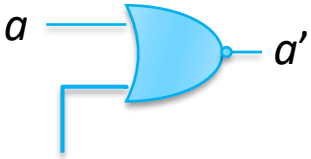


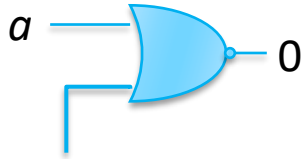


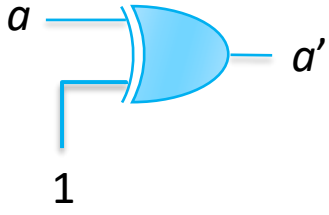


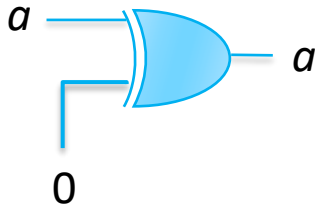


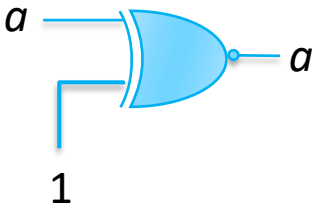


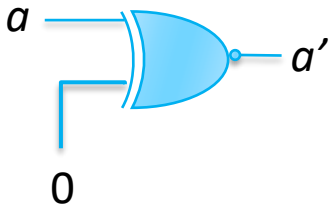












Insiemi di operatori completi

- **L'algebra di Boole è definita dai tre operatori base**
  - AND, OR e NOT
  - Con essi, per il teorema di Shannon, è possibile rappresentare *tutte* le funzioni booleane
- **Gli esempi precedenti mostrano che altri insiemi di operatori possono essere usati per definire l'algebra**
  - AND, NOT
  - OR, NOT
  - NAND
  - NOR
  - XOR, AND
  - XOR, OR
- **Si dicono insiemi di operatori *completi***
  - Gli insiemi di operatori completi sono sufficienti per realizzare tutte le funzioni logiche