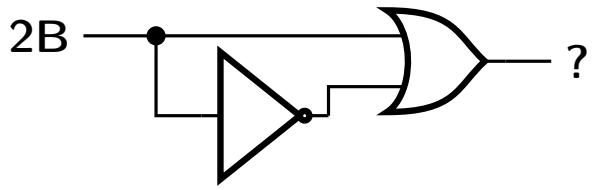
Tutorato di Informatica

Foglio 4 - Reti Logiche e Algebra Booleana

Matematici I Anno

26 Marzo 2026

"William Shakespeare, Computer Science Edition:"



This is the question.

Esercizi sulle Reti Logiche

Riscaldamento: Operazioni Fondamentali

Testo: Riordina e completa la Tabella 1:

Operazione	Not. Algebrica	Not. Logica	Not. Insiemistica
AND		$\dot{\vee}$	A^c
		$\wedge$	$\cap$
NOT	$+$		
XOR		$\neg$	Δ

Tabella 1: Tabella da completare

Soluzione:
Le operazioni fondamentali dell'algebra booleana hanno una corrispondenza diretta con gli operatori logici proposizionali e con le operazioni sugli insiemi. Di seguito la Tabella 2 completata:

Operazione	Not. Algebrica	Not. Logica	Not. Insiemistica
AND	$\cdot$	$\wedge$	$\cap$ (Intersezione)
OR	$+$	$\vee$	$\cup$ (Unione)
NOT	$\overline{X}$	$\neg$	A^c (Complemento)
XOR	$\oplus$	$\dot{\vee}$	Δ (Differenza Simmetrica)

Tabella 2: Soluzione: Tabella delle operazioni logiche

Esercizio 01: Espressioni Booleane

Testo: Semplificare le seguenti espressioni booleane indicando i passaggi eseguiti.

1. $\overline{X}(X + Y)$
2. $(\neg x \vee \neg y) \wedge (\neg x \vee y)$
3. $(x \cdot 1)(\overline{x} + y) + (y + 0)$
4. $\neg(\neg(A \vee B) \wedge (\neg B \vee C))$
5. $AB\overline{C}D + A(\overline{B \cdot C})D + \overline{A}BCD + A\overline{B}CD + \overline{A}BC\overline{D}$

Soluzione:

1. Ricordiamo che la moltiplicazione logica si distribuisce sull'addizione:

$$\begin{aligned} \overline{X}(X + Y) &= \overline{X}X + \overline{X}Y && \text{(Proprietà distributiva)} \\ &= 0 + \overline{X}Y && \text{(Assioma del complemento: } X \cdot \overline{X} = 0) \\ &= \overline{X}Y && \text{(Identità)} \end{aligned}$$

2. Anche in questo caso applichiamo la distributiva, raccogliendo il termine comune $\neg x$:

$$\begin{aligned} (\neg x \vee \neg y) \wedge (\neg x \vee y) &= \neg x \vee (\neg y \wedge y) && \text{(Proprietà distributiva inversa)} \\ &= \neg x \vee 0 && \text{(Assioma del complemento: } y \wedge \neg y = 0) \\ &= \neg x && \text{(Identità)} \end{aligned}$$

3. Semplifichiamo prima gli elementi neutri (identità):

$$\begin{aligned} (x \cdot 1)(\overline{x} + y) + (y + 0) &= x(\overline{x} + y) + y && \text{(Identità: } x \cdot 1 = x,\ y + 0 = y) \\ &= x\overline{x} + xy + y && \text{(Proprietà distributiva)} \\ &= 0 + xy + y && \text{(Assioma del complemento)} \\ &= y(x + 1) && \text{(Raccoglimento di } y) \\ &= y(1) = y && \text{(Assorbimento: } x + 1 = 1) \end{aligned}$$

4. Per gestire la negazione esterna, applichiamo i Teoremi di De Morgan ($\neg(X \wedge Y) = \neg X \vee \neg Y$):

$$\begin{aligned} \neg(\neg(A \vee B) \wedge (\neg B \vee C)) &= \neg(\neg(A \vee B)) \vee \neg(\neg B \vee C) && \text{(De Morgan)} \\ &= (A \vee B) \vee (B \wedge \neg C) && \text{(Doppia negazione e De Morgan)} \\ &= A \vee (B \vee (B \wedge \neg C)) && \text{(Associatività)} \\ &= A \vee B && \text{(Assorbimento: } B \vee (B \wedge \dots) = B) \end{aligned}$$

5. Applichiamo De Morgan per espandere il prodotto negato $\overline{B \cdot C}$:

$$\begin{aligned} &= AB\overline{C}D + A(\overline{B} + \overline{C})D + \overline{A}BC(D + \overline{D}) + A\overline{B}CD && \text{(De Morgan e raccoglimento su } \overline{A}BC) \\ &= AB\overline{C}D + AD(\overline{B} + \overline{C}) + \overline{A}BC(1) + A\overline{B}CD && \text{(Complemento: } D + \overline{D} = 1) \\ &= AB\overline{C}D + A\overline{B}D + A\overline{C}D + \overline{A}BC + A\overline{B}CD && \text{(Espansione)} \\ &= (AB\overline{C}D + A\overline{C}D) + (A\overline{B}D + A\overline{B}CD) + \overline{A}BC && \text{(Riordino dei termini)} \\ &= A\overline{C}D(B + 1) + A\overline{B}D(1 + C) + \overline{A}BC && \text{(Raccoglimento a fattor comune)} \\ &= A\overline{C}D + A\overline{B}D + \overline{A}BC && \text{(Assorbimento logico)} \end{aligned}$$

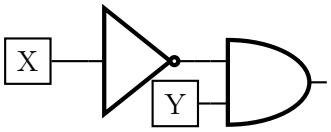
Esercizio 02: Traduzione in Circuiti Base

Testo: Disegnare i circuiti logici corrispondenti alle espressioni booleane dell'Esercizio 01. Si consiglia di utilizzare le formule semplificate ricavate nell'esercizio precedente per ottenere circuiti ottimali.

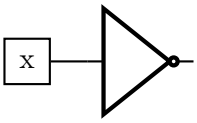
Soluzione:

Tradurre un'espressione booleana in un circuito significa mappare gli operatori alle relative porte logiche. Avendo semplificato le formule, i circuiti risulteranno composti dal minor numero di porte possibili.

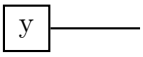
1. **Espressione semplificata:** $\overline{X}Y$ (Richiede un NOT su X e un AND tra i segnali)



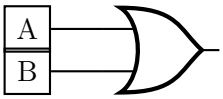
2. **Espressione semplificata:** $\neg x$ (Richiede solo una porta NOT)



3. **Espressione semplificata:** y (Nessuna porta, è un semplice collegamento)



4. **Espressione semplificata:** $A \vee B$ (Richiede una singola porta OR)



5. **Espressione semplificata:** $A\overline{C}D + A\overline{B}D + \overline{A}BC$

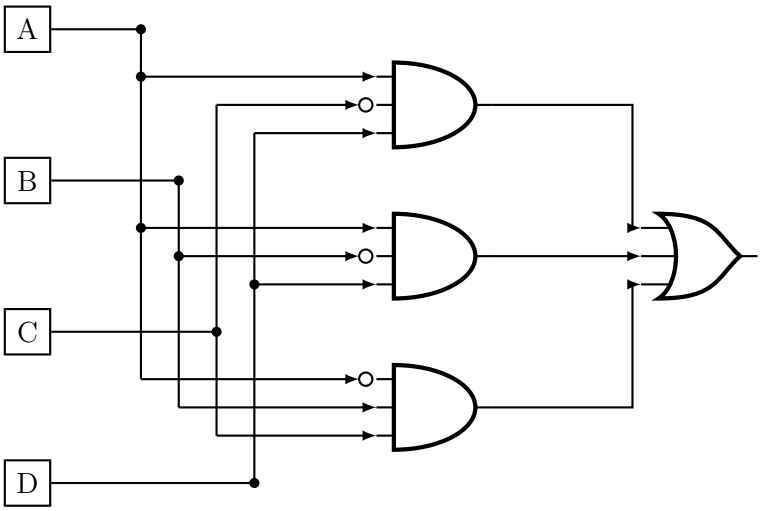


Figura 1: Implementazione dei circuiti minimizzati

Esercizio 03: Analisi di una Rete Logica con Retroazione

Testo: Dato il diagramma in Figura 2:

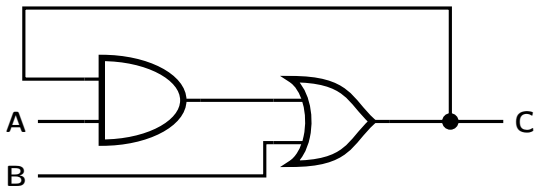


Figura 2: Rete logica con loop di retroazione

Determinare quali possibili configurazioni dei canali binari A, B e C sono ammissibili (ovvero, quali configurazioni possono effettivamente verificarsi).

Soluzione:
Siamo di fronte a un circuito sequenziale (retroazione). Per trovare le configurazioni stabili, scriviamo l'equazione algebrica del circuito. L'uscita C è il risultato di un OR (+) tra il segnale B e l'uscita della porta AND (prodotto tra A e C):

$$C = AC + B$$

Una configurazione (A, B, C) è ammissibile solo se rispetta l'uguaglianza. Verifichiamo gli stati:

- Se $(0,0,0) \implies C = 0 \cdot 0 + 0 = 0$. **Ammissibile**.
- Se $(0,0,1) \implies C = 0 \cdot 1 + 0 = 0$. **Non ammissibile** $(0 \neq 1)$.
- Se $(0,1,0) \implies C = 0 \cdot 0 + 1 = 1$. **Non ammissibile** $(1 \neq 0)$.
- Se $(0,1,1) \implies C = 0 \cdot 1 + 1 = 1$. **Ammissibile**.
- Se $(1,0,0) \implies C = 1 \cdot 0 + 0 = 0$. **Ammissibile**.
- Se $(1,0,1) \implies C = 1 \cdot 1 + 0 = 1$. **Ammissibile**.
- Se $(1,1,0) \implies C = 1 \cdot 0 + 1 = 1$. **Non ammissibile** $(1 \neq 0)$.
- Se $(1,1,1) \implies C = 1 \cdot 1 + 1 = 1$. **Ammissibile**.

In conclusione, le configurazioni **001, 010 e 110** sono instabili.

Esercizio 04: Derivare la Tabella di Verità

Testo: Determinare la tabella di verità del circuito logico in Figura 3. Cosa rappresenta il circuito proposto?

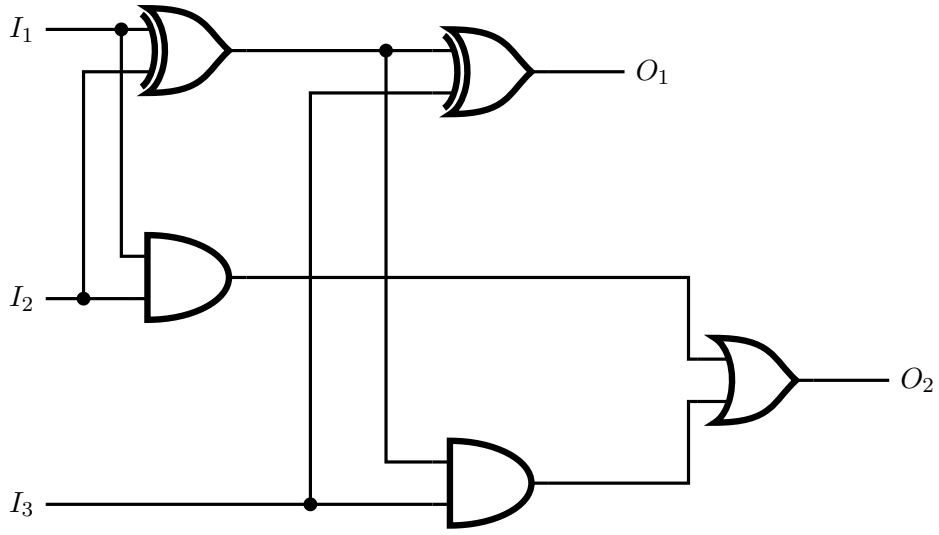


Figura 3: Circuito logico per l'Esercizio 04

Soluzione:
Per ricavare in modo sistematico la tabella di verità, dividiamo il circuito in due fasi, analizzando prima i segnali intermedi:

- Indichiamo con X_1 l'uscita della prima porta XOR: $X_1 = I_1 \oplus I_2$.
- Indichiamo con A_1 l'uscita della prima porta AND: $A_1 = I_1 I_2$.

Ora analizziamo lo stadio finale che genera le uscite:

- O_1 è generato da una seconda porta XOR che accetta X_1 e I_3. Quindi: $O_1 = X_1 \oplus I_3 = I_1 \oplus I_2 \oplus I_3$.
- O_2 è l'uscita di una porta OR che somma A_1 e l'uscita della seconda porta AND ($X_1 I_3$). Quindi: $O_2 = I_1 I_2 + (I_1 \oplus I_2) I_3$.

Costruendo la Tabella 3 valutando queste formule per ogni combinazione di ingressi, otteniamo:

I_1	I_2	I_3	O_1	O_2
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
0	0	1	1	0
1	1	0	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Tabella 3: Tabella di verità del Full Adder

Osservando i risultati, notiamo che O_1 conta le "unità" dei bit accesi e O_2 funge da "riporto". Questo circuito è infatti il noto **Sommatore Completo (Full Adder)** a 1 bit.

Esercizio 05: Diagramma del Decoder

Testo: Disegnare il diagramma di un decoder a 3 bit e determinare la sua tabella di verità. Qual è la rappresentazione "one-hot" della stringa binaria 110 calcolata dal decoder?

Soluzione:
Un decoder è un circuito combinatorio che converte un codice binario in ingresso di N bit in una singola linea attiva in uscita tra 2^N linee totali. Utilizzando 3 bit (I_0, I_1, I_2), avremo $2^3 = 8$ possibili combinazioni. Per questo necessitiamo di porte NOT (per ricavare le versioni negate dei segnali di ingresso) e di 8 porte AND a tre ingressi (una per ogni mintermine o configurazione unica).

Il diagramma in Figura 4 illustra questo concetto smistando le combinazioni dritte e negate agli ingressi delle rispettive porte AND.

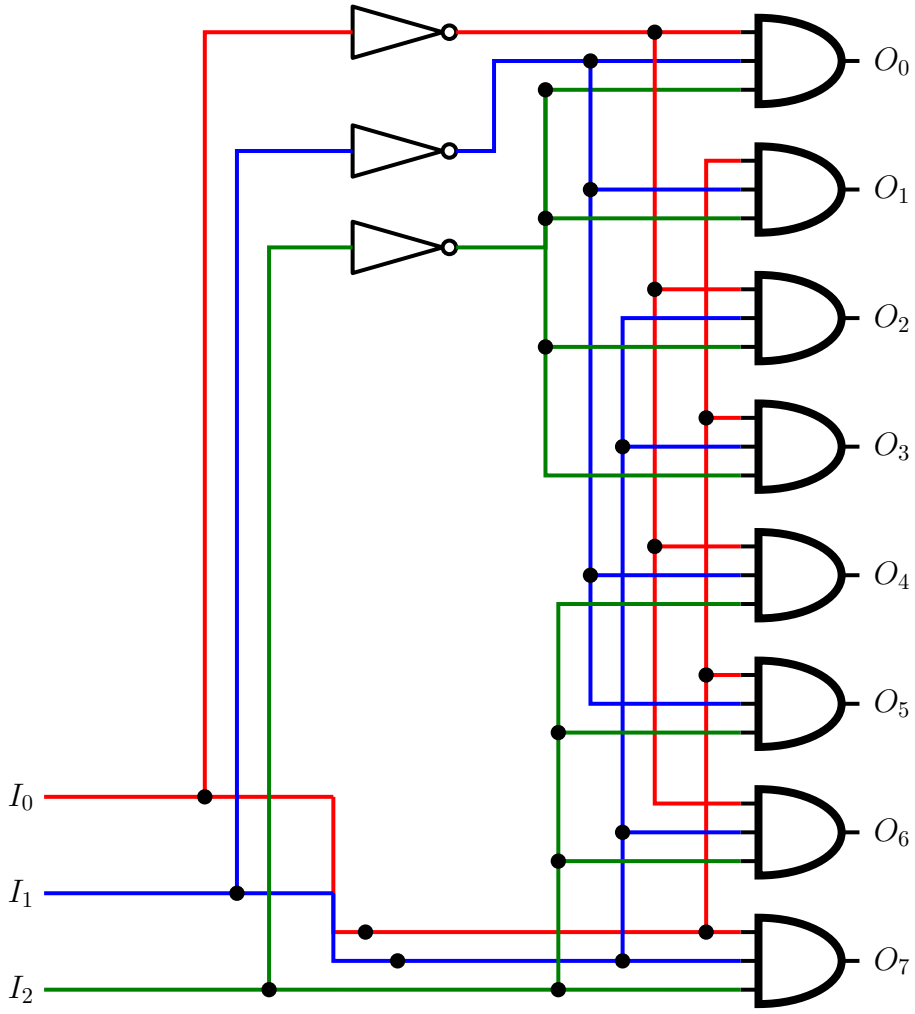


Figura 4: Diagramma del decoder a 3 bit

Per trovare la rappresentazione "one-hot" (dove un solo bit dell'array è acceso) convertiamo la stringa binaria d'ingresso in base decimale: l'input 110_2 corrisponde al numero decimale 6. Questo significa che l'output del decoder attiverà esclusivamente la linea O_6. Tradotto in una "word" a 8-bit, in cui solo il bit in posizione 6 (partendo da destra e contando da indice 0) è 1, si ottiene la configurazione **0100 0000**. La Tabella 4 riassume il comportamento completo.

I_0	I_1	I_2	$O_7, ..., O_0$
0	0	0	0000 0001
0	0	1	0000 0010
0	1	0	0000 0100
0	1	1	0000 1000
1	0	0	0001 0000
1	0	1	0010 0000
1	1	0	0100 0000
1	1	1	1000 0000

Tabella 4: Tabella di verità del decoder

Esercizio 06: Progettazione a Partire dalla Tabella di Verità

Testo: Disegnare un circuito con 3 input (A, B, C) e due output (O_1, O_2) che realizza la tabella di verità indicata in Tabella 5:

A	B	C	O_1	O_2
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	1

Tabella 5: Tabella di verità da realizzare

Soluzione:
Per ricavare il circuito ottimiziamo le funzioni di uscita partendo dalla somma dei mintermini (ovvero unendo in OR tutti i casi in cui l'uscita desiderata vale 1).

Semplificazione per O_1: Individuiamo le righe in cui $O_1 = 1$: $(0,0,0), (0,1,1), (1,0,1), (1,1,0)$.

$$\begin{aligned} O_1 &= \overline{A}\overline{B}\overline{C} + \overline{A}BC + A\overline{B}C + AB\overline{C} \\ &= \overline{A}(\overline{B}\overline{C} + BC) + A(\overline{B}C + B\overline{C}) \qquad \text{(Raccoglimento di } \overline{A} \text{ e } A) \end{aligned}$$

Ricordiamo la definizione delle porte esclusive: $(B \oplus C) = \overline{B}C + B\overline{C}$ e la sua negazione $\overline{B \oplus C} = \overline{B}\overline{C} + BC$. Sostituendo si ha:

$$O_1 = \overline{A}(\overline{B \oplus C}) + A(B \oplus C)$$

Se consideriamo la variabile fittizia $X = (B \oplus C)$, l'espressione è $\overline{A}\overline{X} + AX$, che è l'operazione XNOR. Quindi:

$$O_1 = \overline{A \oplus (B \oplus C)} = \overline{A \oplus B \oplus C}$$

Semplificazione per O_2: Individuiamo le righe in cui $O_2 = 1$: $(1,0,1)$ e $(1,1,1)$.

$$\begin{aligned} O_2 &= A\overline{B}C + ABC \\ &= AC(\overline{B} + B) \qquad \text{(Raccoglimento parziale di } AC) \\ &= AC(1) \qquad \text{(Complemento)} \\ &= AC \end{aligned}$$

Grazie a queste drastiche riduzioni algebriche, il circuito risultante si traduce in un design pulito ed elegantissimo, richiedendo solo un AND, un XOR e un XNOR (Figura 5).

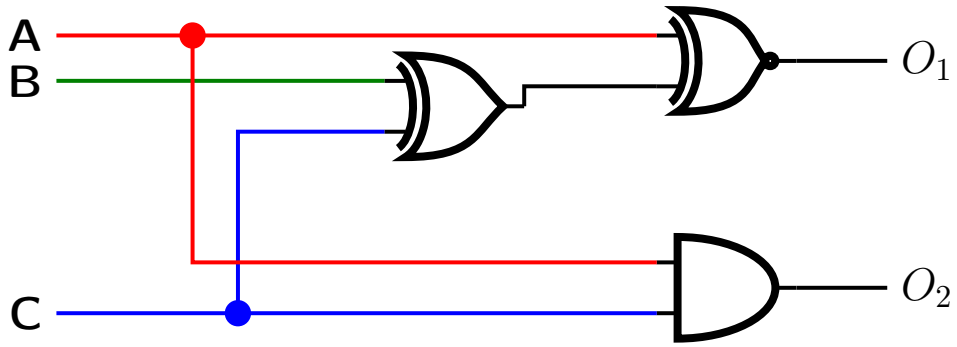


Figura 5: Circuito logico derivato per l'Esercizio 06

Esercizi Extra Per i Più Coraggiosi (Lab 4)

01. Il Rilevatore di Numeri Primi e la Mappa del Tesoro

Testo: Siete stati assunti dalla *Prime Directive Corp.* Il vostro compito è progettare un circuito combinatorio a 4 bit per blindare un caveau. Il sistema si sblocca solo ed esclusivamente se il codice binario inserito in ingresso (A, B, C, D) corrisponde a un **numero primo** nell'intervallo da 0 a 15. Inserite un codice sbagliato, e scatterà l'allarme. Ricavate l'espressione logica minima del circuito.

Suggerimento: Ricordate che 0 e 1 non sono numeri primi. Prima di perdervi in un mare di algebra booleana, stilate l'elenco dei numeri primi, convertiteli in binario e osservate i mintermini. Se conoscete le Mappe di Karnaugh, questo è il momento perfetto per disegnarne una: la disposizione spaziale dei "veri" vi suggerirà come raggrupparli a colpo d'occhio.

Soluzione:

Passo 1: L'intuizione matematica e i Mintermini I numeri primi strettamente compresi tra 0 e 15 (usando un codice a 4 bit) sono solamente sei: 2, 3, 5, 7, 11 e 13. Convertendoli in binario, otteniamo i nostri 6 casi "vincenti" in cui l'uscita del circuito deve valere 1:

- $2_{10} = 0010_2 \implies \overline{A}B\overline{C}\overline{D}$
- $3_{10} = 0011_2 \implies \overline{A}B\overline{C}D$
- $5_{10} = 0101_2 \implies \overline{A}B\overline{C}D$
- $7_{10} = 0111_2 \implies \overline{A}BCD$
- $11_{10} = 1011_2 \implies A\overline{B}CD$
- $13_{10} = 1101_2 \implies AB\overline{C}D$

Passo 2: La forma canonica (Somma di Prodotti) Unendo tutti i mintermini con l'operatore OR, otteniamo l'espressione logica completa che risolve il problema:

$$O = \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}CD + AB\overline{C}D$$

Costruire fisicamente questa funzione richiederebbe ben 6 porte AND a 4 ingressi. Un vero incubo ingegneristico! Dobbiamo semplificare.

Passo 3: Il Segreto Informatico (L'Ottimizzazione Algebrica) Per semplificare, dobbiamo cercare coppie di termini che differiscono per *una sola* variabile. Qui entra in gioco un trucco potentissimo dell'algebra di Boole: la **legge dell'idempotenza** ($X + X = X$). Questa legge ci permette di "duplicare" un mintermine utile per poterlo accoppiare con più vicini diversi.

Nel nostro caso, i mintermini corrispondenti al 3 ($\overline{A}\overline{B}CD$) e al 5 ($\overline{A}B\overline{C}D$) sono molto comodi. Li duplichiamo nella nostra equazione:

$$\begin{aligned} O = {} & \overline{A}\overline{B}C\overline{D} \\ & + (\overline{A}\overline{B}CD + \overline{A}\overline{B}CD) && \text{(Duplichiamo il 3)} \\ & + (\overline{A}B\overline{C}D + \overline{A}B\overline{C}D) && \text{(Duplichiamo il 5)} \\ & + \overline{A}BCD + A\overline{B}CD + AB\overline{C}D \end{aligned}$$

Ora riordiniamo ad arte i termini per creare **quattro coppie perfette**, dove in ciascuna cambia una sola lettera:

$$\begin{aligned} O = {} & (\overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD) && \text{(Coppia 2 e 3)} \\ & + (\overline{A}B\overline{C}D + \overline{A}BCD) && \text{(Coppia 5 e 7)} \\ & + (\overline{A}\overline{B}CD + A\overline{B}CD) && \text{(Coppia 3 e 11, sfruttando il 3 duplicato)} \\ & + (\overline{A}B\overline{C}D + AB\overline{C}D) && \text{(Coppia 5 e 13, sfruttando il 5 duplicato)} \end{aligned}$$

Passo 4: Il Raccoglimento Finale Ora non ci resta che applicare il raccoglimento a fattor comune per ogni parentesi. Ricordando che la somma di una variabile con il suo opposto vale sempre 1 ($X + \overline{X} = 1$), i termini ridondanti spariscono:

$$\begin{aligned} O = {} & \overline{A}\overline{B}C(\overline{D} + D) \\ & + \overline{A}BD(\overline{C} + C) \\ & + \overline{B}CD(\overline{A} + A) \\ & + B\overline{C}D(\overline{A} + A) \end{aligned}$$

L'espressione collassa definitivamente nel nostro circuito compatto ed elegante:

$$O = \overline{A}\overline{B}C \vee \overline{A}BD \vee \overline{B}CD \vee B\overline{C}D$$

02. Il Lupo, la Capra, il Cavolo e l'Allarme Hardware

Testo: Un contadino deve trasportare un lupo, una capra e un cavolo oltre un fiume. Conoscete tutti il classico indovinello: se lasciati soli sulla stessa sponda, il lupo mangia la capra, e la capra mangia il cavolo. Vogliamo costruire un **allarme hardware** che suoni istantaneamente se il contadino compie una mossa letale.

Siano C, L, P, V quattro variabili booleane che valgono 1 se il rispettivo soggetto (Contadino, Lupo, Pecora/Capra, Verza/Cavolo) si trova sulla sponda sinistra, e 0 se si trova sulla sponda destra. Scrivere l'equazione logica dell'allarme A.

Suggerimento: Vietato scrivere un'enorme tabella della verità a 16 righe! Usate la logica pura. Quando due entità sono sulla *stessa* sponda, le loro variabili sono *uguali* (entrambe a 0 o entrambe a 1). Quale porta logica restituisce 1 quando i suoi due input sono identici? E quale se sono diversi?

Soluzione:

Passo 1: Il setup e l'approccio classico (da evitare) Iniziamo definendo le variabili: C, L, P, V valgono 1 se il rispettivo soggetto si trova sulla sponda sinistra, e 0 se si trova sulla destra. L'approccio "forza bruta" richiederebbe la costruzione di una tabella di verità con ben 16 combinazioni (2^4). Dovremmo cercare le righe in cui si verifica un disastro e scrivere una lunghissima somma di prodotti (mintermini). Invece di fare questo lavoro meccanico, tradurremo direttamente il linguaggio naturale in logica elegante!

Passo 2: Le condizioni letali (L'operatore XNOR) Il primo problema si verifica quando il Lupo e la Pecora si trovano sulla *stessa* sponda. Questo significa che le loro variabili devono avere lo stesso valore: o sono entrambe a sinistra ($L = 1$ AND $P = 1$) oppure sono entrambe a destra ($L = 0$ AND $P = 0$). Tradotto in algebra booleana: $(L \cdot P) + (\overline{L} \cdot \overline{P})$. Questa espressione è la definizione esatta dell'operazione di **Equivalenza Logica**, ovvero la porta **XNOR** ($\odot$). Quindi, il segnale di pericolo per Lupo e Pecora si riassume in: $(L \odot P)$.

Seguendo l'identico ragionamento, la Pecora e il Cavolo si trovano sulla stessa sponda (e quindi scatta il pericolo) quando le loro variabili coincidono: $(P \odot V)$.

Passo 3: Il Salvatore (L'operatore XOR) Tuttavia, queste "vicinanze" letali si trasformano in un pasto *solo se* il Contadino (C) non è presente a sorvegliare la Pecora (P). Il disastro accade **MENTRE** (operatore AND) il contadino è sulla sponda *opposta* rispetto alla pecora. Ciò significa che le loro variabili sono diverse: se il contadino è a sinistra la pecora è a destra ($C = 1, P = 0$), o viceversa ($C = 0, P = 1$). Algebricamente si scrive: $(C \cdot \overline{P}) + (\overline{C} \cdot P)$. Questa è la definizione assoluta dell'operatore **OR Esclusivo (XOR)**, indicato col simbolo $\oplus$. La condizione "il contadino è lontano dalla pecora" diventa quindi: $(C \oplus P)$.

Passo 4: Il Segreto Informatico (L'Equazione Magica e la Distributiva) Ora possiamo assemblare l'intero allarme. Il circuito deve suonare in due casi distinti:

1. C'è il pericolo Lupo-Pecora **E** il contadino è lontano: $(L \odot P) \cdot (C \oplus P)$
2. **OPPURE** c'è il pericolo Pecora-Cavolo **E** il contadino è lontano: $(P \odot V) \cdot (C \oplus P)$

Unendo questi due casi con una porta OR (+), otteniamo l'equazione completa:

$$A = [(L \odot P) \cdot (C \oplus P)] + [(P \odot V) \cdot (C \oplus P)]$$

A questo punto, per ridurre il numero di porte logiche da comprare per il nostro circuito, sfruttiamo la **proprietà distributiva** dell'algebra di Boole. Proprio come nell'algebra classica, notiamo che il termine $(C \oplus P)$ è in comune a entrambi i blocchi. Lo raccogliamo a fattor comune, ottenendo un design circuitale brillantemente compatto:

$$A = (C \oplus P) \cdot [(L \odot P) + (P \odot V)]$$

03. La Regola 30: Dall'Algebra Booleana al Caos Deterministico

Testo: Siete biologi computazionali e state studiando gli *Automi Cellulari*, griglie matematiche in cui ogni "cellula" decide se vivere o morire al "giorno" successivo basandosi esclusivamente sul proprio stato attuale e su quello dei suoi due vicini immediati.

Nel 1983, il matematico Stephen Wolfram scoprì un automa unidimensionale, chiamato **Regola 30**, in cui una cella al centro (C) osserva il vicino di sinistra (L) e di destra (R). Sebbene le regole siano fisse e deterministiche, il pattern generato nel tempo è talmente caotico, aperiodico e statisticamente perfetto da essere utilizzato nei sistemi informatici come generatore di **numeri pseudo-casuali** crittografici. La natura lo ha persino implementato fisicamente: il guscio della lumaca marina *Conus textile* sfoggia un pattern molto simile alla Regola 30.

Figura 6: Pattern del guscio della Conus textile.

La tabella che governa la nascita (1) o la morte (0) della cellula al tempo successivo è la seguente:

L (Sinistra)	C (Centro)	R (Destra)	Nuovo Stato (O)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Tabella 6: Regola 30 di Wolfram

Ricavate ed estremamente semplificate l'espressione logica che governa questo intero universo microscopico.

Suggerimento: La tabella vi dà 4 mintermini. Scrivete la Somma di Prodotti. Notate qualcosa in comune nei primi tre termini? Provate a raccogliere $\overline{L}$. Ricordatevi anche del "Teorema di De Morgan" per trattare l'ultimo termine.

Soluzione:

Passo 1: I Mintermini e l'Estrazione dalla Tabella Analizzando la Tabella di Verità, identifichiamo le quattro righe in cui l'uscita O (Nuovo Stato) è pari a 1:

- 001 $\implies \overline{L}\overline{C}R$
- 010 $\implies \overline{L}C\overline{R}$
- 011 $\implies \overline{L}CR$
- 100 $\implies L\overline{C}\overline{R}$

Sommando questi mintermini, otteniamo l'equazione canonica:

$$O = \overline{L}\overline{C}R + \overline{L}C\overline{R} + \overline{L}CR + L\overline{C}\overline{R}$$

Passo 2: Il Raccoglimento a Fattor Comune Osservando i primi tre termini, notiamo che la variabile L è sempre negata ($\overline{L}$). Possiamo quindi raccoglierla a fattor comune:

$$O = \overline{L} \cdot (\overline{C}R + C\overline{R} + CR) + L\overline{C}\overline{R}$$

Passo 3: L'Analisi della Parentesi (La nascita dell'OR) L'espressione $(\overline{C}R+C\overline{R}+CR)$ copre tutti i casi in cui almeno una tra C e R è accesa. Algebricamente, possiamo semplificarla raccogliendo C dagli ultimi due termini: $\overline{C}R+C(\overline{R}+R) = \overline{C}R+C$. Per la legge dell'assorbimento, questa si riduce alla porta **OR**: $(C + R)$. L'equazione diventa:

$$O = \overline{L}(C + R) + L\overline{CR}$$

Passo 4: Il trucco di De Morgan Il termine isolato $L\overline{CR}$ contiene il prodotto di due negazioni. Applicando il **Teorema di De Morgan**, sappiamo che $\overline{C}\cdot\overline{R} = \overline{(C + R)}$. Sostituendo:

$$O = \overline{L}(C + R) + L\overline{(C + R)}$$

Passo 5: Il Segreto Informatico (Il Caos nello XOR) Se consideriamo il blocco $(C+R)$ come un'unica variabile, l'espressione ha la forma $\overline{A}B + A\overline{B}$, che è la definizione esatta dell'OR esclusivo (**XOR**):

$$O = L \oplus (C + R)$$

Analisi del Risultato ed Evoluzione Temporale L'immagine in Figura 7 mostra l'esecuzione iterata di questa semplicissima equazione. Partendo da un singolo bit acceso, ogni riga successiva viene calcolata applicando $O = L \oplus (C \vee R)$.

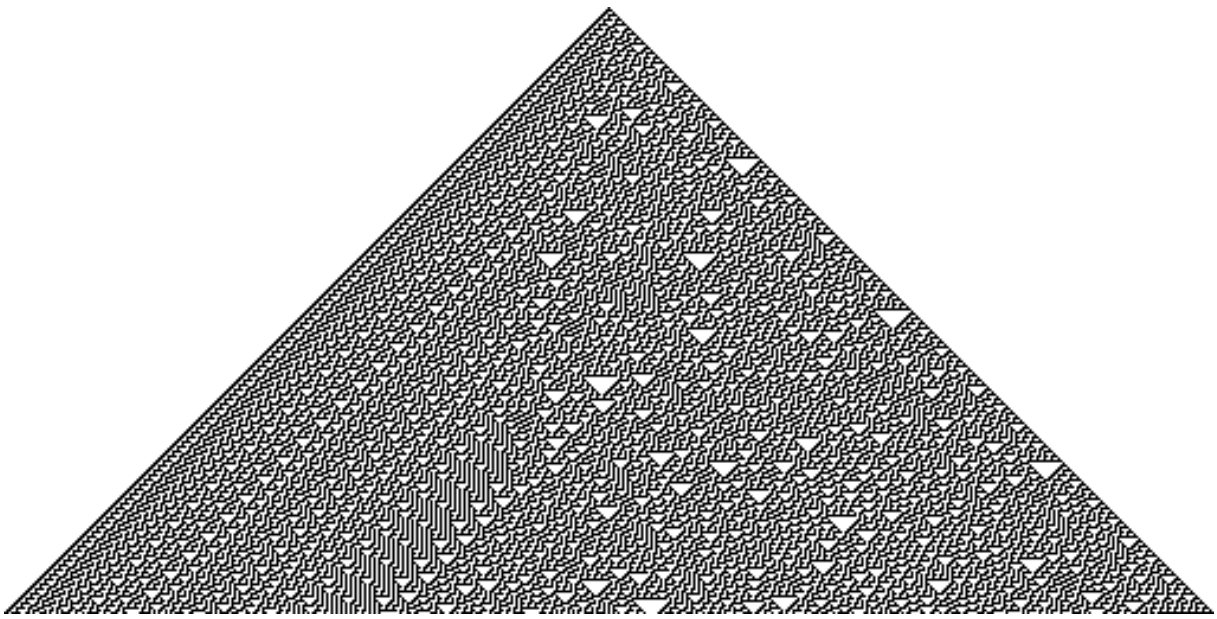


Figura 7: Evoluzione temporale della Rule 30 (256 generazioni)

Siamo di fronte a un esempio di **irriducibilità computazionale**: nonostante la regola sia deterministica e semplicissima, il pattern generato (specialmente al centro e a destra) è aperiodico e caotico. Non esiste una scorciatoia matematica per prevedere lo stato di una cella nel futuro senza far "girare" fisicamente il circuito riga dopo riga. Questa imprevedibilità è il motivo per cui la Regola 30 è una delle basi per la generazione di numeri casuali nei computer moderni.