Università degli Studi di Milano
Facoltà di Scienze e Tecnologie

Corso di Laurea in
Sicurezza dei Sistemi e delle Reti Informatiche

FUNZIONI PRIVATAMENTE COMPUTABILI IN LOGICA MULTIVALORE

Relatore: Prof.ssa Valentina Ciriani

Correlatore: Prof. Stelvio Cimato

Tesi di Laurea di:
Mauro Galimberti
Matricola: 943278

Anno Accademico 2022/2023

Indice

Elenco delle figure

Elenco degli pseudocodici

Capitolo 1

Introduzione

Gli indubbi vantaggi offerti dalla diffusione delle tecnologie digitali si accompagnano a nuove sfide per la difesa della privacy e delle libertà fondamentali dell'individuo. Hardware e software di ogni genere vengono utilizzati dall'utente medio nella convinzione, data quasi per scontata, che tali strumenti consentano l'utilizzo dei servizi dell'infrastruttura informatica svolgendo esclusivamente i compiti per cui sono stati progettati e nulla più. Sfortunatamente non è sempre così: come testimoniano recenti attività di *leaking* e *whistleblowing* [20], una riflessione più attenta su come proteggere con efficacia i propri diritti fondamentali appare quantomai doverosa.

Abituarsi a considerare un qualunque sistema informatico come un ambiente sempre dotato di un certo grado di insicurezza stimola la ricerca di soluzioni alternative in campi di studio ancora poco battuti. Può, ad esempio, spingere a sviluppare nuovi strumenti e tecnologie concepite appositamente per effettuare computazioni garantite come private, anche all'interno di scenari basati su infrastrutture poco affidabili.

La crittografia visuale è un esempio di tecnica che ben si adatta ad un utilizzo in tali scenari. Permette di decifrare informazioni segrete anche senza l'ausilio di mezzi digitali, basandosi unicamente sulla capacità visiva umana. Implementare tale tecnica in sistemi crittografici in uso nella comunicazione tra due parti attenuerebbe il problema della presenza di potenziali parti non fidate e/o malevole, garantendo un maggior grado di sicurezza e privatezza delle informazioni scambiate.

Nella comunicazione tra più parti vengono inoltre introdotti particolari schemi di *secret sharing* (utilizzabili anche in applicazioni di crittografia visuale) che sfruttano la particolare proprietà di basarsi su funzioni omomorfiche, concetto di cui si darà un'opportuna definizione formale.

Come in ogni campo della crittografia moderna, prima di arrivare ad ottenere un prodotto finito ed utilizzabile nei più vari contesti applicativi, occorrerà predisporre una sufficiente base teorica, supportata da adeguati strumenti matematici.

Tra questi uno dei più importanti è rappresentato da una particolare categoria di funzioni, denominate *privatamente computabili*. È proprio a questo tipo di oggetti matematici che il presente elaborato dedica uno studio approfondito, partendo da alcune definizioni e teoremi già noti in letteratura scientifica [6, 13] e considerando i recenti risultati sperimentali esposti in altre tesi di laurea che hanno affrontato l'argomento [1, 10, 14].

Viene avviata un'analisi teorica e sperimentale, consistente nello sviluppo di opportuni algoritmi e nel successivo *coding* di programmi efficienti, creati allo scopo di produrre output che consentano di rispondere a svariati quesiti legati sia alla quantità di funzioni privatamente computabili presenti in insiemi di funzioni in logica multivalore, sia alla loro specifica natura.

Il lavoro svolto è orientato allo studio di funzioni in logica ternaria, un insieme ancora poco esplorato in bibliografia, con l'obiettivo di determinare anzitutto il numero delle funzioni privatamente computabili all'interno dell'insieme richiamato. A tale scopo si è sviluppato ogni passaggio basandosi su definizioni, criteri e teoremi già noti e dimostrati, in modo da supportare adeguatamente la correttezza dei dati ottenuti.

Si è in seguito provveduto a stampare le funzioni privatamente computabili in un output opportunamente ordinato, con l'obiettivo di trovare una modalità per avviare un'analisi qualitativa organizzata ed efficace sulle singole funzioni e su gruppi di funzioni in logica ternaria. Tale obiettivo andava raggiunto sviluppando in autonomia strumenti e metodologie originali, considerando che in letteratura scientifica questo tipo di analisi non era stata ancora affrontata ad un simile livello di dettaglio.

Si è infine valutato in quale misura il lavoro svolto poteva essere adattato ed esteso ad insiemi di funzioni in logiche multivalore superiori a 3.

1.1 Organizzazione della tesi

Il presente elaborato è organizzato come segue:

- il capitolo 2 riassume lo stato dell'arte della materia presa in oggetto e presenta la base teorica necessaria per la successiva analisi sperimentale. Verranno introdotti due possibili contesti applicativi: la tecnica della crittografia visuale e la comunicazione tra due parti sfruttando modelli di *secret sharing homomorphism*.

  Si esporranno dunque i concetti fondamentali legati al tema delle funzioni privatamente computabili (argomento centrale di questa tesi) e alcuni risultati notevoli già raggiunti per funzioni in logica booleana e per funzioni in logica ternaria. Costituiranno fondamentali punti di partenza per l'analisi che verrà condotta nei capitoli successivi;

- il capitolo 3 sarà dedicato allo studio approfondito delle funzioni in logica ternaria. Una prima analisi di tipo quantitativo porterà a sviluppare algoritmi atti a determinare con esattezza quante delle funzioni presenti in tale insieme godano della fondamentale proprietà di essere privatamente computabili.

  Un'analisi qualitativa ancor più raffinata sul sottoinsieme delle funzioni privatamente computabili individuate porterà a far emergere un limitato numero di schemi ricorrenti (*pattern*), raccolti in modo da costituire un rapido classificatore di funzioni ternarie privatamente computabili;

- nel capitolo 4 si esporranno i risultati delle analisi sviluppate al capitolo precedente. Si ragionerà su come sia possibile aumentare l'efficienza del programma sviluppato e si concluderà la caratterizzazione dei pattern già individuati alla fine del capitolo 3. Si discuterà infine un possibile uso pratico di tali pattern, impiegati quali classificatori di funzioni in logica ternaria;

- nel capitolo 5 si trarranno le conclusioni relative alla parte sperimentale sviluppata nell'elaborato. Verranno inoltre indicate diverse possibilità di sviluppi di ricerca futuri.

Alcune appendici sono state raccolte al termine della tesi. Ciascuna riporta il dettaglio di un passaggio ritenuto importante nel procedimento sperimentale seguito. Trattandosi di output troppo corposi per essere inseriti nel flusso principale della trattazione senza ostacolarne la consultazione, si è preferito raccogliere tali risultati al termine dell'elaborato. Per ciascuna appendice verranno fornite le necessarie istruzioni d'uso al momento di esporre l'argomento cui sono direttamente collegate.

Capitolo 2

Stato dell'arte

Il capitolo presenta in modo sintetico definizioni e concetti chiave di quanto attualmente noto in bibliografia in merito ad una particolare categoria di funzioni, denominate *privatamente computabili*. Tali funzioni sono oggetto di numerosi studi in quanto impiegabili in svariati contesti applicativi meritevoli di particolare attenzione: tecniche come la crittografia visuale o le comunicazioni tra due parti sfruttando modelli di *secret sharing homomorphism* garantiscono un maggior livello di sicurezza e privatezza delle informazioni scambiate, specie in infrastrutture che dimostrano un certo grado di inaffidabilità. Si vedrà come le corretta implementazione di tali tecniche non possa prescindere dall'uso di funzioni privatamente computabili, dotate di peculiari caratteristiche e proprietà che verrano discusse in dettaglio a partire dalla sezione 2.3.

2.1 Crittografia visuale

Un'immagine può essere utilizzata per comunicare informazioni di vario tipo, compresi schemi o testi segreti. La crittografia visuale si occupa di studiare le tecniche che consentono di scomporre l'immagine di partenza in più componenti irriconoscibili. Tali componenti (in letteratura noti come *share*) si riveleranno del tutto inservibili agli occhi di un potenziale utente malevolo che si trovasse nella condizione di intercettarne una parte non sufficiente a ricostruire il segreto. Saranno invece utilizzate in numero sufficiente dal destinatario per visualizzare l'immagine segreta di partenza per mezzo di un'operazione di ricostruzione tramite sovrapposizione delle share. L'aspetto interessante è che l'intero procedimento può essere condotto senza l'ausilio di strumenti esterni, siano essi hardware o software.

Il modello più semplice di crittografia visuale [12] prevede che sia il testo cifrato che la chiave siano stampati su due fogli trasparenti distinti: entrambi i lucidi sono griglie della stessa dimensione composte dallo stesso numero di pixel, ciascuno dei quali può essere trasparente oppure opaco. I colori dei pixel del lucido rappresentante la chiave

vengono scelti casualmente, senza che vi sia correlazione tra di essi. A partire dalla chiave viene generato il lucido rappresentante il messaggio cifrato. L'informazione in chiaro si ottiene sovrapponendo esattamente quest'ultimo lucido a quello contente la chiave, nonostante entrambi appaiano come un insieme di pixel casuali.

Il successivo contributo di Naor e Shamir [15] propone l'utilizzo di schemi di secret sharing *(k, n) - Threshold*. Tali schemi a soglia prevedono che, data un'immagine, si generino n lucidi in totale. L'immagine di partenza sarà visibile a patto di sovrapporre almeno k degli n lucidi prodotti (con $k < n$). Nessuna informazione di alcun tipo potrà essere dedotta sovrapponendo un numero di lucidi minore di k.

Si supponga di confrontarsi con un contesto reale nel quale si ha la necessità di distribuire un'immagine, contenente un segreto, ad un gruppo di partecipanti. Una parte fidata (un *dealer*) conosce il segreto e genera le *share*, una per ogni partecipante. Tra l'intero insieme dei partecipanti vi saranno alcuni sottogruppi autorizzati a ricostruire il segreto, raggruppando e sovrapponendo le loro share. Tali sottogruppi sono detti *qualificati* e indicati con la lettera $\mathcal{Q}$. Esistono poi altri sottogruppi di partecipanti, detti *proibiti* e indicati con la lettera $\mathcal{P}$, che non sono in grado di recuperare alcuna informazione a partire dalle loro share. La peculiarità più interessante della crittografia visuale è che la ricostruzione del segreto viene eseguita senza il supporto di alcuna computazione: un insieme di partecipanti qualificati dovrà semplicemente sovrapporre le proprie share (vale a dire i propri lucidi, opportunamente creati dal dealer) per mostrare il segreto. Viceversa insiemi di partecipanti proibiti non ricaveranno alcuna informazione dalle share, né sovrapponendole, né tramite altra computazione.

Uno schema di crittografia visuale può essere descritto per mezzo di *matrici di distribuzione*. Siano n ed m due numeri interi, dove n rappresenta il numero delle parti ed m l'espansione dei pixel (ovvero ogni pixel segreto viene espanso in m sottopixel). Si supponga di voler sviluppare uno schema per il caso base di un'immagine in bianco e nero. Tale schema è definito tramite due collezioni di matrici $\mathcal{C}_\circ$ e $\mathcal{C}_\bullet$, di formato $n \times m$, a valori in $\{\circ, \bullet\}$. Per costruire le share, per ogni pixel dell'immagine segreta di partenza, il dealer sceglie casualmente una matrice di distribuzione M da $\mathcal{C}_\circ$ se il pixel segreto è bianco, oppure da $\mathcal{C}_\bullet$, se il pixel segreto è nero, usando la riga i-esima della matrice M per assegnare i pixel alla i-esima share.

Si mostra di seguito un esempio di collezioni di matrici di distribuzione che può essere usato per realizzare uno schema di crittografia visuale per $\mathcal{Q} = \{\{1, 2\}\}$ e $\mathcal{P} = \{\{1\}, \{2\}\}$:

$$\mathcal{C}_\circ = \left\{ \begin{bmatrix} \circ\bullet \\ \circ\bullet \end{bmatrix}, \begin{bmatrix} \bullet\circ \\ \bullet\circ \end{bmatrix} \right\} \quad \mathcal{C}_\bullet = \left\{ \begin{bmatrix} \circ\bullet \\ \bullet\circ \end{bmatrix}, \begin{bmatrix} \bullet\circ \\ \circ\bullet \end{bmatrix} \right\}$$

Si è così provveduto ad istanziare uno schema a soglia di tipo (n,n)-threshold di Naor e Shamir [19] con $n = 2$, in breve indicato con la sigla (2,2)-NS.

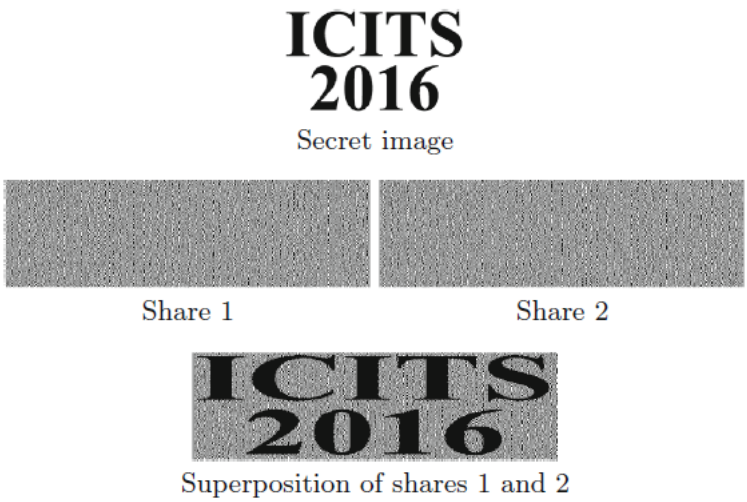


Figura 1: esempi di share e sovrapposizione in uno schema (2,2)-NS [6]

Ora per distribuire un'immagine in bianco e nero, per ogni pixel dell'immagine, il dealer ripete i seguenti passi: sceglie casualmente una matrice dalla collezione $\mathcal{C}_\sigma$, dove σ è il colore del pixel che deve essere distribuito. La riga 1 della matrice scelta sarà la share da assegnare al primo partecipante. La riga 2 sarà la share da assegnare al secondo partecipante. Con questo particolare schema ogni pixel segreto viene espanso in $m = 2$ sotto-pixel.

I partecipanti che vogliono ricostruire il segreto necessitano di sovrapporre le loro share. Si indica l'operazione di sovrapposizione con il simbolo $\oplus$. Un pixel segreto bianco è sempre ricostruito come un pixel bianco e uno nero. Un pixel segreto nero è sempre ricostruito come due pixel neri. Si noti infatti come, quando le share sono prese da $\mathcal{C}_\circ$, l'operazione di sovrapposizione eseguita per la ricostruzione si possa esprimere nei seguenti due modi:

$$[\circ\bullet] \oplus [\circ\bullet] = [\circ\bullet] \text{ oppure } [\bullet\circ] \oplus [\bullet\circ] = [\bullet\circ]$$

Si noti invece come, quando le share sono prese da $\mathcal{C}_\bullet$, la sovrapposzione si possa tradurre nei seguenti due modi:

$$[\circ\bullet] \oplus [\bullet\circ] = [\bullet\bullet] \text{ oppure } [\bullet\circ] \oplus [\circ\bullet] = [\bullet\bullet]$$

Risulta perciò evidente come ogni share, presa singolarmente, non fornisca alcuna informazione utile per poter risalire al colore del pixel segreto di partenza. Questo perchè ogni share appare in ciascuna delle collezioni delle matrici di distribuzione costruite. Significa anche che le collezioni $\mathcal{C}_\circ$ e $\mathcal{C}_\bullet$ sono state formate in modo da garantire una codifica efficace. Si riporta in figura 1 un esempio di applicazione di schema (2,2)-NS appena descritto.

2.2 Secret sharing homomorphism

La nozione di secret sharing homomorphism fu introdotta per la prima volta da Benaloh [3]. Informalmente è possibile definire uno schema di secret sharing *omomorfico* se, quando tra due partecipanti P_1 e P_2 si distribuiscono i rispettivi segreti, a e b, la somma delle due share ricevute da ciascun partecipante equivale ad una share composta dalla somma dei due segreti, ovvero:

$$P_1 \text{ verifica che: } a_1 + b_1 = (a + b)_1$$

$$P_2 \text{ verifica che: } a_2 + b_2 = (a + b)_2$$

Rimanendo nell'ambito di funzioni booleane si fornisce la seguente definizione formale:

Definizione 1. Sia data una funzione booleana $f : \{0,1\} \times \{0,1\} \to \{0,1\}$ e siano a e b due bit segreti. Siano inoltre date le share a_i e b_i con $i = 1, 2$ per ciascuno dei segreti a e b. La funzione f è computabile tramite un secret sharing homomorphism (in breve f è *share-homomorphic*) se esistono due funzioni efficientemente computabili $g_1 : \{0,1\}^+ \times \{0,1\}^+ \to \{0,1\}^+$ e $g_2 : \{0,1\}^+ \times \{0,1\}^+ \to \{0,1\}^+$ tali che, dati $c_i = g_i(a_i, b_i)$ con $i = 1, 2$, risulta che $Rec(c_1, c_2) = f(a, b)$, dove Rec rappresenta la funzione di ricostruzione del segreto condiviso.

Ciò significa che se f è share-homomorphic, allora i due partecipanti P_1 e P_2, ottenendo le share dei due segreti a e b, possono computare, ciascuno autonomamente, utilizzando g_1 e g_2, nuove share tali che, ricostruendo il segreto da queste, essi possano ottenere il medesimo risultato che otterrebbero applicando f ai segreti a e b.

Le funzioni share-homomorphic godono della proprietà di essere privatamente computabili. Questo perchè, se si guarda ad a e b come agli input segreti mantenuti dai due partecipanti, si nota come le share a_i e b_i, a causa delle proprietà di sicurezza degli schemi di secret sharing, non rilasciano alcuna informazione circa i segreti a e b: questo perchè la computazione di $c_i = g(a_i, b_i)$ è privata e non interattiva. Ne deriva che la computazione di $f(c_1, c_2)$ non rilascia alcuna informazione sugli input dei partecipanti, fatta eccezione da ciò che può essere dedotto da ciascuna delle parti dal risultato della funzione e dal proprio input.

Più formalmente è possibile enunciare il seguente teorema (per la dimostrazione del quale si rimanda alla lettura di D'Arco *et al.* [6]):

Teorema 1. Se f è una funzione share-homomorphic, allora f può essere computata privatamente in una computazione a due parti.

Si noti inoltre come, in generale, non sia vero il contrario: ci si potrebbe cioè imbattere in funzioni privatamente computabili che non sono funzioni share-homomorphic. In questo senso è possibile concludere come le funzioni share-homomorphic costituiscano un sottoinsieme proprio delle funzioni privatamente computabili, che ci si appresta a presentare in dettaglio nella prossima sezione.

2.3 Funzioni privatamente computabili

Nell'ambito della comunicazione tra due parti esiste una particolare categoria di funzioni che è possibile definire *privatamente computabili*. In presenza di due partecipanti P_1 e P_2 che custodiscono ciascuno un proprio input segreto (rispettivamente x e y) una funzione $f(x, y)$ si definisce privatamente computabile se sia P_1 che P_2 sono in grado di computare il valore corretto di f senza essere a conoscenza del valore di input dell'altro partecipante, ad eccezione di ciò che può essere dedotto dal valore della funzione in output e dal proprio input. Una tale tipologia di funzioni garantisce dunque la privatezza degli input di ciascun partecipante, attenuando i rischi causati dall'eventuale presenza di agenti malevoli coinvolti nella computazione.

Per giungere ad una definizione più formale di funzione privatamente computabile è necessario prima definire un modello in cui due parti P_1 e P_2 debbano calcolare il valore della funzione:

$$f : \{0,1\}^n \times \{0,1\}^n \to \{0,1,...,m-1\}$$

basandosi sui rispettivi input privati $x \in \{0,1\}^n$ e $y \in \{0,1\}^n$.

Si utilizza un protocollo probabilistico A, grazie al quale ai partecipanti è consentito effettuare delle decisioni casuali durante la propria computazione locale. Le due parti si scambiano messaggi, alternandosi. Il messaggio q_i che un partecipante manda al momento i-esimo è una funzione del suo input, delle sue decisioni casuali e della stringa di comunicazione composta dai messaggi $q_1, ..., q_{i-1}$ ricevuti fino all'istante i. In un tale protocollo $A(x, y)$ si assume che l'ultimo messaggio scambiato contenga il valore della funzione $f(x, y)$.

Più precisamente un protocollo A è in grado di calcolare la funzione f se:

$$\forall x, y \in \{0,1\}^n : Pr(A(x,y) = f(x,y)) > \frac{1}{2}$$

Kushilevitz [13] ed, in seguito, D'Arco *et al.* [6] formalizzano la seguente definizione di funzione privatamente computabile.

Definizione 2. Una funzione $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1,...,m-1\}$ è privatamente computabile relativamente a P_1 se esiste un protocollo A che calcola sempre il valore corretto della funzione, ovvero:

$$A(x,y) = f(x,y)\ \forall x, y \in \{0,1\}^n$$

e tale che per ogni coppia di input (x, y_1) e (x, y_2) tali per cui $f(x, y_1) = f(x, y_2)$, per ogni sequenza di comunicazione c e per ogni stringa di bit casuali r_1 posseduta da P_1 si ha che

$$Pr(c|r_1, (x, y_1)) = Pr(c|r_1, (x, y_2))$$

dove la probabilità è calcolata su tutte le possibili stringhe casuali di P_2. Un definizione simile può essere fornita anche relativamente a P_2.

In protocolli a due parti siffatti risulta conveniente rappresentare qualunque funzione $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1,...,m-1\}$ come una matrice $2^n \times 2^n$ con elementi a valori in $\{0,1,...,m-1\}$. Una tale matrice quadrata verrà indicata con M_f. Ciascuna riga di M_f rappresenta un possibile input di x relativo a P_1, ciascuna colonna rappresenta invece un possibile input di y relativo a P_2. Ciascuna cella (x, y) di M_f conterrà il valore di $f(x, y)$.

Risulta qui utile ricordare dall'algebra lineare la definizione di sottomatrice di una matrice quadrata:

Definizione 3. Si definisce sottomatrice di una matrice $M_{n\times n}$, con n intero positivo, una matrice $B_{r\times s}$, con r ed s tali che $0 \leq r \leq n$ e $0 \leq s \leq n$, ottenuta da M rimuovendo $n - r$ righe e $n - s$ colonne.

Di conseguenza si può fornire la successiva definizione:

Definizione 4. Si definisce *monocromatica* una sottomatrice di M_f dove la funzione f è costante, ovvero restituisce in output sempre lo stesso valore.

$f_1(x,y)$	$y=0$	$y=1$
$x=0$	0	0
$x=0$	0	0

$f_2(x,y)$	$y=0$	$y=1$	$y=2$
$x=0$	1	0	1
$x=1$	0	0	0
$x=2$	1	0	1

$f_3(x,y)$	$y=0$	$y=1$	$y=2$	$y=3$
$x=0$	1	1	0	2
$x=1$	1	1	3	2
$x=2$	3	2	1	0
$x=3$	0	3	2	1

Figura 2: esempi di sottomatrici monocromatiche

Si riportano in figura 2 alcuni esempi di matrici rappresentanti funzioni contenenti sottomatrici monocromatiche, evidenziate in giallo. In particolare si noti come in f_1 l'intera matrice possa definirsi monocromatica in quanto la funzione è costante su tutte le sue celle. In f_2 la sottomatrice monocromatica è quella ottenuta eliminando la seconda riga e la seconda colonna. In f_3 la sottomatrice monocromatica è quella ottenuta eliminando terza e quarta riga e terza e quarta colonna.

Kushilevitz [13] fornisce un pratico criterio (denominato di *caratterizzazione combinatoria completa*) utilizzabile nell'analisi delle funzioni computabili privatamente. È basato su alcune relazioni di equivalenza rilevabili sulla matrice che rappresenta la funzione. Costituirà un criterio fondamentale per distinguere tra funzioni privatamente computabili e funzioni che non godono di tale proprietà. La seguente definizione illustra in cosa consista il criterio di caratterizzazione combinatoria completa, di cui si farà ampio uso nei capitoli successivi dell'elaborato:

Definizione 5. Sia $M = C \times D$ una matrice rappresentante una funzione $f(x,y)$. La relazione $\sim$ sulle righe di M viene definita come segue: $x_1, x_2 \in C$ soddisfano $x_1 \sim x_2$ se esiste $y \in D$ tale che $M_{x_1,y} = M_{x_2,y}$. La relazione di equivalenza $\equiv$ sulle righe di M viene definita come la *chiusura transitiva* della relazione $\sim$, ovvero $x_1, x_2 \in C$ soddisfano $x_1 \equiv x_2$ se esistono $z_1, z_2, ..., z_l \in C$ tali che $x_1 \sim z_1 \sim z_2 \sim ... \sim z_l \sim x_2$. Le relazioni $\sim$ e $\equiv$ sono definite analogamente anche sulle colonne della matrice M.

Più informalmente si può affermare che due righe rispettano la relazione $\sim$ se, in corrispondenza della stessa colonna, queste assumono lo stesso valore. Analogamente due colonne sono in relazione $\sim$ se assumono lo stesso valore in corrispondenza di una stessa riga. La relazione di equivalenza $\equiv$ è estesa a tutte le righe/colonne della

matrice e può essere raggiunta tramite chiusura transitiva tra righe/colonne già in relazione tra loro, per le quali è possibile costruire una catena di relazioni $\sim$ che possa arrivare a coprire tutte le n righe/colonne.

Considerato dunque il criterio appena fornito è possibile suddividere le matrici rappresentanti funzioni in logica multivalore in due categorie distinte:

- **matrici vietate**, rappresentanti funzioni multivalore che non godono della proprietà di essere privatamente computabili;
- **matrici ammesse**, rappresentanti funzioni multivalore privatamente computabili.

La definizione di matrice vietata viene fornita di seguito:

Definizione 6. Una matrice M viene detta *vietata* se non è monocromatica, tutte le sue righe sono equivalenti e tutte le sue colonne sono equivalenti, cioè se ogni $x_1, x_2 \in C$ soddisfa $x_1 \equiv x_2$ e ogni $y_1, y_2 \in D$ soddisfa $y_1 \equiv y_2$.

Significa, ad esempio, che la matrice della funzione f_1 alla figura 2 non può essere definita vietata in quanto monocromatica. Si riportano due esempi di matrici vietate in figura 3. La funzione f_4 è vietata perché non è monocromatica, l'equivalenza di riga $x_0 \sim x_1$ è raggiunta in corrispondenza della colonna y_1 e l'equivalenza di colonna $y_0 \sim y_1$ è raggiunta in corrispondenza della riga x_0. La funzione f_5 è altresì vietata perché non monocromatica, l'equivalenza di riga $x_0 \sim x_2$ e $x_1 \sim x_2$ è raggiunta rispettivamente in corrispondenza delle colonne y_0 e y_2 (la catena di relazioni $x_0 \sim x_1 \sim x_2$ è ottenuta tramite chiusura transitiva) e l'equivalenza di colonna $y_0 \sim y_1$ e $y_1 \sim y_2$ è raggiunta rispettivamente in corrispondenza delle righe x_1 e x_0 (e la catena di relazioni $y_0 \sim y_1 \sim y_2$ è ottenuta tramite chiusura transitiva).

$f_4(x,y)$	y_0	y_1
x_0	1	1
x_1	0	1

$f_5(x,y)$	y_0	y_1	y_2
x_0	1	3	3
x_1	4	4	2
x_2	1	0	2

Figura 3: esempi di matrici vietate

Si riporta di seguito la definizione di matrice ammessa:

Definizione 7. Una matrice M viene detta *ammessa* se è monocromatica oppure se non raggiunge l'equivalenza a livello di riga e/o a livello di colonna nelle modalità descritte dal criterio di caratterizzazione combinatoria completa.

In figura 4 si evidenziano due esempi di matrici ammesse. La funzione f_6 è ammessa perché, pur non essendo monocromatica, raggiunge l'equivalenza solo a livello di riga (in corrispondenza delle colonne y_0 e y_1) e non a livello di colonna. La funzione f_7 è altresì ammessa perché, pur non essendo monocromatica, non raggiunge né l'equivalenza di riga (l'unica relazione valida è $x_0 \sim x_2$ in corrispondenza delle colonne y_0 e y_1) né quella di colonna (l'unica relazione valida è $y_0 \sim y_1$ in corrispondenza della riga x_1).

$f_6(x,y)$	y_0	y_1
x_0	0	1
x_1	0	1

$f_7(x,y)$	y_0	y_1	y_2
x_0	1	0	3
x_1	4	4	0
x_2	1	0	2

Figura 4: esempi di matrici ammesse

Kushilevitz, tramite il lavoro sviluppato in [13], arriva infine a dimostrare il seguente teorema:

Teorema 2. Se una funzione f è rappresentata da una matrice M_f che contiene almeno una sottomatrice vietata, allora f non è privatamente computabile ed M_f è vietata.

È inoltre possibile dimostrare [13] come tale teorema rappresenti una condizione *necessaria e sufficiente* per discriminare tra funzioni privatamente computabili e funzioni che non godono di tale proprietà. Ne segue il seguente corollario:

Corollario 1. Se una funzione f è rappresentata da una matrice M_f che non contiene alcuna sottomatrice vietata, allora f è privatamente computabile ed M_f è ammessa.

Verrà dunque avviata nel seguito un'analisi sulle funzioni in logica multivalore. Lo scopo è quello di far emergere quante e quali matrici ammesse (ovvero rappresentanti funzioni privatamente computabili) si possano ricavare, nelle differenti logiche multivalore.

Si inizierà esponendo risultati già noti raccolti in letteratura scientifica, in primo luogo per funzioni in logica binaria, in seguito introducendo l'insieme delle funzioni in logica ternaria, trattato poi in maniera estesa nei capitoli successivi.

2.3.1 Risultati noti per funzioni in logica binaria

In questa sezione si riportano alcuni risultati fondamentali ottenuti da D'Arco *et al.* [6] nell'ambito dello studio di funzioni privatamente computabili in logica binaria. Studiare funzioni in logica a 2 valori significa porre $m = 2$ e concentrare l'analisi sulla seguente casistica:

$$f : \{0,1\} \times \{0,1\} \to \{0,1\}$$

Si è in presenza di matrici quadrate 2×2 con due valori possibili per ogni cella $\{0,1\}$. Le permutazioni con ripetizioni per ciascuna riga saranno 2^2, è possibile perciò ottenere un totale di $(2^2)^2 = 16$ matrici differenti rappresentanti funzioni in logica binaria (figura 5).

0	$y=0$	$y=1$
$x=0$	0	0
$x=1$	0	0

and	$y=0$	$y=1$
$x=0$	0	0
$x=1$	0	1

x$\overline{\text{y}}$	$y=0$	$y=1$
$x=0$	0	0
$x=1$	1	0

x	$y=0$	$y=1$
$x=0$	0	0
$x=1$	1	1

$\overline{\text{x}}$y	$y=0$	$y=1$
$x=0$	0	1
$x=1$	0	0

y	$y=0$	$y=1$
$x=0$	0	1
$x=1$	0	1

xor	$y=0$	$y=1$
$x=0$	0	1
$x=1$	1	0

or	$y=0$	$y=1$
$x=0$	0	1
$x=1$	1	1

$\overline{\text{or}}$	$y=0$	$y=1$
$x=0$	1	0
$x=1$	0	0

$\overline{\text{xor}}$	$y=0$	$y=1$
$x=0$	1	0
$x=1$	0	1

$\overline{\text{y}}$	$y=0$	$y=1$
$x=0$	1	0
$x=1$	1	0

x + $\overline{\text{y}}$	$y=0$	$y=1$
$x=0$	1	0
$x=1$	1	1

$\overline{\text{x}}$	$y=0$	$y=1$
$x=0$	1	1
$x=1$	0	0

$\overline{\text{x}}$ + y	$y=0$	$y=1$
$x=0$	1	1
$x=1$	0	1

$\overline{\text{and}}$	$y=0$	$y=1$
$x=0$	1	1
$x=1$	1	0

1	$y=0$	$y=1$
$x=0$	1	1
$x=1$	1	1

Figura 5: 16 matrici rappresentanti funzioni booleane [6]

Si procede applicando il criterio di caratterizzazione combinatoria completa a tutte le matrici in figura 5, giungendo a suddividere le matrici nei seguenti due gruppi:

- **Gruppo 1**: comprendente le funzioni 0, x, y, xor, $\overline{xor}$, $\overline{y}$, $\overline{x}$, 1. Costituisce il gruppo delle matrici ammesse, rappresentanti funzioni privatamente computabili. Queste potranno perciò essere calcolate dai partecipanti P_1 e P_2 senza rivelare alcuna informazione sui rispettivi input privati.

- **Gruppo 2**: comprendente le funzioni *and*, $x\overline{y}$, $\overline{x}y$, *or*, $\overline{or}$, $x + \overline{y}$, $\overline{x} + y$, $\overline{and}$. Costituisce il gruppo delle matrici vietate, rappresentanti funzioni non privatamente computabili. Significa che, nel tentativo di calcolare tali funzioni, i partecipanti P_1 e P_2 sarebbero costretti a rivelare informazioni sui rispettivi input privati.

Si noti come emerga un unico schema ricorrente (un *pattern*) per il gruppo 2 appena individuato. Si fornirà di seguito una definizione formale di tale schema:

Pattern [3|1]**.** Nell'ambito di funzioni booleane, se esistono $x_1, x_2, y_1, y_2 \in \{0, 1\}$ tali che $f(x_1, y_1) = f(x_1, y_2) = f(x_2, y_1) = a$ e $f(x_2, y_2) = b$, allora la funzione f non è privatamente computabile.

Informalmente, se una matrice 2×2 in logica binaria si presenta con 3 valori uguali tra loro ed uno diverso (da qui il nome attribuito al pattern) si può concludere di essere in presenza di una matrice vietata.

Riguardo il gruppo 1 si noti come le 8 matrici ammesse rappresentino altrettante funzioni lineari in logica binaria, esprimibili come:

$$f(x, y) = ax +_2 by +_2 c$$

con

$$a, b, c \in \{0, 1\}$$

dove $+_2$ rappresenta la somma modulo 2.

Si elencano di seguito le 8 funzioni lineari privatamente computabili ottenibili e come queste sono equivalenti alle funzioni citate nel gruppo 1:

- le funzioni 0, x, y, 1 (già presenti nel gruppo 1);
- la funzione $x +_2 1$ (equivalente alla funzione $\overline{x}$);
- la funzione $y +_2 1$ (equivalente alla funzione $\overline{y}$);
- la funzione $x +_2 y$ (equivalente alla funzione xor);
- la funzione $x +_2 y +_2 1$ (equivalente alla funzione $\overline{xor}$).

Si può perciò concludere quanto segue:

- su 16 matrici totali in logica binaria la metà (gruppo 1) rappresentano funzioni privatamente computabili;
- una condizione necessaria e sufficiente affinchè in logica binaria una funzione sia privatamente computabile è che questa sia lineare rispetto alla somma modulo 2;
- l'altra metà (gruppo 2) rappresenta funzioni non privatamente computabili. Queste risultano rapidamente individuabili tramite il pattern [3|1] precedentemente definito.

2.3.2 Risultati noti per funzioni in logica ternaria

L'estensione dell'analisi alle funzioni in logica a 3 valori prevede di concentrarsi su matrici quadrate in formato 3×3. Inoltre si ragionerà su $m = 3$ possibili differenti valori per ciascun bit $\{0, 1, 2\}$.

L'insieme di matrici da indagare crescerà esponenzialmente rispetto allo scenario in logica binaria esposto alla sezione 2.3.1. Le permutazioni con ripetizioni per ciascuna riga saranno $3^3 = 27$, è possibile perciò ottenere un totale di $(3^3)^3 = 19683$ matrici differenti rappresentanti funzioni in logica ternaria.

Si riassume di seguito quanto già presentato nella tesi di Bacci [1] in forma più estesa e dettagliata. Si è provato, innanzitutto, a ripetere l'indagine limitatamente alla funzioni lineari, ora in logica ternaria, esprimibili come:

$$f(x, y) = ax +_3 by +_3 c$$

con

$$a, b, c \in \{0, 1, 2\}$$

dove $+_3$ rappresenta la somma modulo 3.

Si riportano in figura 6 le matrici rappresentanti le 27 funzioni lineari ottenibili. Applicando il criterio di caratterizzazione combinatoria completa e tenendo presente le definizioni di matrici vietate e ammesse, è possibile concludere, per osservazione diretta, come siano tutte funzioni privatamente computabili. Le funzioni 0, 1 e 2 in particolare lo sono in quanto monocromatiche. Le altre 24 funzioni lo sono perchè non raggiungono l'equivalenza di riga e/o l'equivalenza di colonna (si ricorda che entrambe le equivalenze devono essere raggiunte perché una matrice non monocromatica possa definirsi vietata).

0	y = 0	y = 1	y = 2
x = 0	0	0	0
x = 1	0	0	0
x = 2	0	0	0

1	y = 0	y = 1	y = 2
x = 0	1	1	1
x = 1	1	1	1
x = 2	1	1	1

2	y = 0	y = 1	y = 2
x = 0	2	2	2
x = 1	2	2	2
x = 2	2	2	2

X	y = 0	y = 1	y = 2
x = 0	0	0	0
x = 1	1	1	1
x = 2	2	2	2

Y	y = 0	y = 1	y = 2
x = 0	0	1	2
x = 1	0	1	2
x = 2	0	1	2

$X +_3 1$	y = 0	y = 1	y = 2
x = 0	1	1	1
x = 1	2	2	2
x = 2	0	0	0

$X +_3 2$	y = 0	y = 1	y = 2
x = 0	2	2	2
x = 1	0	0	0
x = 2	1	1	1

$Y +_3 1$	y = 0	y = 1	y = 2
x = 0	1	2	0
x = 1	1	2	0
x = 2	1	2	0

$Y +_3 2$	y = 0	y = 1	y = 2
x = 0	2	0	1
x = 1	2	0	1
x = 2	2	0	1

$X +_3 Y$	y = 0	y = 1	y = 2
x = 0	0	1	2
x = 1	1	2	0
x = 2	2	0	1

$X +_3 Y +_3 1$	y = 0	y = 1	y = 2
x = 0	1	2	0
x = 1	2	0	1
x = 2	0	1	2

$X +_3 Y +_3 2$	y = 0	y = 1	y = 2
x = 0	2	0	1
x = 1	0	1	2
x = 2	1	2	0

2X	y = 0	y = 1	y = 2
x = 0	0	0	0
x = 1	2	2	2
x = 2	1	1	1

$2X +_3 1$	y = 0	y = 1	y = 2
x = 0	1	1	1
x = 1	0	0	0
x = 2	2	2	2

$2X +_3 2$	y = 0	y = 1	y = 2
x = 0	2	2	2
x = 1	1	1	1
x = 2	0	0	0

$2X +_3 Y$	y = 0	y = 1	y = 2
x = 0	0	1	2
x = 1	2	0	1
x = 2	1	2	0

$2X +_3 Y +_3 1$	y = 0	y = 1	y = 2
x = 0	1	2	0
x = 1	0	1	2
x = 2	2	0	1

$2X +_3 Y +_3 2$	y = 0	y = 1	y = 2
x = 0	2	0	1
x = 1	1	2	0
x = 2	0	1	2

2Y	y = 0	y = 1	y = 2
x = 0	0	2	1
x = 1	0	2	1
x = 2	0	2	1

$2Y +_3 1$	y = 0	y = 1	y = 2
x = 0	1	0	2
x = 1	1	0	2
x = 2	1	0	2

$2Y +_3 2$	y = 0	y = 1	y = 2
x = 0	2	1	0
x = 1	2	1	0
x = 2	2	1	0

$2Y +_3 X$	y = 0	y = 1	y = 2
x = 0	0	2	1
x = 1	1	0	2
x = 2	2	1	0

$2Y +_3 X +_3 1$	y = 0	y = 1	y = 2
x = 0	1	0	2
x = 1	2	1	0
x = 2	0	2	1

$2Y +_3 X +_3 2$	y = 0	y = 1	y = 2
x = 0	2	1	0
x = 1	0	2	1
x = 2	1	0	2

$2Y +_3 2X$	y = 0	y = 1	y = 2
x = 0	0	2	1
x = 1	2	1	0
x = 2	1	0	2

$2Y +_3 2X +_3 1$	y = 0	y = 1	y = 2
x = 0	1	0	2
x = 1	0	2	1
x = 2	2	1	0

$2Y +_3 2X +_3 2$	y = 0	y = 1	y = 2
x = 0	2	1	0
x = 1	1	0	2
x = 2	0	2	1

Figura 6: matrici rappresentanti funzioni lineari in logica ternaria [1]

La caratteristica di linearità della funzione costituisce dunque una condizione sufficiente per l'individuazione di funzioni privatamente computabili in logica ternaria. Non si tratta tuttavia di una condizione necessaria, considerando che, verificando a campione alcune funzioni non lineari, ci si può imbattere sia in matrici ammesse (figura 7) sia in matrici vietate (figura 8).

X^2	$y=0$	$y=1$	$y=2$
$x=0$	0	0	0
$x=1$	1	1	1
$x=2$	1	1	1

$X^2 +_3 Y^2$	$y=0$	$y=1$	$y=2$
$x=0$	0	1	1
$x=1$	1	2	2
$x=2$	1	2	2

Figura 7: esempi di matrici ammesse, rappresentanti funzioni non lineari [1]

XY	$y=0$	$y=1$	$y=2$
$x=0$	0	0	0
$x=1$	0	1	2
$x=2$	0	2	1

X^2Y	$y=0$	$y=1$	$y=2$
$x=0$	0	0	0
$x=1$	0	1	2
$x=2$	0	1	2

X^2Y^2	$y=0$	$y=1$	$y=2$
$x=0$	0	0	0
$x=1$	0	1	1
$x=2$	0	1	1

Figura 8: esempi di matrici vietate, rappresentanti funzioni non lineari [1]

Appare evidente come, con gli strumenti attualmente a disposizione, nulla si possa concludere al momento in merito alle altre $19683 - 27 = 19656$ matrici che rappresentano l'insieme delle funzioni a 3 valori che si vorrebbe esplorare. Proprio lo sviluppo di strumenti e tecniche per rendere organizzata ed efficace un'analisi sistematica su un tale insieme di funzioni sarà l'oggetto del prossimo capitolo 3.

Capitolo 3

Analisi di funzioni in logica ternaria

Il presente capitolo sarà dedicato ad approfondire lo studio delle funzioni in logica a 3 valori. Si è proceduto ad un'analisi quantitativa con l'obiettivo di determinare l'esatto numero di funzioni privatamente computabili tra tutte le funzioni in logica ternaria. Gli algoritmi di controllo sono stati sviluppati basandosi sul criterio di caratterizzazione combinatoria completa proposto da Kushilevitz [13].

L'output ottenuto è stato ulteriormente indagato in dettaglio al fine di far emergere un limitato numero di schemi ricorrenti (o *pattern*) con precise caratteristiche. Si vedrà come a ciascuno di tali pattern sarà possibile abbinare un sottoinsieme di funzioni tra la totalità di quelle privatamente computabili. Proprio il tentativo di ricondurre sistematicamente gruppi di funzioni ad un determinato schema costituirà il punto di partenza per avviare un'analisi qualitativa, cercando di distinguere le funzioni che ricadono nei pattern individuati da quelle che, non appartenendo ad alcuno di essi, risulteranno non privatamente computabili.

Ogni passaggio è supportato da una verifica sperimentale svolta tramite lo sviluppo di opportuni algoritmi i cui punti di attenzione fondamentali verranno riportati sotto forma di pseudocodice.

3.1 Analisi quantitativa di funzioni privatamente computabili in logica ternaria

Si effettuerà di seguito un'analisi quantitativa delle funzioni privatamente computabili in logica ternaria. L'obiettivo è arrivare a determinare quante funzioni nella forma $f(x, y)$ con $x, y \in \{0, 1, 2\}$ sono privatamente computabili.

Si giungerà alla risposta avvalendosi della base teorica sviluppata dal lavoro di Kushilevitz [13], partendo dalle definizioni già fornite di:

- matrici monocromatiche, matrici vietate e matrici ammesse;
- criterio di caratterizzazione combinatoria completa.

3.1.1 Generazione di tutte le matrici 3×3

Si è proceduto innanzitutto a sviluppare gli algoritmi necessari per generare tutte le matrici 3×3, ragionando secondo le possibili permutazioni delle righe. Ciascun bit può assumere uno dei tre valori $\{0, 1, 2\}$. In matrici di tali dimensioni ciascuna riga può quindi apparire in un numero di varianti pari alle possibili permutazioni con ripetizioni di 3 valori su una stringa di lunghezza 3, ovvero $3^3 = 27$. Inoltre, fissata la permutazione alla prima riga, anche la seconda potrà variare in 27 modi differenti. Infine, fissate le permutazioni per le prime due righe, anche la terza potrà variare in 27 possibili configurazioni differenti.

Risulta perciò possibile immaginare 3 iterazioni annidate, necessarie per la generazione di un totale di $27 \times 27 \times 27 = (3^3)^3 = (27^3) = 19683$ matrici in logica ternaria.

Si faccia riferimento allo pseudocodice 1 per maggiori dettagli sull'approccio iterativo scelto.

Pseudocodice 1 Generazione di tutte le possibili matrici 3×3

```
for permutazioni da 0 a 26 do
    PERMUTA(riga_3)
    for permutazioni da 0 a 26 do
        PERMUTA(riga_2)
        for permutazioni da 0 a 26 do
            PERMUTA(riga_1)
        end for
    end for
end for
```

La funzione PERMUTA, i cui dettagli sono riportati allo pseudocodice 2, si occupa di generare le 27 differenti combinazioni per ciascuna riga. Si specificano di seguito i parametri in ingresso alla funzione:

- mtx[n][n]: matrice in costruzione di n righe $\times$ n colonne ($n = 3$). Tale struttura matriciale, prima di essere passata alla funzione, viene opportunamente inizializzata ponendo tutte le celle della matrice pari a 0;
- numRiga: il numero di riga che verrà sottoposto a permutazione;
- numPerm: il numero di iterazione del ciclo *for* esterno alla funzione richiamata (visibile allo pseudocodice 1). Indicherà il numero della permutazione alla quale si è giunti;
- val[ ]: array in cui sono riportati in ordine crescente i valori $\{0, 1, 2\}$ che ogni cella della matrice può assumere;
- lenVal: la cardinalità dell'array dei valori;
- lenPerm: la lunghezza della permutazione da effettuare.

Pseudocodice 2 funzione PERMUTA

```
function PERMUTA(mtx[n][n], numRiga, numPerm, val[ ], lenVal, lenPerm)
    for i = 0 to i < lenPerm do
     mtx[numRiga][n] = val[numPerm % lenVal]
     numPerm = numPerm / lenVal
    end for
end function
```

L'esecuzione di PERMUTA darà luogo alle 27 permutazioni nel seguente ordine:

- 00: **0 0 0**
- 01: **1 0 0**
- 02: **2 0 0**
- 03: **0 1 0**
- 04: **1 1 0**
- 05: **2 1 0**
- 06: **0 2 0**
- 07: **1 2 0**
- 08: **2 2 0**
- 09: **0 0 1**
- 10: **1 0 1**
- 11: **2 0 1**
- 12: **0 1 1**
- 13: **1 1 1**
- 14: **2 1 1**
- 15: **0 2 1**
- 16: **1 2 1**
- 17: **2 2 1**
- 18: **0 0 2**
- 19: **1 0 2**
- 20: **2 0 2**
- 21: **0 1 2**
- 22: **1 1 2**
- 23: **2 1 2**
- 24: **0 2 2**
- 25: **1 2 2**
- 26: **2 2 2**

L'ordinamento descritto si rifletterà sulla sequenza con cui verranno generate le matrici 3×3. A ciascuna di queste verrà associato un ID univoco, una numerazione progressiva da 1 a 19683. Di seguito si riportano alcune matrici generate con relativo ID. Consultando l'ordine delle permutazioni riportato in precedenza si può risalire al perché un certa matrice è stata generata nell'ordine e con l'ID indicato.

- matrice ID 1:

  0 0 0 ($riga_1$ 00)
  0 0 0 ($riga_2$ 00)
  0 0 0 ($riga_3$ 00)

- matrice ID 2:

  1 0 0 ($riga_1$ 01)
  0 0 0 ($riga_2$ 00)
  0 0 0 ($riga_3$ 00)

- matrice ID 3:

  2 0 0 ($riga_1$ 02)
  0 0 0 ($riga_2$ 00)
  0 0 0 ($riga_3$ 00)

- ...

- matrice ID 27:

  2 2 2 ($riga_1$ 26)
  0 0 0 ($riga_2$ 00)
  0 0 0 ($riga_3$ 00)

- matrice ID 28:

  0 0 0 ($riga_1$ 00)
  1 0 0 ($riga_2$ 01)
  0 0 0 ($riga_3$ 00)

- matrice ID 29:

  1 0 0 ($riga_1$ 01)
  1 0 0 ($riga_2$ 01)
  0 0 0 ($riga_3$ 00)

- ...

- matrice ID 729 (27^2):

  2 2 2 ($riga_1$ 26)
  2 2 2 ($riga_2$ 26)
  0 0 0 ($riga_3$ 00)

- matrice ID 730:

  0 0 0 ($riga_1$ 00)
  0 0 0 ($riga_2$ 00)
  1 0 0 ($riga_3$ 01)

- matrice ID 731:

  1 0 0 ($riga_1$ 01)
  0 0 0 ($riga_2$ 00)
  1 0 0 ($riga_3$ 01)

- ...

- ID 19681:

  0 2 2 ($riga_1$ 24)
  2 2 2 ($riga_2$ 26)
  2 2 2 ($riga_3$ 26)

- ID 19682:

  1 2 2 ($riga_1$ 25)
  2 2 2 ($riga_2$ 26)
  2 2 2 ($riga_3$ 26)

- ID 19683 (27^3):

  2 2 2 ($riga_1$ 26)
  2 2 2 ($riga_2$ 26)
  2 2 2 ($riga_3$ 26)

Sono 3 le matrici monocromatiche in un insieme così definito:

- ID 1:

```
0 0 0
0 0 0
0 0 0
```

- ID 9842:

```
1 1 1
1 1 1
1 1 1
```

- ID 19683:

```
2 2 2
2 2 2
2 2 2
```

Trattandosi di matrici ammesse per definizione si decide di filtrarle in partenza, escludendole dall'insieme delle matrici da processare. Una semplice funzione che può servire allo scopo è riportata allo pseudocodice 3.

Pseudocodice 3 funzione CHECKMTXMONO

```
function CHECKMTXMONO(mtx[n][n])
    for row = 0 to row < 3 do
        for col = 0 to col < 3 do
            if mtx[0][0] != mtx[row][col] then
                return 0                          ▷ MONO FALSE
            end if
        end for
    end for
     return 1                                     ▷ MONO TRUE
end function
```

Dovendo eseguire su ciascuna matrice una parte rilevante della computazione del programma che si sta sviluppando è opportuno sin da subito comprendere se e come sia possibile ridurre in partenza il numero di matrici da processare.

Il criterio di caratterizzazione combinatoria completa non è in effetti legato a quali specifici valori assumano i bit nella matrice. È piuttosto la loro posizione reciproca assunta all'interno della matrice ad essere rilevante. È possibile notare a tal proposito come le rimanenti 19680 matrici non monocromatiche abbiano tutte la seguente particolarità: ciascuna può essere generata con $3! = 6$ diverse permutazioni semplicemente variando il valore dei bit, non la loro posizione reciproca. Si consideri la seguente matrice ID 2:

```
1 0 0
0 0 0
0 0 0
```

Una tale matrice è chiaramente vietata, in quanto non monocromatica ed equivalente sia a livello di righe che di colonne.

Far processare al programma la matrice ID 2 e le altre sue 5 permutazioni (riportate di seguito) sarebbe uno spreco in termini di tempo e risorse computazionali, in quanto si giungerebbe sempre alla medesima conclusione: sarebbero dichiarate tutte matrici vietate.

- ID 2:

  1 0 0
  0 0 0
  0 0 0

- ID 9841:

  0 1 1
  1 1 1
  1 1 1

- ID 19681:

  0 2 2
  2 2 2
  2 2 2

- ID 3:

  2 0 0
  0 0 0
  0 0 0

- ID 9843:

  2 1 1
  1 1 1
  1 1 1

- ID 19682:

  1 2 2
  2 2 2
  2 2 2

Si è quindi proceduto a creare una firma identificativa (una *signature*) per ogni schema matriciale. Lo scopo è assegnare la stessa signature alle 6 matrici permutate e a nessun altra. In tal modo si processerà solo la prima (ID 2), ignorando le altre.

Nello pseudocodice 4 si riporta un esempio di funzione in grado di generare, per ogni matrice, una simile signature. Alla funzione viene passata in ingresso la matrice, oltre ai tre parametri $bit1$, $bit2$, $bit3$. Il parametro $bit1$ è settato pari al bit della matrice in posizione [0][0]. Si procede dunque a visitare da sinistra a destra gli elementi della matrice per la prima riga, dunque la seconda riga e infine la terza. Il parametro $bit2$ è settato uguale al primo bit diverso dal $bit1$ che si incontra nella visita descritta. Il parametro $bit3$ (se presente) è settato uguale al primo bit diverso dal $bit2$ e dal $bit3$.

Prendendo come esempio la matrice ID 14228 si noti di seguito come risulterebbero popolati i 3 parametri:

$$(bit1 = 1 \mid bit2 = 2 \mid bit3 = 0)$$

1 2 2
1 1 1
1 0 2

Pseudocodice 4 funzione BUILDMTXSIGN

```
function BUILDMTXSIGN(mtx[n][n], bit1, bit2, bit3)
    cell = 8;
    for row = 0 to row < 3 do
        for col = 0 to col < 3 do
            exp = cell;
            value = 1, base = 10;
            while exp != 0 do
                value *= base;
                exp = exp - 1;
            end while
            if mtx[row][col] == bit1 then
                sign = sign + 1 * value;
            else if mtx[row][col] == bit2 then
                sign = sign + 2 * value;
            else if mtx[row][col] == bit3 then
                sign = sign + 3 * value;
            end if
            cell = cell - 1;
        end for
    end for
end function
```

La signature ritornata dalla funzione BUILDMTXSIGN è pari al seguente numero intero di 9 cifre (una per ogni bit cui viene attribuito un peso):

- Signature ID 14228: **`122 111 132`**;
- Alle celle pari a *bit*1 viene attribuito peso 1;
- Alle celle pari a *bit*2 viene attribuito peso 2;
- Alle celle pari a *bit*3 viene attribuito peso 3.

Allo stesso modo e applicando la medesima logica, ci si può convincere come le 6 matrici permutate riportate in precedenza (ID 2, 3, 9841, 9843, 19681, 19682) condividano la medesima signature (**`122 222 222`**) e perchè non vi possa essere nessun'altra matrice cui è applicata quella signature.

L'applicazione della logica a signature qui introdotta porta a ridurre di 6 volte il numero di matrici da controllare. Risulteranno perciò:

- $\frac{19680}{6} = 3280$ matrici *uniche* rispetto al criterio di combinazione combinatoria completa che ci si appresta ad applicare.

3.1.2 Sottomatrici di matrice 3×3

La base teorica richiamata nel capitolo 2 evidenzia in più punti la necessità di un controllo da effettuare non soltanto sulla matrice principale M_f ma su ognuna delle sue sottomatrici. Si ricorda innanzitutto che si definisce *sottomatrice* di una matrice $M_{n\times n}$, con n intero positivo, una matrice $B_{r\times s}$, con r ed s tali che $0 \leq r \leq n$ e $0 \leq s \leq n$, ottenuta da M rimuovendo $n - r$ righe e $n - s$ colonne.

In particolare in una matrice quadrata $M_{n\times n}$, volendo trovare il numero esatto di sottomatrici nel formato $B_{r\times s}$ è possibile utilizzare la seguente formula basata sui binomiali [14]:

$$|B_{r\times s}| = \binom{n}{r} \cdot \binom{n}{s}$$

Tenendo presente, dalla definizione di binomiale, come ciò si possa tradurre in:

$$|B_{r\times s}| = \frac{n!}{r!(n-r)!} \cdot \frac{n!}{s!(n-s)!}$$

Si procede applicando la formula a ciascun possibile formato di sottomatrici presenti in una matrice 3×3:

- sottomatrici 1×1: $\binom{3}{1}\binom{3}{1} = 9$
- sottomatrici 1×2: $\binom{3}{1}\binom{3}{2} = 9$
- sottomatrici 2×1: $\binom{3}{2}\binom{3}{1} = 9$
- sottomatrici 2×2: $\binom{3}{2}\binom{3}{2} = 9$
- sottomatrici 1×3: $\binom{3}{1}\binom{3}{3} = 3$
- sottomatrici 3×1: $\binom{3}{3}\binom{3}{1} = 3$
- sottomatrici 2×3: $\binom{3}{2}\binom{3}{3} = 3$
- sottomatrici 3×2: $\binom{3}{3}\binom{3}{2} = 3$
- sottomatrice 3×3: $\binom{3}{3}\binom{3}{3} = 1$ (la matrice principale)

Sommando tutti i contributi (e includendo nella somma anche la matrice principale) si ottengono un totale di $(9 \times 4) + (3 \times 4) + 1 = 36 + 12 + 1 = 49$ sottomatrici.

Risulta qui opportuno ricordare la definizione di matrice vietata:

Definizione 6. Una matrice M viene detta *vietata* se non è monocromatica, tutte le sue righe sono equivalenti e tutte le sue colonne sono equivalenti, cioè se ogni $x_1, x_2 \in C$ soddisfa $x_1 \equiv x_2$ e ogni $y_1, y_2 \in D$ soddisfa $y_1 \equiv y_2$.

Servendosi della definizione è possibile escludere a priori dai futuri controlli alcuni formati di sottomatrici precedentemente elencati. Precisamente:

- le sottomatrici costituite da un solo elemento (1×1) sono monocromatiche per costruzione. Si possono escludere dal controllo in quanto sempre ammesse;
- le sottomatrici costituite da una sola riga oppure una sola colonna (formati 1×2, 1×3, 3×1 e 2×1) possono solamente essere monocromatiche oppure avere almeno un elemento diverso dagli altri. In questo secondo caso l'equivalenza di quella singola riga (o colonna) non si potrà mai verificare. In entrambi i casi si possono escludere tali formati dal controllo perchè le sottomatrici saranno sempre ammesse.

Si ricapitolano di seguito i formati di sottomatrici rimaste, indicandoli su una matrice 3×3 di esempio con gli elementi tutti diversi tra loro (numerati da 1 a 9).

- n° 1 sottomatrice 3×3 (la matrice principale):

```
1 2 3
4 5 6
7 8 9
```

- n° 3 sottomatrici 2×3:

```
1 2 3        1 2 3        4 5 6
4 5 6        7 8 9        7 8 9
```

- n° 3 sottomatrici 3×2:

```
1 2          1 3          2 3
4 5          4 6          5 6
7 8          7 9          8 9
```

- n° 9 sottomatrici 2×2:

```
1 2        1 3        2 3
4 5        4 6        5 6

1 2        1 3        2 3
7 8        7 9        8 9

4 5        4 6        5 6
7 8        7 9        8 9
```

È proprio su tale insieme di (1+3+3+9) = 16 sottomatrici che si implementeranno i controlli necessari per dichiarare ciascuna matrice vietata oppure ammessa.

3.1.3 Controlli implementati

Si richiama di seguito la definizione su cui è basato il criterio di caratterizzazione combinatoria completa:

Definizione 5. Sia $M = C \times D$ una matrice rappresentante una funzione $f(x, y)$. La relazione $\sim$ sulle righe di M viene definita come segue: $x_1, x_2 \in C$ soddisfano $x_1 \sim x_2$ se esiste $y \in D$ tale che $M_{x_1,y} = M_{x_2,y}$. La relazione di equivalenza $\equiv$ sulle righe di M viene definita come la *chiusura transitiva* della relazione $\sim$, ovvero $x_1, x_2 \in C$ soddisfano $x_1 \equiv x_2$ se esistono $z_1, z_2, ..., z_l \in C$ tali che $x_1 \sim z_1 \sim z_2 \sim ... \sim z_l \sim x_2$. Le relazioni $\sim$ e $\equiv$ sono definite analogamente anche sulle colonne della matrice M.

Il criterio verrà applicato alle 3280 matrici derivanti dal procedimento di prefiltraggio già illustrato nella sezione 3.1.1.

Lo pseudocodice 5 presenta la funzione di controllo generale CHECKMTX che riceve la matrice come unico parametro in ingresso. La matrice passa in successione attraverso 4 procedure che eseguono ognuna il controllo su un differente formato di sottomatrici. È sufficiente che una sola tra queste fallisca (restituendo *false*) per uscire dalla funzione CHECKMTX dichiarando vietata la matrice. Viceversa il superamento di tutti i controlli applicati dalle 4 funzioni porterà alla restituzione del valore *true* e alla conseguente marcatura della matrice come ammessa.

La scelta di implementare 4 costrutti *if* uno di seguito all'altro non è casuale, bensì costituisce un pratico modo di sperimentare: è possibile commentare alcuni rami *if* (lasciandone in esecuzione altri) oppure si può variare l'ordine delle 4 procedure di

controllo, osservando se e come cambiano i risultati. Nello pseudocodice 5 i controlli sono stati applicati dalla sottomatrice di formato più ampio e quella di dimensioni più ridotte. Si vedrà in seguito (sezione 4.1.1) come una particolare variazione nell'ordine dei controlli farà emergere un interessante spunto per giungere ad una versione di programma più efficiente.

Pseudocodice 5 funzione CHECKMTX

```
function CHECKMTX(mtx[n][n])
    if CHECKMTX3X3(mtx) then return false;          ▷ VIETATA
    end if
    if CHECKMTX2X3(mtx) then return false;          ▷ VIETATA
    end if
    if CHECKMTX3X2(mtx) then return false;          ▷ VIETATA
    end if
    if CHECKMTX2X2(mtx) then return false;          ▷ VIETATA
    end if
     return true;                                   ▷ AMMESSA
end function
```

Di seguito si descriveranno in dettaglio alcune tra le più rappresentative funzioni di controllo implementate, fornendo per ciascuna un estratto in pseudocodice al fine di farne intuire il meccanismo di funzionamento.

- Pseudocodice 6 - funzione CHECKMTX3X3: si fa uso di due flag. Il primo flag (*equivRow*) viene posto pari a *true* in caso si verifichi l'equivalenza di riga. Si procede a verificare allo stesso modo l'equivalenza a livello di colonna solo nel caso in cui il primo flag sia stato posto a *true*. La funzione restituirà *true* solo quando entrambi i flag saranno stati posti a *true*: la matrice in tal caso sarà marcata come vietata. Altrimenti la funzione ritornerà *false* e la matrice verrà considerata ammessa. Si noti come nello pseudocodice alcune casistiche vengano omesse (...) in quanto saranno identiche alla prima ramificazione (indicata per intero), varieranno solamente gli indici delle righe/colonne della matrice. Si omette inoltre di riportare gli pseoudocodici delle funzioni CHECKMTX2X3 e CHECKMTX3X2 in quanto molto simili nello sviluppo generale, cambierà unicamente il numero di righe/colonne sulle quali effettuare i confronti.

- Pseudocodice 7 - funzione CHECKMTX2X2: le 9 sottomatrici 2×2 rientrano nell'ambito di una casistica già sottoposta ad analisi approfondita nella sezione 2.3.1. In particolare si può sfruttare l'unico pattern che su questo formato di matrici identifica matrici vietate: il pattern [3|1]. Si può utilizzare una matrice

Pseudocodice 6 funzione checkMtx3x3

```
function checkMtx3x3(mtx[n][n])
    equivRow = false;
    equivCol = false;

    for i = 0 to i < 3 do                                          ▷ CHECK ROW
        if mtx[0][i] == mtx[1][i] then          ▷ relazione tra prima e seconda riga
            for j = 0 to j < 3 do
                if mtx[1][j] == mtx[2][j]  or mtx[0][j] == mtx[2][j]  then
                    equivRow = true;
                end if
            end for
        else if mtx[1][i] == mtx[2][i] then      ▷ relazione tra seconda e terza riga
            ...
        else if mtx[0][i] == mtx[2][i] then       ▷ relazione tra prima e terza riga
            ...
        end if
    end for

    if equivRow == true then
        for i = 0 to i < 3 do                                      ▷ CHECK COL
            if mtx[i][0] == mtx[i][1] then       ▷ relazione tra prima e seconda col.
                for j = 0 to j < 3 do
                    if mtx[j][1] == mtx[j][2]  or mtx[j][0] == mtx[j][2]  then
                        equivCol = true;
                    end if
                end for
            else if mtx[i][1] == mtx[i][2] then  ▷ relazione tra seconda e terza col.
                ...
            else if mtx[i][0] == mtx[i][2] then    ▷ relazione tra prima e terza col.
                ...
            end if
        end for
    end if

    if equivRow == true  and equivCol == true then
        return true;                                                ▷ VIETATA
    end if

    return false;                                                   ▷ AMMESSA

end function
```

di appoggio dove indicare le coordinate limite di righe e colonne di ciascuna delle 9 sottomatrici. Si utilizza un costrutto *switch* per contare, per ciascuna sottomatrice, quanti bit di valore pari a 0, 1 o 2 sono presenti. Infine è sufficiente verificare che uno dei contatori sia pari a 3 per ritornare il valore *true* utilizzato per indicare la presenza di una sottomatrice vietata. Altrimenti verrà restituito il valore $false$ (matrice ammessa).

Pseudocodice 7 funzione CHECKMTX2X2

```
function CHECKMTX2X2(mtx[n][n])
    m[9][4] = [                                          ▷ matrice di appoggio
    (0, 1, 0, 1), (0, 1, 0, 2), (0, 1, 1, 2)
    (0, 2, 0, 1), (0, 2, 0, 2), (0, 2, 1, 2),
    (1, 2, 0, 1), (1, 2, 0, 2), (1, 2, 1, 2) ]

    for i = 0 to i < 9 do
        count0 = 0, count1 = 0, count2 = 0;
        for r = m[i][0] to r <= m[i][1] do
            for c = m[i][2] to c <= m[i][3] do
                switch mtx[r][c] do
                    case 0 : count0++;
                    case 1 : count1++;
                    case 2 : count2++;
                if c+1 < m[i][3] then c = m[i][3] - 1;
                end if      ▷ necessario per sottom. 2 × 2 non contigue per colonne
            end for
            if r+1 < m[i][1] then r = m[i][1] - 1;
            end if          ▷ necessario per sottom. 2 × 2 non contigue per righe
        end for
        if count0 == 3  or count1 == 3  or count2 == 3 then
            return true;                                 ▷ VIETATA
        end if
    end for

    return false;                                        ▷ AMMESSA

end function
```

3.2 Analisi qualitativa di funzioni privatamente computabili in logica ternaria

Terminato lo sviluppo del modulo di programma per la determinazione del numero di matrici ammesse in logica ternaria, si è passati all'osservazione diretta di tali matrici. L'obiettivo è trovare risposta ai seguenti quesiti:

- tra le matrici che il programma restituisce in output esistono degli schemi ricorrenti che potrebbero aiutare a raggruppare ed organizzare un tale insieme di elementi?
- è possibile giungere fino al punto di individuare un limitato numero di pattern che aiutino a classificare in modo immediato funzioni privatamente computabili in logica ternaria?

Tali obiettivi guideranno la ricerca che verrà esposta nelle prossime sezioni.

3.2.1 Osservazioni sull'elemento in posizione centrale

Si cerca ora di valorizzare quanto più possibile il lavoro di analisi quantitativa decritto nella sezione 3.1, andando oltre la determinazione dell'esatto numero di funzioni privatamente computabili ricavabili dall'insieme di tutte le funzioni in logica ternaria.

Si è organizzato il flusso del programma per provvedere alla stampa in output dell'intero elenco di funzioni che il programma andava a marcare come ammesse. Un tale elenco è riportato all'appendice A. Le matrici presenti in lista sono state sottoposte ad una serie di analisi statistiche su righe e colonne. Per ogni matrice è stato effettuato il conteggio di quante righe/colonne presentavano bit tutti uguali e quante bit tutti differenti. Ricordando che la definizione di matrice vietata presuppone di andare a verificare la contemporanea presenza di un'equivalenza di riga e di colonna, si è provveduto a contare quante coppie di bit erano presenti rispettivamente a livello di colonna e di riga, nel tentativo di cercare di scoprire una condizione minima oltre la quale la matrice si sarebbe potuta considerare ammessa. Tuttavia nessuna delle analisi appena esposte si è avvicinata a dare risposta ai quesiti posti in precedenza.

Si è dunque preferito cambiare tipologia di approccio, puntando a far emergere pattern ricorrenti dall'osservazione diretta delle matrici, scegliendo un bit della matrice che potesse fungere da riferimento per future osservazioni. In particolare, avendo una matrice 3×3 un unico bit centrale in posizione $M_{1,1}$, si è pensato di scegliere proprio tale bit come punto di riferimento per iniziare le indagini.

In una prima versione dell'output dell'elenco matrici il bit centrale poteva apparire rappresentato dal primo, secondo o terzo bit, vale a dire le lettere A, B, o C. Per far emergere graficamente con più facilità eventuali pattern si è scelto di intercettare

opportunamente il bit in posizione centrale e modificare la funzione di stampa in output per rappresentare il bit centrale (ed ogni sua eventuale ulteriore occorrenza nella matrice) tramite un simbolo diverso. Si è scelto per tale scopo di utilizzare il trattino (−), che compare infatti nell'appendice A, in altre appendici presentate in seguito e nella futura trattazione relativa ai pattern via via individuati. Saranno perciò i bit dell'elemento centrale a determinare la struttura base di ciascun pattern.

3.2.2 Occorrenze del bit in posizione centrale

Evidenziando con un simbolo differente (−) il bit in posizione centrale e le altre sue occorrenze all'interno della matrice, si può notare una certa ripetitività di schemi. Si decide dunque di procedere per affinamenti successivi. Una prima suddivisione consiste nel separare le matrici per numero di occorrenze del bit centrale. Si avranno matrici con una sola occorrenza del bit centrale (ovvero il bit centrale stesso), matrici con due occorrenze del bit centrale e così via fino al caso più numeroso di occorrenze pari a 6. Di seguito si riportano tre esempi di matrici per ciascun raggruppamento individuato:

- Esempi di matrici con 1 occorrenza del bit centrale

ID 433	ID 1136	ID 2360
A A A	**A B B**	**A B A**
B − B	**B − A**	**B − B**
A A A	**A B B**	**B A B**

- Esempi di matrici con 2 occorrenze del bit centrale

ID 379	ID 449	ID 1156
A A A	**A − A**	**A B −**
B − −	**A − A**	**A − B**
A A A	**C C C**	**B A A**

- Esempi di matrici con 3 occorrenze del bit centrale

ID 352	ID 404	ID 456
A A A	**− B B**	**− B −**
− − −	**B − −**	**B − B**
A A A	**C C C**	**C C C**

- Esempi di matrici con 4 occorrenze del bit centrale

ID 393	ID 477	ID 1055
A - -	**- - B**	**- B B**
A - -	**- - B**	**B - -**
C C C	**C C C**	**- B B**

- Esempi di matrici con 5 occorrenze del bit centrale

ID 769	ID 772	ID 2461
- B B	**- B C**	**- B -**
B - -	**C - -**	**B - B**
B - -	**C - -**	**- B -**

- Esempi di matrici con 6 occorrenze del bit centrale

ID 14	ID 15	ID 3515
A A A	**A B B**	**- - B**
- - -	**- - -**	**- - C**
- - -	**- - -**	**- - B**

E' possibile osservare in questa fase come il massimo numero di occorrenze del bit centrale per una matrice ammissibile sia proprio 6. Si tratta di un risultato che fornisce una prima conferma della correttezza del programma e dell'output da questi prodotto. Infatti una matrice con un numero di occorrenze del bit centrale pari a 7 oppure 8 sarebbe certamente vietata, in quanto la matrice presenterà chiaramente equivalenza sia di riga che di colonna (pur non essendo monocromatica).

- Esempi di matrici vietate con 7 oppure 8 occorrenze del bit centrale

A - A	**- - -**
- - -	**- - -**
- - -	**B - -**

Infine il caso di 9 occorrenze del bit centrale è equivalente alla matrice monocromatica. Si tratta di matrice ammessa per definizione e perciò rappresentante funzioni privatamente computabili. Tuttavia non appare nell'output del programma descritto in quanto le sole 3 matrici monocromatiche esistenti in logica ternaria sono già state filtrate in una fase preliminare del programma (tramite la funzione checkMtxMono descritta allo pseudocodice 3).

3.2.3 Pattern ricorrenti emersi

Tramite le osservazioni precedenti si giunge a determinare un limitato numero di pattern, raccolti in figura 9.

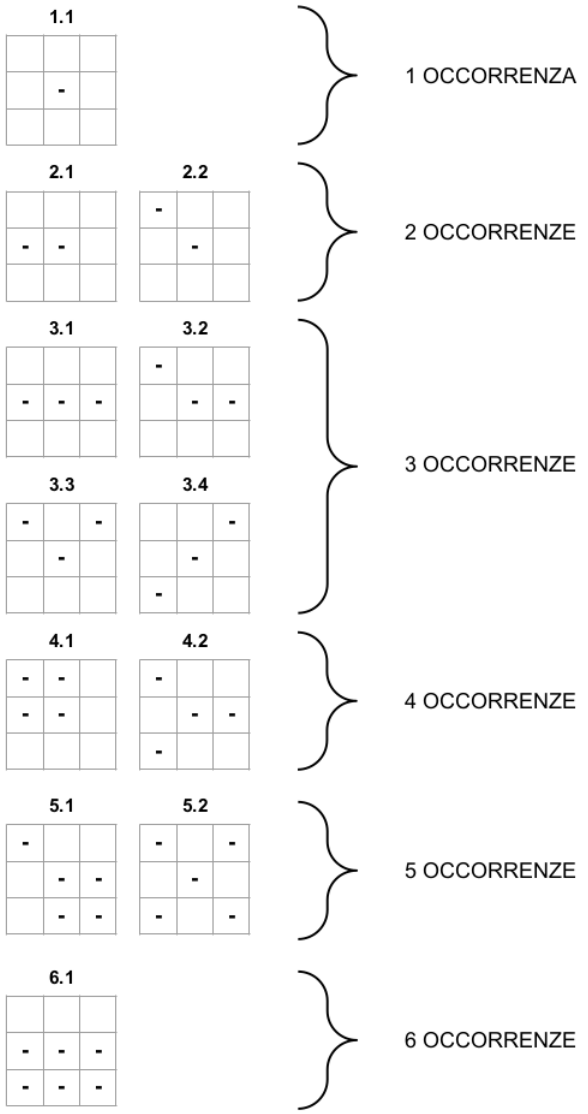


Figura 9: pattern suddivisi per numero di occorrenze dell'elemento centrale

Da notare come, in generale, non si tratti di pattern statici, ovvero da considerare solo nella posizione rappresentata in figura, bensì ciascun pattern può presentarsi ruotato e/o riflesso a seconda delle sue caratteristiche di simmetria.

Vi sono a tal proposito alcuni casi particolari. Ad esempio sui pattern 1.1 e 5.2 una rotazione non provocherebbe alcuna variazione. I pattern 3.1 e 3.4, pur ruotando, possono assumere solo due configurazioni alternative, come mostrato in figura 10.

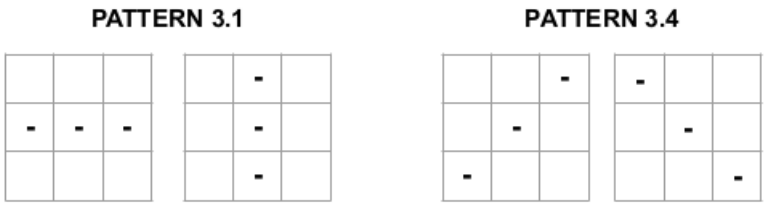


Figura 10: rotazioni possibili sui pattern 3.1 e 3.4

Tutti gli altri pattern, ruotando, danno luogo a 4 configurazioni alternative, ad eccezione del pattern 3.2 che, sottoposto a riflessione, dà luogo ad ulteriori 4 configurazioni, per un totale di 8 (figura 11).

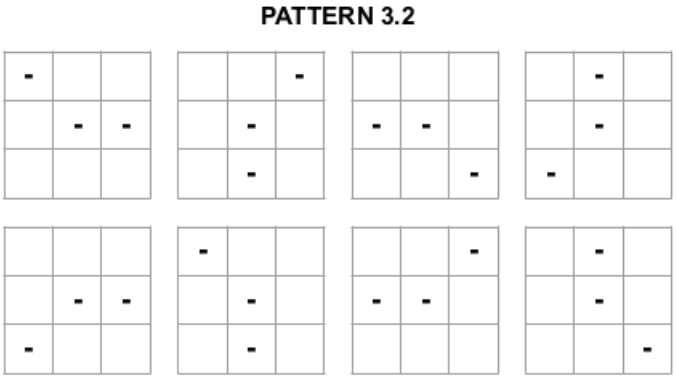


Figura 11: rotazioni e riflessioni possibili sul pattern 3.2

Nel capitolo 4 si esporranno i risultati raggiunti grazie alla determinazione di tali pattern. Si scoprirà che, in realtà, la suddivisione qui introdotta non è sufficiente per giungere ad una caratterizzazione completa dei pattern qui proposti. Sarà invece necessario un ulteriore passaggio che verrà dedotto dalla presentazione dei risultati dell'analisi quantitativa (si consulti la sezione 4.1).

Capitolo 4

Risultati sperimentali raggiunti

In questo capitolo si procederà a presentare i risultati raggiunti grazie al programma sviluppato per lo studio dell'insieme delle funzioni privatamente computabili in logica ternaria. Si vedrà come il raggiungimento di tali risultati porterà ad una riflessione sull'efficienza dell'algoritmo utilizzato. Attraverso alcuni passaggi dimostrativi si ricaverà la base teorica per modificare l'algoritmo, riducendo il numero di sottomatrici da controllare per distinguere tra matrici vietate e ammesse. Inoltre sarà possibile sfruttare questo risultato per concludere la caratterizzazione dei pattern presentati nella sezione 3.2.3. Un gruppo di pattern completamente definito verrà infine presentato. Fungerà da classificatore rapido per tutte le funzioni in logica a 3 valori. Il capitolo si concluderà presentando alcuni esempi di classificazione di funzioni in logica ternaria e alcune ulteriori riflessioni che si potranno trarre dai risultati raggiunti.

4.1 Risultati derivanti dall'analisi quantitativa

È opportuno riepilogare di seguito la cardinalità delle matrici di funzioni in logica a 3 valori dalla quale si è partiti, arrivando infine ad esporre i risultati restituiti in output dal programma sviluppato.

All'analisi sono state sottoposte tutte le matrici di funzioni in logica ternaria:

$$27 \times 27 \times 27 = 19683 \text{ matrici}$$

Sono state immediatamente scartate le 3 matrici monocromatiche (ammesse per definizione). Delle 19680 matrici rimaste è stata calcolata un'apposita signature, come descritto nella sezione 3.1.1. Ogni signature rispecchia la disposizione reciproca dei bit nella matrice. Per ogni signature, è stata sottoposta a controllo una sola matrice (saltando il controllo sulle altre 5 matrici con bit permutati che presentavano la medesima firma).

È stato perciò sufficiente eseguire gli algoritmi di controllo su:

$$\frac{19680}{6} = 3280 \text{ matrici}$$

Procedendo in tal modo e applicando la funzione di controllo generale CHECKMTX indicata allo pseudocodice 5, è stato ottenuto il seguente output:

```
TOTALE MATRICI ANALIZZATE:      3280
MATRICI AMMESSE (PC):            283
MATRICI VIETATE (NON PC):       2997
-----------------------------------
di cui VIETATE 3x3:             2025
di cui VIETATE 2x3:              360
di cui VIETATE 3x2:              360
di cui VIETATE 2x2:              252
```

Per quanto già esposto in precedenza, significa contare un totale di:

- $(283 \times 6) + 3 =$ **1701 matrici ammesse** ($\sim 8,64\%$)
  (rappresentanti funzioni privatamente computabili)

A fronte di:

- $2997 \times 6 =$ **17982 matrici vietate** ($\sim 91,36\%$)
  (rappresentanti funzioni non privatamente computabili)

Si può notare come le 4 funzioni di controllo abbiano tutte inciso nell'individuazione e nel filtraggio delle matrici vietate. La maggior parte di queste (2025 su 2997) sono state dichiarate vietate perchè è stata la matrice principale stessa a presentare l'equivalenza di riga e di colonna. Vi sono però da registrare ulteriori 360 matrici dichiarate vietate perchè hanno almeno una sottomatrice 2×3 vietata, altre 360 matrici dichiarate vietate perchè hanno almeno una sottomatrice 3×2 vietata e, infine, 252 matrici che hanno superato i controlli applicati dalle prime 3 funzioni: queste ultime sono state intercettate dalla funzione CHECKMTX2X2, in quanto presentano almeno una sottomatrice 2×2 vietata.

La presenza di una tale suddivisione merita un'indagine più approfondita. Provare infatti ad alterare l'ordine reciproco delle 4 funzioni di controllo per verificare se e come cambino i numeri permetterà di avere la certezza che non esista una sequenza di controlli più efficiente di quella attualmente adottata.

4.1.1 Aumentare l'efficenza dell'algoritmo

Tra i test eseguiti si è provato a ribaltare l'ordine delle funzioni di controllo, verificando prima le sottomatrici di formato più piccolo 2×2 e spostando in ultima posizione il controllo a livello delle matrice principale 3×3. La funzione CHECKMTXBIS così modificata è visibile allo pseudocodice 8.

Pseudocodice 8 funzione CHECKMTXBIS

```
function CHECKMTXBIS(mtx[n][n])
    if CHECKMTX2X2(mtx) then return false;          ▷ VIETATA
    end if
    if CHECKMTX2X3(mtx) then return false;          ▷ VIETATA
    end if
    if CHECKMTX3X2(mtx) then return false;          ▷ VIETATA
    end if
    if CHECKMTX3X3(mtx) then return false;          ▷ VIETATA
    end if
    return true;                                    ▷ AMMESSA
end function
```

Eseguendo il programma in tale configurazione viene prodotto il seguente output:

```
TOTALE MATRICI ANALIZZATE:      3280
MATRICI AMMESSE (PC):            283
MATRICI VIETATE (NON PC):       2997
-----------------------------------
di cui VIETATE 2x2:             2997
di cui VIETATE 2x3:                0
di cui VIETATE 3x2:                0
di cui VIETATE 3x3:                0
```

Si noti come il risultato sul conteggio finale di matrici ammesse e vietate sia invariato rispetto ai risultati esposti nella sezione 4.1 , tuttavia è cambiata la suddivisione con cui le matrici vietate sono state intercettate. In particolare la funzione CHECKMTX2X2, ora eseguita per prima, è in grado di dichiarare vietate tutte le 2997 matrici che prima venivano intercettate a vari livelli su 4 funzioni differenti. Si possono perciò formulare le seguenti due ipotesi sperimentali:

- le sottomatici 2×3, 3×2 e 3×3 che vengono dichiarate vietate contengono al loro interno sempre almeno una sottomatrice 2×2 vietata;
- il controllo sulle 9 sottomatrici 2×2 di cui una matrice 3×3 si compone è sufficiente per discriminare tra matrici vietate e ammesse.

Dimostrare tali ipotesi si tradurrebbe in un indubbio aumento di efficienza del programma nel suo complesso. Infatti, per ogni matrice da analizzare, le sottomatrici da processare non sarebbe più 16 in differenti formati ma solamente 9 e tutte nel formato 2×2.

Nel seguito si svilupperanno due varianti del programma originario atte a fornire le dimostrazioni necessarie. Quanto si andrà a dimostrare influenzerà anche i risultati della successiva analisi qualitativa, presentata alla sezione 4.2.

- Dimostrazione per sottomatrici 2×3

Una matrice 2×3 come la seguente:

```
1 2 3
4 5 6
```

Presenta al suo interno $\binom{2}{2}\binom{3}{2} = 3$ sottomatrici 2×2:

```
1 2        1 3        2 3
4 5        4 6        5 6
```

È stato sviluppato un programma simile a quello già descritto, opportunamente adattato per generare matrici di formato 2×3. Partendo dalla generazione di un totale di $27 \times 27 = 729$ matrici 2×3, si scartano, tramite apposita funzione, le 3 matrici monocromatiche. Si generano quindi le signature per le rimanenti 726 matrici (con la medesima modalità già descritta allo pseudocodice 4). Si processa una sola matrice per ogni signature, per un totale di $\frac{726}{6} = 121$ matrici da controllare. Lo si può fare applicando la funzione CHECKMTX2X3, restituendo in output le sole matrici vietate. Si verifica (tramite osservazione diretta oppure riprocessando le matrici vietate in output con la funzione CHECKMTX2X2) che ciascuna matrice vietata 2×3 contenga almeno una sottomatrice 2×2.

Di seguito si riportano alcuni esempi, presi a campione, di matrici vietate 2×3 con relativa evidenziazione in rosso dei bit che formano il pattern della sottomatrice vietata 2×2 contenuta. Il simbolo ? sta ad indicare un bit ininfluente nel raggiungimento dell'equivalenza di riga e di colonna. Tale bit può pertanto essere ignorato per lo scopo dimostrativo che ci si prefigge:

```
A A A      A B B      A A B
? A A      A A ?      B ? B
```

La dimostrazione completa è riportata all'appendice B e contiene tutte le casistiche uniche di matrici vietate 2×3 generate dal programma, con relativa evidenziazione in rosso del pattern della sottomatrice vietata 2×2 contenuta. Si noti come la presente dimostrazione sia valida anche per sottomatrici 3×2, è sufficiente invertire righe e colonne sulle matrici ottenute.

- Dimostrazione per sottomatrici 3×3

Una matrice 3×3 come la seguente:

```
1 2 3
4 5 6
7 8 9
```

Presenta al suo interno $\binom{3}{2}\binom{3}{2} = 9$ sottomatrici 2×2:

```
1 2          1 3          2 3
4 5          4 6          5 6

1 2          1 3          2 3
7 8          7 9          8 9

4 5          4 6          5 6
7 8          7 9          8 9
```

In tal caso la variante di programma sviluppata ricalca il programma già descritto alla sezione 3.1, fino al punto in cui le 3280 matrici vengono processate dalla funzione di controllo generica. Questa conterrà al suo interno la sola funzione CHECKMTX3X3 e, come già dedotto dall'output del risultato ottenuto alla sezione 4.1, si otterranno esattamente 2025 matrici vietate 3×3.

La funzione CHECKMTX3X3 è stata modificata per sviluppare un secondo sistema di signature che andasse a distinguere ogni singola configurazione in cui si può ottenere l'equivalenza di riga e l'equivalenza di colonna su una matrice 3×3. Si tratta di un numero intero di 4 cifre associato a ciascuna matrice: le prime due cifre indicano in quale modo viene raggiunta l'equivalenza di riga, le successive due in quale modo viene raggiunta l'equivalenza di colonna.

Si considerino, per esempio, le seguenti 12 matrici che vengono tutte marcate con la medesima signature (1032). I bit che determinano l'equivalenza di riga e di colonna sono stati evidenziati in giallo.

A	B	B	A	B	B	A	B	B	A	B	B
A	B	B	A	B	C	A	C	B	A	C	B
A	A	A	A	A	A	A	A	B	A	A	C
A	B	B	A	B	B	A	B	B	A	B	B
A	C	C	A	B	B	A	B	C	A	C	C
A	A	A	A	A	B	A	A	C	A	A	B
A	B	B	A	B	B	A	B	B	A	B	B
A	C	B	A	C	C	A	B	C	A	B	B
A	A	A	A	A	C	A	A	B	A	A	C

Ai fini della dimostrazione che occorre portare a termine è sufficiente sottoporre a verifica una sola matrice per ogni signature di questo tipo. Per questo motivo la seguente matrice (con signature 1032) sarà la sola ad essere controllata (in rosso si evidenzia la sottomatrice vietata 2×2 contenuta all'interno della matrice principale):

A	B	B
A	?	?
A	A	?

In tal modo è possibile ridurre sensibilmente il numero di matrici da sottoporre a verifica. La dimostrazione completa è consultabile all'appendice C.

4.2 Risultati derivanti dall'analisi qualitativa

L'osservazione delle 283 matrici rappresentanti funzioni privatamente computabili (si consulti l'appendice A) ha già permesso di individuare un limitato numero di pattern, presentati nella sezione 3.2.3. Per giungere ad un risultato concreto (e fare così di questa selezione di pattern uno strumento utilizzabile in un contesto applicativo) occorre verificare che ciascuno schema sia completamente definito e consenta sempre di discriminare tra matrici vietate e matrici ammesse, senza possibilità di errore.

Partendo dal programma già sviluppato risulta ora relativamente semplice andare a filtrare un pattern alla volta tra quelli indicati in figura 9. L'obiettivo è verificare che ciascun pattern identifichi solo matrici ammesse. Diversamente vi rientrerebbero anche matrici vietate e, in questo caso, occorrerà trovare un modo per completare la caratterizzazione del pattern in esame. Si vedrà a tal proposito come quanto dimostrato nella sezione 4.1.1 possa aiutare a colmare alcune lacune rimaste in essere.

4.2.1 Pattern necessari e sufficienti

Si verifica come, tra i 12 pattern già presentati in figura 9, ve ne siano tre che hanno la proprietà di essere sia necessari che sufficienti. In altre parole significa che, in presenza di uno di questi pattern, è subito possibile classificare la matrice come ammessa (ovvero rappresentante una funzione privatamente computabile).

I pattern in questione sono il 5.1, il 5.2 e il 6.1. Sotto tali pattern ricadono un totale di 56 matrici ammesse sulle 283 totali (e nessuna matrice vietata). Tali pattern verranno da questo momento evidenziati con un colore verde.

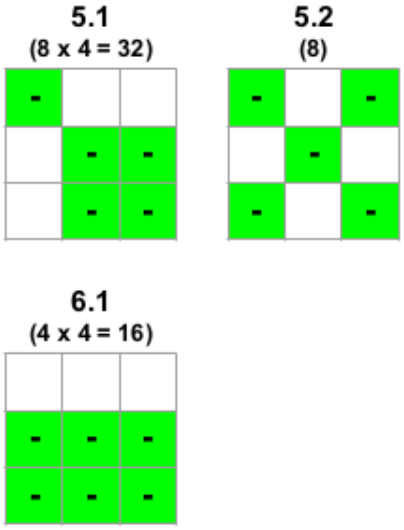


Figura 12: pattern necessari e sufficienti

Si ricorda che ciascun pattern può presentarsi ruotato e/o riflesso (v. sezione 3.2.3), motivo per cui da qui in avanti si provvederà ad indicare anche la moltiplicazione ($\times 2$, $\times 4$, $\times 8$) che dà luogo al computo totale di matrici ammesse che ricadono in ciascun pattern.

4.2.2 Pattern necessari e pattern negativi

Gli altri 9 pattern individuati risultano essere necessari ma non sufficienti per l'individuazione delle rimanenti matrici ammesse. Significa che vi ricadono tutte le restanti matrici ammesse presentate all'appendice A, insieme ad un numero ancor maggiore di matrici vietate. Ciò attribuirebbe a tali schemi scarsa utilità. L'aver però dimostrato nella sezione 4.1.1 che per distinguere tra matrici vietate e matrici ammesse è sufficiente il controllo sulle 9 sottomatrici 2×2 contenute nella matrice principale 3×3 è un risultato che ora torna utile per completare la caratterizzazione di tali pattern.

Si ricorda innanzitutto come, esponendo i risultati noti per funzioni booleane (si consulti a tal proposito la sezione 2.3.1), si è giunti alla conclusione che esiste un solo pattern che identifica matrici vietate in logica binaria: il pattern [3|1], costituito da 3 elementi uguali ed uno solo diverso. Le modalità con cui un tale pattern può manifestarsi in una matrice 3×3 (contenente 9 sottomatrici 2×2) sono esattamente tre, visibili in figura 13.

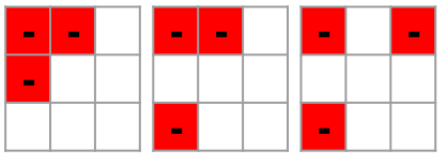


Figura 13: possibili configurazioni del pattern [3|1] in matrici 3×3

Essendo fino ad ora andati alla ricerca di pattern per l'individuazione di matrici ammesse, risulta invece evidente come i pattern in figura 13 assolvano allo scopo esattamente opposto, ovvero identificano matrici vietate. Verranno chiamati perciò *pattern negativi* e si procederà a evidenziarli con un colore rosso. In ciascuno dei tre casi si può ritrovare lo stesso pattern applicando opportune operazioni di traslazione, rotazione o riflessione (o una particolare combinazione tra queste). Si veda a tal proposito la figura 14.

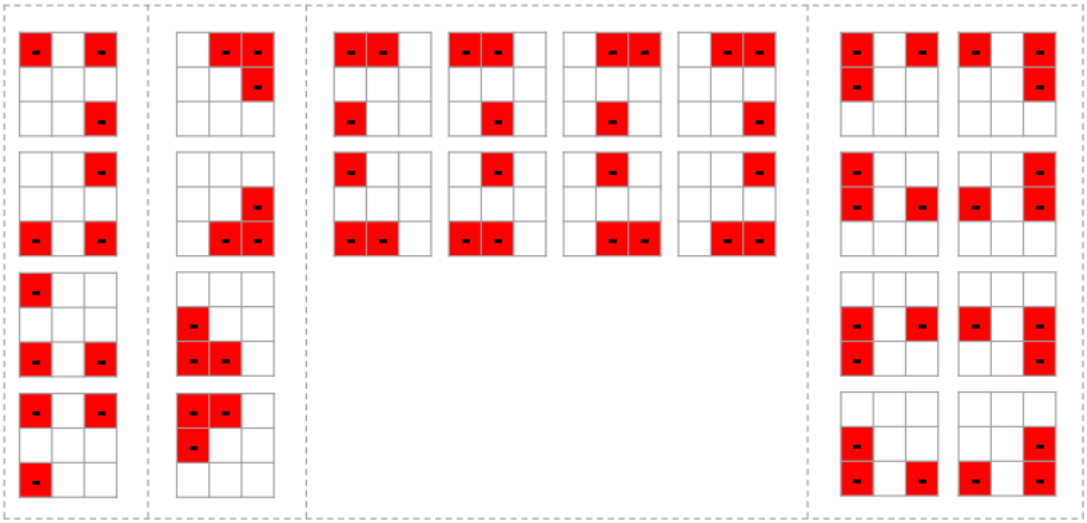


Figura 14: varianti in cui si può presentare il pattern [3|1] in una matrice 3×3

Si noti come tutte le configurazioni dei pattern negativi raccolte in figura 14 non si sovrappongano mai al bit centrale. L'idea è quella di combinare opportunamente i pattern necessari (d'ora in poi evidenziati in giallo) con i pattern negativi appena individuati. Ciò consentirà di concludere la caratterizzazione di ciascuno schema, come si mostrerà di seguito tramite alcuni esempi.

- Pattern 1.1: il pattern necessario è riportato in figura 15.

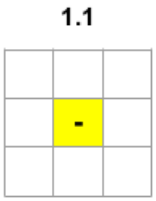


Figura 15: pattern 1.1

Si combina il pattern necessario 1.1 con tutti i pattern negativi che possono presentarsi al suo interno. Si veda a tal proposito la figura 16.

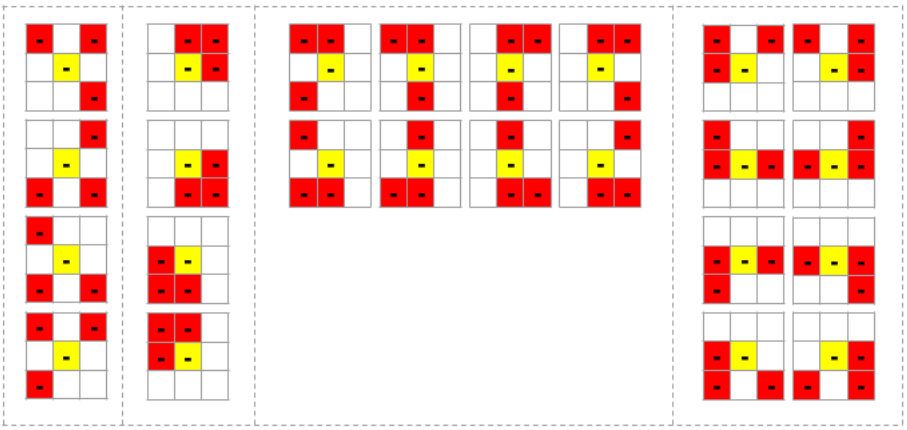


Figura 16: possibili pattern 1.1 negativi

Si conclude che un matrice appartenente al pattern 1.1 sarà vietata se rientra in uno degli schemi evidenziati in figura 16, ammessa altrimenti.

- Pattern 3.3: il pattern necessario è riportato in figura 17.

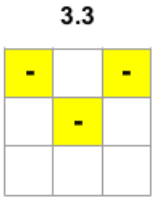


Figura 17: pattern 3.3

Si combina il pattern necessario 3.3 con tutti i pattern negativi che possono presentarsi al suo interno. Si veda a tal proposito la figura 18.

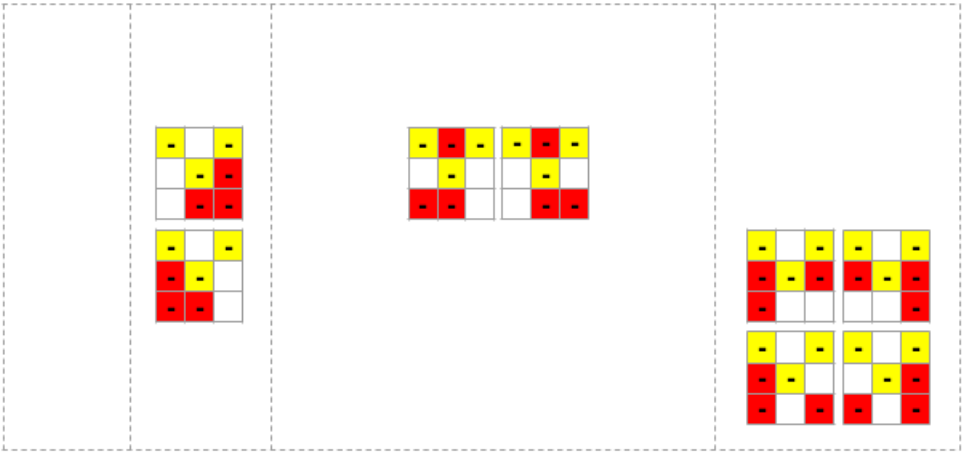


Figura 18: possibili pattern 3.3 negativi

Si conclude che un matrice appartenente al pattern 3.3 sarà vietata se rientra in uno degli schemi evidenziati in figura 18, ammessa altrimenti.

- Pattern 4.1: il pattern necessario è riportato in figura 19.

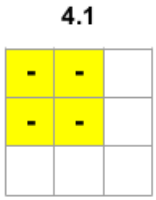


Figura 19: pattern 4.1

Si combina il pattern necessario 4.1 con tutti i pattern negativi che possono presentarsi al suo interno. Si veda a tal proposito la figura 20.

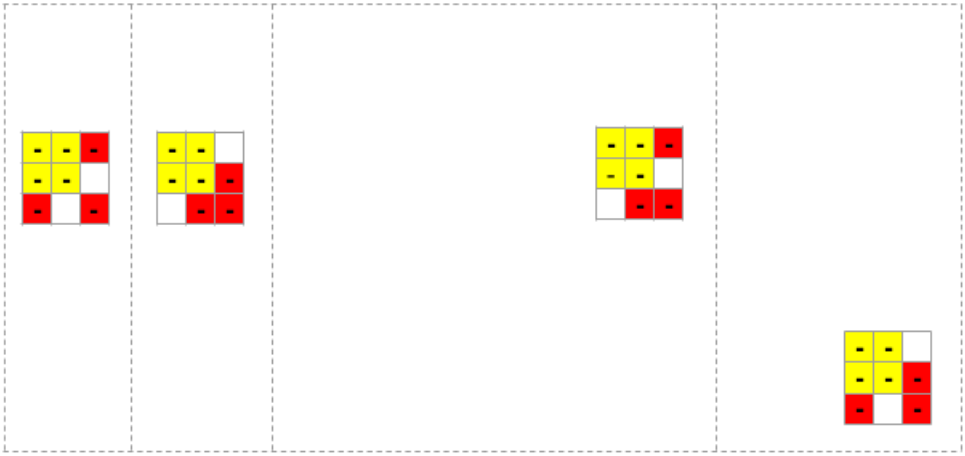


Figura 20: possibili pattern 4.1 negativi

Si conclude che un matrice appartenente al pattern 4.1 sarà vietata se rientra in uno degli schemi evidenziati in figura 20, ammessa altrimenti.

E così via per ciascuno dei 9 pattern necessari individuati. La caratterizzazione completa di ciascuno dei pattern è riportata all'appendice D. Nella sezione 4.3 verranno analizzati esempi reali di funzioni in logica ternaria e come queste possano essere classificate in base ai pattern individuati.

4.2.3 Riepilogo dei 12 pattern completamente caratterizzati

La caratterizzazione completa di tutti i pattern è dunque riepilogabile attraverso la figura 21. Per ciascuno dei 12 pattern vengono indicate quante matrici ammesse (delle 283 totali) ricadono in ciascuno schema.

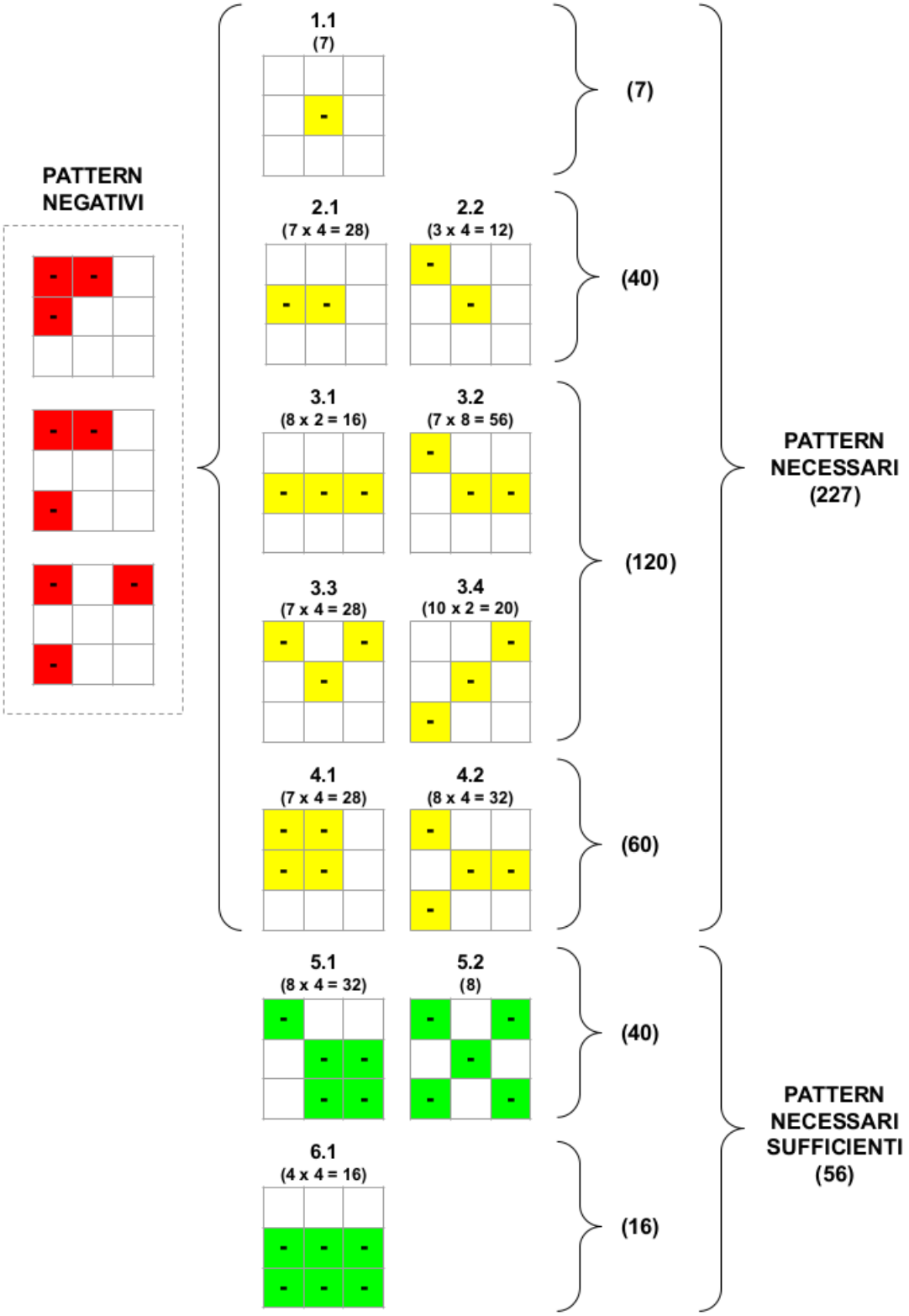


Figura 21: riepilogo dei 12 pattern completamente caratterizzati

Si faccia riferimento all'appendice E per una presentazione dei pattern ad un maggior livello di dettaglio. Si riportano infatti, per ciascun pattern, gli ID delle matrici ammesse che appartengono a quel pattern e si evidenziano alcune varianti individuabili contando le occorrenze dei bit alternativi al bit centrale. Tali varianti possono ulteriormente accelerare la classificazione delle funzioni, come si spiegherà nella prossima sezione 4.3.

4.3 Classificare funzioni in base al pattern

Ora che la suddivisione in pattern è completa, si può utilizzare tale insieme di schemi come un classificatore rapido di funzioni in logica ternaria. Questo procedimento offre diversi vantaggi:

- distinguere tra matrici vietate e matrici ammesse, basandosi sui pattern sviluppati;
- identificare le funzioni privatamente computabili e associarle ad uno specifico pattern;
- dare un'organizzazione alle funzioni privatamente computabili scoperte, raggruppandole in sottoinsiemi ed evidenziando caratteristiche comuni.

Si mostreranno di seguito vari esempi di come le funzioni in logica ternaria possano essere classificate in base agli strumenti ed ai metodi sviluppati. Per le matrici che risulteranno ammesse si arriverà al punto di indicare la variante del pattern (consultare a tal proposito l'appendice E). Per le matrici identificate come vietate verrà messa in evidenza la specifica combinazione di pattern necessari e pattern negativi che hanno portato a determinare quella classificazione (consultare a tal proposito l'appendice D).

Risulta utile partire da un sottoinsieme di funzioni già noto ed analizzato in dettaglio alla sezione 2.3.2: l'insieme delle 24 funzioni lineari rispetto alla somma modulo 3 (si omette qui la trattazione delle tre funzioni monocromatiche 0, 1 e 2, non rientrando queste in alcuno dei pattern sviluppati). Si è già stabilito come la linearità delle funzioni in logica ternaria sia condizione sufficiente per concludere che tali funzioni siano privatamente computabili. Dunque le loro matrici dovrebbero ricadere interamente in uno o più pattern tra quelli individuati in figura 21.

Si cominci considerando la seguente funzione lineare:

$f(x,y) = x$			
3.1	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	0	0
$x = 1$	1	1	1
$x = 2$	2	2	2

$f(x, y) = x$ rientra nel **pattern 3.1 - variante C1**. Si noti come anche le seguenti 5 funzioni vi rientrino: $x +_3 1$, $x +_3 2$, $2x$, $2x +_3 1$, $2x +_3 2$. Si tratta della stessa matrice con i bit permutati nei 6 differenti modi.

Si consideri ora la seguente funzione lineare:

$f(x,y) = y$			
3.1	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	1	2
$x = 1$	0	1	2
$x = 2$	0	1	2

Si tratta ancora del pattern 3.1, semplicemente ruotato in verticale. Dunque $f(x, y) = y$ rientra anch'essa nel **pattern 3.1 - variante C1**. Si noti come anche le seguenti 5 funzioni vi rientrino: $y +_3 1$, $y +_3 2$, $2y$, $2y +_3 1$, $2y +_3 2$. Anche in questo caso si tratta della stessa matrice con i bit permutati nei 6 differenti modi.

È importante notare come il **pattern 3.1 - variante C1**, essendo riscontrabile in sole 2 configurazioni (orizzontale e verticale) si possa manifestare in esattamente 12 funzioni in logica ternaria. In questo caso è stato mostrato come tutte e 12 le funzioni siano state individuate, classificate ed organizzate come da suddivisione suggerita in appendice E.

Si proceda ora a considerare la seguente funzione lineare:

$f(x,y) = x +_3 y$			
3.4	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	1	2
$x = 1$	1	2	0
$x = 2$	2	0	1

$f(x, y) = x +_3 y$ rientra nel **pattern 3.4 - variante B5**. Si noti come anche le seguenti 5 funzioni vi rientrino: $x +_3 y +_3 1$, $x +_3 y +_3 2$, $2x +_3 2y$, $2x +_3 2y +_3 1$, $2x +_3 2y +_3 2$. Si tratta della stessa matrice con i bit permutati nei 6 differenti modi.

Ora si consideri la seguente funzione lineare:

$f(x, y) = 2x +_3 y$			
3.4	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	1	2
$x = 1$	2	0	1
$x = 2$	1	2	0

Anche in questo caso è possibile notare come si tratti ancora del pattern 3.4, semplicemente ruotato sulla diagonale opposta. Dunque $f(x, y) = 2x +_3 y$ rientra anch'essa nel **pattern 3.4 - variante B5**. Si noti come anche le seguenti 5 funzioni vi rientrino: $2x +_3 y +_3 1$, $2x +_3 y +_3 2$, $x +_3 2y$, $x +_3 2y +_3 1$, $x +_3 2y +_3 2$. Anche in questo caso si tratta della stessa matrice con i bit permutati nei 6 differenti modi.

Anche nel caso del **pattern 3.4 - variante B5** valgono le medesime considerazioni fatte per il pattern e la variante analizzati in precedenza. Tutte e 12 le funzioni sono state individuate, classificate ed organizzate come da suddivisione suggerita in appendice E. Significa anche che non esisteranno altre funzioni (e dunque altre matrici) in logica ternaria che potranno ricadere nei 2 pattern e nelle 2 varianti già completamente individuati. Tutte le 24 funzioni lineari non monocromatiche sono state perciò collocate all'interno delle suddivisioni create.

Si tratterebbe ora di procedere per questa strada, identificando tutte la funzioni privatamente computabili che ricadono in ciascun pattern, provando anche ad analizzare funzioni non privatamente computabili, in modo da far emergere differenze e caratteristiche peculiari di ciascun gruppo.

Si procede ad analizzare le funzioni già riportate alla sezione 2.3.2. Si mostrerà come classificare le funzioni riportate in figura 7 e verrà spiegato come arrivare a concludere che le funzioni in figura 8 sono non privatamente computabili.

$f(x, y) = x^2$			
6.1	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	0	0
$x = 1$	1	1	1
$x = 2$	1	1	1

$f(x, y) = x^2 +_3 y^2$			
4.1	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	1	1
$x = 1$	1	2	2
$x = 2$	1	2	2

$f(x, y) = x^2$ rientra nel **pattern 6.1**. Trattandosi di un pattern necessario e sufficiente risulta superfluo in questi casi suddividere in varianti le funzioni che ricadono in tale pattern. È inoltre intuitivo notare come anche la funzione y^2 ricada nello stesso pattern, essendo la stessa matrice ruotata di 90° in senso antiorario.

$f(x, y) = x^2 +_3 y^2$ rientra nel **pattern 4.1 - variante A1**. Anche le seguenti funzioni vi rientrano: $2x^2 +_3 2y^2 + 1$, $2x^2 +_3 2y^2 + 2$, $x(1 -_3 x) +_3 y(1 -_3 y)$.

Si considerino invece le seguenti funzioni:

$f(x, y) = xy$			
NPC	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	0	0
$x = 1$	0	1	2
$x = 2$	0	2	1

$f(x, y) = x^2y$			
NPC	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	0	0
$x = 1$	0	1	2
$x = 2$	0	1	2

$f(x, y) = x^2y^2$			
NPC	$y = 0$	$y = 1$	$y = 2$
$x = 0$	0	0	0
$x = 1$	0	1	1
$x = 2$	0	1	1

Si tratta di 3 funzioni inquadrabili in differenti pattern: $f(x, y) = xy$ nel **pattern 2.2**, $f(x, y) = x^2y$ nel **pattern 2.1** e $f(x, y) = x^2y^2$ nel **pattern 4.1**. Trattandosi però di funzioni che ricadono in pattern necessari – ma non sufficienti – occorre prima controllare che a questi non si possa sovrapporre almeno uno tra i pattern negativi indicati in figura 14. È proprio il caso delle 3 funzioni indicate che, per questo motivo, saranno classificate come non privatamente computabili (**NPC**).

Si fornisce infine la classificazione delle seguenti 4 matrici ammesse, rappresentanti funzioni privatamente computabili:

$f(x,y) = H$			
1.1	$y=0$	$y=1$	$y=2$
$x=0$	0	1	0
$x=1$	1	2	1
$x=2$	0	1	0

$f(x,y) = x^2 +_3 y$			
3.2	$y=0$	$y=1$	$y=2$
$x=0$	0	1	2
$x=1$	1	2	0
$x=2$	1	2	0

$f(x,y) = x^2 +_3 2y^2$			
5.1	$y=0$	$y=1$	$y=2$
$x=0$	0	2	2
$x=1$	1	0	0
$x=2$	1	0	0

$f(x,y) = K$			
5.2	$y=0$	$y=1$	$y=2$
$x=0$	0	1	0
$x=1$	1	0	1
$x=2$	0	1	0

- $f(x,y) = H$, con $H = x(2 -_3 x) +_3 y(2 -_3 y)$: **pattern 1.1 - variante C1**;
- $f(x,y) = x^2 +_3 y$: **pattern 3.2 - variante C2**;
- $f(x,y) = x^2 +_3 2y^2$: **pattern 5.1** (si tratta di uno dei pattern necessari e sufficienti, non occorre specificare alcuna variante);
- $f(x,y) = K$, con $K = x(2 -_3 x) +_3 y(2 -_3 y) +_3 xy(2 -_3 x)(2 -_3 y)$: **pattern 5.2** (si tratta di uno dei pattern necessari e sufficienti, non occorre specificare alcuna variante).

Si noti come, pur avendo eseguito la classificazione su un esiguo numero di funzioni in logica ternaria rispetto all'insieme totale (di cardinalità pari a 19683 funzioni), si possano provare a formulare alcune conclusioni e ipotesi sperimentali (da dimostrare):

- si è confermato che le 24 funzioni lineari in logica ternaria rispetto alla somma modulo 3 sono privatamente computabili. Si è provveduto a classificarle secondo il sistema di pattern sviluppato, mostrando come siano suddivisibili esattamente a metà tra 2 pattern e varianti;
- dalle classificazioni effettuare si ipotizza che anche le funzioni quadratiche possano essere privatamente computabili rispetto alla somma modulo 3;

- si noti come le funzioni che presentano entrambi gli ingressi x, y moltiplicati tra loro risultino non privatamente computabili. Fa eccezione l'ultima funzione classificata, $f(x, y) = K$, con $K = x(2-_3 x)+_3 y(2-_3 y)+_3 xy(2-_3 x)(2-_3 y)$, dove però sono presenti anche altri termini. Questo aspetto andrebbe ulteriormente indagato, per far emergere le caratteristiche peculiari di tali tipologie di funzioni;

- si fa notare infine come esistano molte altre tipologie di funzioni che qui non sono state prese in considerazione: per esempio andrebbero esplorate le porte logiche (*and*, *or*, *xor*, ecc.), che, in ambito di logica a 3 valori, vedono modificarsi di conseguenza le rispettive tabelle di verità.

Capitolo 5

Conclusioni

Il presente elaborato è finalizzato allo studio delle funzioni privatamente computabili, una particolare categoria di funzioni dotate della proprietà di essere calcolabili in una comunicazione tra due parti senza che trapelino informazioni sui rispettivi input. Lo stato dell'arte sul tema mostra come tali oggetti matematici siano impiegabili in molteplici contesti applicativi dal grande potenziale ma ancora poco esplorati: due esempi rappresentativi sono costituiti dalla crittografia visuale e dalla comunicazione tra due parti con impiego di tecniche di secret sharing homomorphism.

Sono dunque stati introdotti concetti, definizioni e teoremi fondamentali. A completamento della base teorica su cui poggerà il resto della trattazione sperimentale sviluppata nell'elaborato sono stati esposti i risultati già noti in letteratura scientifica in merito alle funzioni privatamente computabili in logica binaria e in logica ternaria.

Si è quindi iniziata un'analisi sperimentale approfondita sull'intero insieme delle funzioni in logica ternaria. Sono stati descritti alcuni algoritmi sviluppati allo scopo di giungere alla determinazione dell'esatto numero di funzioni privatamente computabili sul totale delle funzioni in logica ternaria. Si è mostrato come un tale programma potesse costituire un solido punto di partenza per avviare un'analisi qualitativa a livello di singole funzioni e di gruppi di funzioni. In particolare si è osservato come le funzioni privatamente computabili si presentano, provvedendo a sviluppare tecniche, strumenti e metodologie per la loro suddivisione e successiva classificazione sistematica. A tale scopo è stato creato un sistema di 12 pattern basati sull'osservazione del bit in posizione centrale e delle sue occorrenze all'interno della matrice 3×3 che rappresenta ciascuna funzione in logica a 3 valori.

Dai risultati dell'analisi quantitativa sono emerse ulteriori osservazioni che hanno portato a modificare il programma, permettendone un'esecuzione più efficiente. È stato altresì possibile colmare certe lacune emerse in un primo momento sui pattern cui si stava tentando – fino a quel momento invano – di dare una caratterizzazione completa.

Si è mostrato infine in che modo i pattern creati possano essere impiegati per classificare rapidamente e con precisione ogni funzione in logica ternaria. Ciò permette di svolgere in modo organizzato un'attività che altrimenti si sarebbe rivelata ostica, favorendo così lo studio di gruppi di funzioni con caratteristiche simili e cercando di far emergere le peculiarità delle funzioni che godono della proprietà di essere privatamente computabili.

5.1 Possibili sviluppi futuri

Il presente elaborato presenta tecniche e strumenti che si prestano ad essere utilizzati per proseguire nell'attività di ricerca legata alle funzioni privatamente computabili in logica multivalore.

È innanzitutto possibile proseguire la classificazione iniziata alla sezione 4.3, che si conclude riportando alcune ipotesi sperimentali ancora da dimostrare.

Si fa notare inoltre come alcuni passaggi deduttivi (persino il modo in cui i pattern sono stati costruiti) potrebbero mantenersi validi, in tutto o in parte, anche per approcciare lo studio di funzioni in logica superiore a 3. A tal proposito si fornisce qualche spunto su come l'autore imposterebbe lo studio di funzioni privatamente computabili in logica a 4 valori:

- l'analisi quantitativa andrebbe condotta per un insieme esponenzialmente più grande di funzioni: $(4^4)^4 = 4\,294\,967\,296$ matrici in logica a 4 valori. Indirizzare i propri sforzi sull'aumento di efficienza dell'algoritmo è perciò ancor più importante. In questo senso cercare di estendere il ragionamento fatto alla sezione 4.1.1 potrebbe rivelarsi la strada migliore da seguire. Questa tesi contiene già le dimostrazioni che matrici in formato 2×3, 3×2 e 3×3 contengono sempre almeno una sottomatrice 2×2 al loro interno (si consultino a tal proposito le appendici B e C). Si adattino queste dimostrazioni ad una logica a 4 valori e si aggiungano le dimostrazioni per i formati di sottomatrici 2×4, 4×2, 3×4, 4×3 e 4×4. Disporre di queste dimostrazioni significherebbe sviluppare un algoritmo che vada a controllare le sole sottomatrici 2×2 presenti in un matrice 4×4, vale a dire:

$$\binom{4}{2}\binom{4}{2} = 36 \text{ sottomatrici } 2 \times 2$$

Si forniscono di seguito i risultati prodotti da una versione di programma, opportunamente modificata, che, per ciascuna matrice rappresentante una funzione in logica a 4 valori, esegue il controllo sulle sole 36 sottomatrici 2×2:

  - **20 736 352 matrici ammesse** ($\sim 0,48\%$)
    (rappresentanti funzioni privatamente computabili)
  - **4 274 230 944 matrici vietate** ($\sim 99,52\%$)
    (rappresentanti funzioni non privatamente computabili)

- l'analisi qualitativa dovrebbe prevedere la creazione di un sistema di pattern sulla falsariga di quello già proposto per le funzioni in logica ternaria e riassunto in figura 21. Da notare che, non esistendo per le matrici 4×4 un bit centrale, occorrerà scegliere un differente riferimento prima di procedere. Questo potrebbe essere il bit in posizione $M_{0,0}$, suddividendo poi i pattern in base al numero di occorrenze con cui tale bit si presenta in ciascuna matrice. In alternativa si propone di cominciare l'analisi dalla sottomatrice 2×2 in posizione centrale (bit in posizione $M_{1,1}$, $M_{1,2}$, $M_{2,1}$, $M_{2,2}$) e, dall'osservazione di tale sottomatrice, tentare l'individuazione di un insieme di schemi ricorrenti.

Appendici

Appendice A

Elenco delle matrici ammesse

Si riporta di seguito la lista completa delle 283 matrici ammesse che il programma sviluppato restituisce in output.

I bit nelle matrici sono rappresentati dalle lettere A, B e C. Il primo bit (A) può essere associato ai numeri 0, 1 oppure 2. Il secondo bit (B) è abbinabile a uno dei due numeri non ancora assegnati ad A. Al terzo bit (C, se presente) viene abbinato il valore non ancora associato ad A e B.

La lettera associata al bit centrale (quello in posizione $M_{1,1}$) viene sostituita con un trattino ($-$) al fine di rendere più evidente il pattern della matrice. Le altre informazioni utili che accompagnano ciascuna matrice sono:

- un numero progressivo (da 1 a 283) delle matrici in lista;
- un identificativo univoco ID (compreso tra 1 e 19683) che il programma associa ad ogni matrice che genera e processa. Sfruttando tale ID è possibile processare in modo mirato solo alcuni sottoinsiemi di matrici;
- l'indicazione del pattern in cui ricade ciascuna matrice, secondo la suddivisione indicata in figura 9.

1 | ID 14
PATTERN 6.1

	0	1	2
0	A	A	A
1	-	-	-
2	-	-	-

7 | ID 378
PATTERN 3.1

	0	1	2
0	A	A	A
1	-	-	-
2	C	C	C

13 | ID 456
PATTERN 3.3

	0	1	2
0	-	B	-
1	B	-	B
2	C	C	C

19 | ID 769
PATTERN 5.1

	0	1	2
0	-	B	B
1	B	-	-
2	B	-	-

25 | ID 781
PATTERN 5.1

	0	1	2
0	-	B	B
1	C	-	-
2	C	-	-

2 | ID 15
PATTERN 6.1

	0	1	2
0	A	B	B
1	-	-	-
2	-	-	-

8 | ID 379
PATTERN 2.1

	0	1	2
0	A	A	A
1	B	-	-
2	A	A	A

14 | ID 460
PATTERN 2.1

	0	1	2
0	A	A	A
1	-	-	C
2	A	A	A

20 | ID 771
PATTERN 4.1

	0	1	2
0	A	B	B
1	B	-	-
2	B	-	-

26 | ID 782
PATTERN 4.1

	0	1	2
0	A	B	B
1	A	-	-
2	A	-	-

3 | ID 17
PATTERN 6.1

	0	1	2
0	A	B	A
1	-	-	-
2	-	-	-

9 | ID 393
PATTERN 4.1

	0	1	2
0	A	-	-
1	A	-	-
2	C	C	C

15 | ID 477
PATTERN 4.1

	0	1	2
0	-	-	B
1	-	-	B
2	C	C	C

21 | ID 772
PATTERN 5.1

	0	1	2
0	-	B	C
1	C	-	-
2	C	-	-

27 | ID 783
PATTERN 4.1

	0	1	2
0	A	A	A
1	B	-	-
2	B	-	-

4 | ID 18
PATTERN 6.1

	0	1	2
0	A	A	B
1	-	-	-
2	-	-	-

10 | ID 404
PATTERN 3.2

	0	1	2
0	-	B	B
1	B	-	-
2	C	C	C

16 | ID 482
PATTERN 3.2

	0	1	2
0	A	A	-
1	-	-	A
2	C	C	C

22 | ID 774
PATTERN 4.1

	0	1	2
0	A	A	B
1	B	-	-
2	B	-	-

28 | ID 785
PATTERN 6.1

	0	1	2
0	A	-	-
1	C	-	-
2	A	-	-

5 | ID 352
PATTERN 3.1

	0	1	2
0	A	A	A
1	-	-	-
2	A	A	A

11 | ID 433
PATTERN 1.1

	0	1	2
0	A	A	A
1	B	-	B
2	A	A	A

17 | ID 758
PATTERN 6.1

	0	1	2
0	A	-	-
1	A	-	-
2	A	-	-

23 | ID 778
PATTERN 5.1

	0	1	2
0	-	B	C
1	B	-	-
2	B	-	-

29 | ID 786
PATTERN 6.1

	0	1	2
0	A	-	-
1	A	-	-
2	C	-	-

6 | ID 365
PATTERN 6.1

	0	1	2
0	-	-	-
1	-	-	-
2	B	B	B

12 | ID 449
PATTERN 2.1

	0	1	2
0	A	-	A
1	A	-	A
2	C	C	C

18 | ID 759
PATTERN 6.1

	0	1	2
0	A	-	-
1	C	-	-
2	C	-	-

24 | ID 780
PATTERN 4.1

	0	1	2
0	A	B	A
1	B	-	-
2	B	-	-

30 | ID 796
PATTERN 5.1

	0	1	2
0	-	B	B
1	C	-	-
2	B	-	-

31 | ID 798
PATTERN 4.1

	0	1	2
0	A	B	B
1	A	-	-
2	B	-	-

37 | ID 1056
PATTERN 3.2

	0	1	2
0	A	B	B
1	B	-	-
2	-	B	B

43 | ID 1109
PATTERN 4.2

	0	1	2
0	-	B	B
1	C	-	-
2	-	B	B

49 | ID 1136
PATTERN 1.1

	0	1	2
0	A	B	B
1	B	-	A
2	A	B	B

55 | ID 1206
PATTERN 4.1

	0	1	2
0	-	-	B
1	-	-	B
2	B	C	C

32 | ID 799
PATTERN 5.1

	0	1	2
0	-	B	C
1	B	-	-
2	C	-	-

38 | ID 1066
PATTERN 5.1

	0	1	2
0	A	-	-
1	A	-	-
2	-	A	A

44 | ID 1110
PATTERN 3.2

	0	1	2
0	A	B	B
1	A	-	-
2	-	B	B

50 | ID 1137
PATTERN 2.2

	0	1	2
0	-	B	B
1	B	-	C
2	C	B	B

56 | ID 1210
PATTERN 3.2

	0	1	2
0	A	B	-
1	-	-	B
2	B	A	A

33 | ID 805
PATTERN 5.1

	0	1	2
0	-	B	C
1	C	-	-
2	B	-	-

39 | ID 1068
PATTERN 5.1

	0	1	2
0	A	-	-
1	C	-	-
2	-	C	C

45 | ID 1120
PATTERN 5.1

	0	1	2
0	A	-	-
1	C	-	-
2	-	A	A

51 | ID 1150
PATTERN 2.1

	0	1	2
0	A	-	C
1	A	-	C
2	C	A	A

57 | ID 1298
PATTERN 3.3

	0	1	2
0	-	B	B
1	B	-	C
2	-	B	B

34 | ID 808
PATTERN 5.1

	0	1	2
0	-	B	B
1	B	-	-
2	C	-	-

40 | ID 1078
PATTERN 3.2

	0	1	2
0	A	B	B
1	A	-	-
2	-	A	A

46 | ID 1122
PATTERN 5.1

	0	1	2
0	A	-	-
1	A	-	-
2	-	C	C

52 | ID 1156
PATTERN 2.2

	0	1	2
0	A	B	-
1	A	-	B
2	B	A	A

58 | ID 1299
PATTERN 2.2

	0	1	2
0	A	B	B
1	B	-	A
2	-	B	B

35 | ID 809
PATTERN 4.1

	0	1	2
0	A	B	B
1	B	-	-
2	A	-	-

41 | ID 1079
PATTERN 4.2

	0	1	2
0	-	B	B
1	C	-	-
2	-	C	C

47 | ID 1132
PATTERN 3.2

	0	1	2
0	A	B	B
1	B	-	-
2	-	A	A

53 | ID 1158
PATTERN 3.3

	0	1	2
0	-	B	-
1	C	-	B
2	B	C	C

59 | ID 1312
PATTERN 3.4

	0	1	2
0	A	B	-
1	A	-	B
2	-	A	A

36 | ID 1055
PATTERN 4.2

	0	1	2
0	-	B	B
1	B	-	-
2	-	B	B

42 | ID 1080
PATTERN 3.2

	0	1	2
0	A	A	A
1	B	-	-
2	-	B	B

48 | ID 1133
PATTERN 4.2

	0	1	2
0	-	B	B
1	B	-	-
2	-	C	C

54 | ID 1190
PATTERN 2.1

	0	1	2
0	A	B	B
1	-	-	A
2	A	B	B

60 | ID 1314
PATTERN 3.4

	0	1	2
0	A	A	-
1	C	-	A
2	-	C	C

61 | ID 1318
PATTERN 3.2

	0	1	2
0	A	-	C
1	A	-	C
2	-	A	A

67 | ID 1390
PATTERN 2.1

	0	1	2
0	A	B	B
1	A	-	-
2	B	A	A

73 | ID 1417
PATTERN 2.1

	0	1	2
0	A	B	B
1	B	-	-
2	B	A	A

79 | ID 1458
PATTERN 6.1

	0	1	2
0	-	-	-
1	-	-	-
2	B	C	C

85 | ID 2286
PATTERN 3.2

	0	1	2
0	A	A	-
1	C	-	C
2	C	-	C

62 | ID 1352
PATTERN 3.3

	0	1	2
0	-	B	B
1	C	-	C
2	-	B	B

68 | ID 1392
PATTERN 3.2

	0	1	2
0	-	B	B
1	C	-	-
2	B	C	C

74 | ID 1419
PATTERN 3.2

	0	1	2
0	-	B	B
1	B	-	-
2	B	C	C

80 | ID 2272
PATTERN 3.1

	0	1	2
0	A	-	A
1	A	-	A
2	A	-	A

86 | ID 2288
PATTERN 3.2

	0	1	2
0	-	B	C
1	B	-	B
2	B	-	B

63 | ID 1366
PATTERN 3.4

	0	1	2
0	A	B	-
1	B	-	B
2	-	A	A

69 | ID 1402
PATTERN 4.1

	0	1	2
0	A	-	-
1	A	-	-
2	C	A	A

75 | ID 1429
PATTERN 4.1

	0	1	2
0	A	-	-
1	C	-	-
2	C	A	A

81 | ID 2275
PATTERN 2.1

	0	1	2
0	A	B	A
1	A	-	A
2	A	-	A

87 | ID 2289
PATTERN 2.1

	0	1	2
0	A	B	A
1	B	-	B
2	B	-	B

64 | ID 1374
PATTERN 3.2

	0	1	2
0	A	-	A
1	A	-	A
2	-	C	C

70 | ID 1403
PATTERN 4.1

	0	1	2
0	A	-	-
1	C	-	-
2	A	C	C

76 | ID 1430
PATTERN 4.1

	0	1	2
0	A	-	-
1	A	-	-
2	A	C	C

82 | ID 2279
PATTERN 4.2

	0	1	2
0	-	B	-
1	B	-	B
2	B	-	B

88 | ID 2292
PATTERN 3.1

	0	1	2
0	A	-	A
1	C	-	C
2	C	-	C

65 | ID 1379
PATTERN 2.1

	0	1	2
0	A	B	B
1	B	-	-
2	A	B	B

71 | ID 1406
PATTERN 2.1

	0	1	2
0	A	B	B
1	A	-	-
2	A	B	B

77 | ID 1433
PATTERN 3.1

	0	1	2
0	A	B	B
1	-	-	-
2	A	B	B

83 | ID 2280
PATTERN 3.2

	0	1	2
0	A	B	-
1	B	-	B
2	B	-	B

89 | ID 2294
PATTERN 3.2

	0	1	2
0	-	B	B
1	C	-	C
2	C	-	C

66 | ID 1380
PATTERN 3.2

	0	1	2
0	-	B	B
1	B	-	-
2	C	B	B

72 | ID 1407
PATTERN 3.2

	0	1	2
0	-	B	B
1	C	-	-
2	C	B	B

78 | ID 1444
PATTERN 3.1

	0	1	2
0	A	B	B
1	-	-	-
2	B	A	A

84 | ID 2285
PATTERN 4.2

	0	1	2
0	-	B	-
1	C	-	C
2	C	-	C

90 | ID 2295
PATTERN 2.1

	0	1	2
0	A	A	A
1	B	-	B
2	B	-	B

91 | ID 2353
PATTERN 1.1

	0	1	2
0	A	B	A
1	A	-	A
2	A	B	A

97 | ID 2370
PATTERN 3.3

	0	1	2
0	-	B	-
1	B	-	B
2	B	C	B

103 | ID 2478
PATTERN 4.2

	0	1	2
0	A	-	A
1	C	-	C
2	-	C	-

109 | ID 2504
PATTERN 4.2

	0	1	2
0	A	-	C
1	C	-	A
2	-	A	-

115 | ID 2640
PATTERN 3.3

	0	1	2
0	-	B	-
1	C	-	C
2	B	C	B

92 | ID 2356
PATTERN 2.1

	0	1	2
0	A	-	A
1	A	-	A
2	A	C	A

98 | ID 2373
PATTERN 3.3

	0	1	2
0	-	B	-
1	C	-	C
2	C	B	C

104 | ID 2481
PATTERN 3.3

	0	1	2
0	A	B	A
1	B	-	B
2	-	B	-

110 | ID 2510
PATTERN 3.3

	0	1	2
0	A	B	B
1	B	-	A
2	-	A	-

116 | ID 2643
PATTERN 3.3

	0	1	2
0	-	B	-
1	B	-	B
2	C	B	C

93 | ID 2360
PATTERN 1.1

	0	1	2
0	A	B	A
1	B	-	B
2	B	A	B

99 | ID 2461
PATTERN 5.2

	0	1	2
0	-	B	-
1	B	-	B
2	-	B	-

105 | ID 2484
PATTERN 3.3

	0	1	2
0	A	A	A
1	B	-	B
2	-	B	-

111 | ID 2623
PATTERN 1.1

	0	1	2
0	A	B	A
1	B	-	B
2	A	B	A

117 | ID 2650
PATTERN 2.1

	0	1	2
0	A	B	A
1	-	-	B
2	A	B	A

94 | ID 2361
PATTERN 2.2

	0	1	2
0	-	B	C
1	B	-	B
2	B	C	B

100 | ID 2464
PATTERN 5.2

	0	1	2
0	-	B	-
1	C	-	C
2	-	C	-

106 | ID 2488
PATTERN 5.2

	0	1	2
0	-	B	-
1	C	-	B
2	-	B	-

112 | ID 2626
PATTERN 2.1

	0	1	2
0	A	-	A
1	C	-	C
2	A	C	A

118 | ID 2664
PATTERN 4.1

	0	1	2
0	-	-	B
1	-	-	B
2	C	B	C

95 | ID 2366
PATTERN 2.1

	0	1	2
0	A	-	A
1	C	-	C
2	C	A	C

101 | ID 2468
PATTERN 4.2

	0	1	2
0	A	-	A
1	A	-	A
2	-	A	-

107 | ID 2491
PATTERN 5.2

	0	1	2
0	-	B	-
1	B	-	C
2	-	C	-

113 | ID 2630
PATTERN 1.1

	0	1	2
0	A	B	A
1	A	-	A
2	B	A	B

119 | ID 2666
PATTERN 3.2

	0	1	2
0	A	B	-
1	-	-	A
2	B	A	B

96 | ID 2369
PATTERN 2.2

	0	1	2
0	A	B	-
1	B	-	B
2	B	A	B

102 | ID 2474
PATTERN 3.3

	0	1	2
0	A	B	A
1	A	-	A
2	-	A	-

108 | ID 2496
PATTERN 4.2

	0	1	2
0	A	-	C
1	A	-	C
2	-	C	-

114 | ID 2636
PATTERN 2.1

	0	1	2
0	A	-	A
1	A	-	A
2	C	A	C

120 | ID 2704
PATTERN 5.2

	0	1	2
0	-	B	-
1	B	-	C
2	-	B	-

121 | ID 2707
PATTERN 5.2

	0	1	2
0	-	B	-
1	C	-	B
2	-	C	-

127 | ID 2738
PATTERN 4.2

	0	1	2
0	A	-	A
1	C	-	C
2	-	A	-

133 | ID 2819
PATTERN 4.2

	0	1	2
0	-	B	-
1	C	-	C
2	B	-	B

139 | ID 2888
PATTERN 4.1

	0	1	2
0	A	-	-
1	A	-	-
2	C	A	C

145 | ID 3034
PATTERN 5.1

	0	1	2
0	A	A	-
1	-	-	A
2	-	-	A

122 | ID 2712
PATTERN 4.2

	0	1	2
0	A	-	C
1	C	-	A
2	-	C	-

128 | ID 2744
PATTERN 3.3

	0	1	2
0	A	B	A
1	B	-	B
2	-	A	-

134 | ID 2825
PATTERN 4.2

	0	1	2
0	-	B	-
1	B	-	B
2	C	-	C

140 | ID 2893
PATTERN 3.1

	0	1	2
0	A	B	A
1	-	-	-
2	A	B	A

146 | ID 3036
PATTERN 5.1

	0	1	2
0	A	B	-
1	-	-	B
2	-	-	B

123 | ID 2718
PATTERN 3.3

	0	1	2
0	A	A	B
1	B	-	A
2	-	B	-

129 | ID 2748
PATTERN 4.2

	0	1	2
0	A	-	A
1	A	-	A
2	-	C	-

135 | ID 2829
PATTERN 2.1

	0	1	2
0	A	B	A
1	A	-	A
2	B	-	B

141 | ID 2900
PATTERN 3.1

	0	1	2
0	A	B	A
1	-	-	-
2	B	A	B

147 | ID 3040
PATTERN 5.1

	0	1	2
0	A	B	-
1	-	-	A
2	-	-	A

124 | ID 2720
PATTERN 4.2

	0	1	2
0	A	-	C
1	A	-	C
2	-	A	-

130 | ID 2751
PATTERN 3.3

	0	1	2
0	A	B	A
1	A	-	A
2	-	B	-

136 | ID 2832
PATTERN 3.1

	0	1	2
0	A	-	A
1	A	-	A
2	C	-	C

142 | ID 2916
PATTERN 6.1

	0	1	2
0	-	-	-
1	-	-	-
2	B	C	B

148 | ID 3042
PATTERN 5.1

	0	1	2
0	A	A	-
1	-	-	C
2	-	-	C

125 | ID 2731
PATTERN 5.2

	0	1	2
0	-	B	-
1	C	-	C
2	-	B	-

131 | ID 2812
PATTERN 3.1

	0	1	2
0	A	-	A
1	C	-	C
2	A	-	A

137 | ID 2866
PATTERN 2.1

	0	1	2
0	A	B	A
1	B	-	-
2	A	B	A

143 | ID 3029
PATTERN 6.1

	0	1	2
0	-	-	B
1	-	-	B
2	-	-	B

149 | ID 3043
PATTERN 4.1

	0	1	2
0	A	A	B
1	-	-	A
2	-	-	A

126 | ID 2734
PATTERN 5.2

	0	1	2
0	-	B	-
1	B	-	B
2	-	C	-

132 | ID 2815
PATTERN 2.1

	0	1	2
0	A	B	A
1	B	-	B
2	A	-	A

138 | ID 2874
PATTERN 3.2

	0	1	2
0	-	B	C
1	C	-	-
2	B	C	B

144 | ID 3033
PATTERN 4.1

	0	1	2
0	A	A	B
1	-	-	B
2	-	-	B

150 | ID 3045
PATTERN 4.1

	0	1	2
0	A	B	A
1	-	-	B
2	-	-	B

151 | ID 3047
PATTERN 6.1

	0	1	2
0	-	-	B
1	-	-	C
2	-	-	C

157 | ID 3150
PATTERN 4.1

	0	1	2
0	-	-	B
1	-	-	C
2	B	B	C

163 | ID 3177
PATTERN 3.2

	0	1	2
0	A	A	B
1	-	-	B
2	B	B	-

169 | ID 3229
PATTERN 3.4

	0	1	2
0	-	B	C
1	B	-	C
2	C	C	-

175 | ID 3342
PATTERN 3.3

	0	1	2
0	-	B	-
1	B	-	C
2	C	C	B

152 | ID 3049
PATTERN 4.1

	0	1	2
0	A	B	B
1	-	-	A
2	-	-	A

158 | ID 3151
PATTERN 3.2

	0	1	2
0	A	A	-
1	-	-	A
2	C	C	A

164 | ID 3178
PATTERN 5.1

	0	1	2
0	-	-	B
1	-	-	C
2	C	C	-

170 | ID 3236
PATTERN 2.2

	0	1	2
0	A	A	B
1	B	-	A
2	A	A	-

176 | ID 3344
PATTERN 2.2

	0	1	2
0	A	A	-
1	C	-	A
2	A	A	C

153 | ID 3051
PATTERN 4.1

	0	1	2
0	A	A	A
1	-	-	C
2	-	-	C

159 | ID 3155
PATTERN 3.2

	0	1	2
0	A	A	-
1	-	-	C
2	A	A	C

165 | ID 3182
PATTERN 3.2

	0	1	2
0	A	A	B
1	-	-	A
2	A	A	-

171 | ID 3238
PATTERN 3.4

	0	1	2
0	-	B	B
1	B	-	C
2	C	C	-

177 | ID 3380
PATTERN 2.1

	0	1	2
0	A	A	B
1	-	-	A
2	A	A	B

154 | ID 3137
PATTERN 2.1

	0	1	2
0	A	A	B
1	-	-	B
2	A	A	B

160 | ID 3164
PATTERN 4.2

	0	1	2
0	A	A	-
1	-	-	A
2	A	A	-

166 | ID 3186
PATTERN 3.2

	0	1	2
0	A	A	A
1	-	-	C
2	C	C	-

172 | ID 3326
PATTERN 1.1

	0	1	2
0	A	A	B
1	B	-	A
2	A	A	B

178 | ID 3384
PATTERN 4.1

	0	1	2
0	-	-	B
1	-	-	C
2	C	C	B

155 | ID 3141
PATTERN 4.1

	0	1	2
0	-	-	B
1	-	-	B
2	C	C	B

161 | ID 3168
PATTERN 4.2

	0	1	2
0	A	A	-
1	-	-	C
2	C	C	-

167 | ID 3218
PATTERN 3.3

	0	1	2
0	A	A	-
1	C	-	A
2	A	A	-

173 | ID 3333
PATTERN 2.2

	0	1	2
0	-	B	C
1	B	-	C
2	C	C	B

179 | ID 3385
PATTERN 2.1

	0	1	2
0	A	A	B
1	-	-	B
2	B	B	A

156 | ID 3142
PATTERN 2.1

	0	1	2
0	A	A	B
1	-	-	A
2	B	B	A

162 | ID 3169
PATTERN 5.1

	0	1	2
0	-	-	B
1	-	-	B
2	B	B	-

168 | ID 3225
PATTERN 3.2

	0	1	2
0	A	-	C
1	A	-	C
2	C	C	-

174 | ID 3337
PATTERN 2.1

	0	1	2
0	A	-	C
1	A	-	C
2	C	C	A

180 | ID 3393
PATTERN 4.1

	0	1	2
0	-	-	B
1	-	-	B
2	B	B	C

181 | ID 3394
PATTERN 3.2

	0	1	2
0	A	A	-
1	-	-	C
2	C	C	A

182 | ID 3398
PATTERN 3.2

	0	1	2
0	A	A	-
1	-	-	A
2	A	A	C

183 | ID 3407
PATTERN 4.2

	0	1	2
0	A	A	-
1	-	-	C
2	A	A	-

184 | ID 3411
PATTERN 4.2

	0	1	2
0	A	A	-
1	-	-	A
2	C	C	-

185 | ID 3412
PATTERN 5.1

	0	1	2
0	-	-	B
1	-	-	C
2	B	B	-

186 | ID 3420
PATTERN 3.2

	0	1	2
0	A	A	B
1	-	-	A
2	B	B	-

187 | ID 3421
PATTERN 5.1

	0	1	2
0	-	-	B
1	-	-	B
2	C	C	-

188 | ID 3425
PATTERN 3.2

	0	1	2
0	A	A	B
1	-	-	B
2	A	A	-

189 | ID 3461
PATTERN 3.3

	0	1	2
0	A	A	-
1	C	-	C
2	A	A	-

190 | ID 3472
PATTERN 3.4

	0	1	2
0	-	B	C
1	B	-	B
2	C	C	-

191 | ID 3477
PATTERN 3.2

	0	1	2
0	A	-	A
1	A	-	A
2	C	C	-

192 | ID 3515
PATTERN 6.1

	0	1	2
0	-	-	B
1	-	-	C
2	-	-	B

193 | ID 3519
PATTERN 4.1

	0	1	2
0	A	A	B
1	-	-	A
2	-	-	B

194 | ID 3520
PATTERN 5.1

	0	1	2
0	A	A	-
1	-	-	C
2	-	-	A

195 | ID 3522
PATTERN 5.1

	0	1	2
0	A	B	-
1	-	-	A
2	-	-	B

196 | ID 3526
PATTERN 5.1

	0	1	2
0	A	B	-
1	-	-	B
2	-	-	A

197 | ID 3528
PATTERN 5.1

	0	1	2
0	A	A	-
1	-	-	A
2	-	-	C

198 | ID 3529
PATTERN 4.1

	0	1	2
0	A	A	B
1	-	-	B
2	-	-	A

199 | ID 3533
PATTERN 6.1

	0	1	2
0	-	-	B
1	-	-	B
2	-	-	C

200 | ID 3569
PATTERN 2.1

	0	1	2
0	A	A	B
1	B	-	-
2	A	A	B

201 | ID 3576
PATTERN 3.2

	0	1	2
0	-	B	C
1	B	-	-
2	C	C	B

202 | ID 3589
PATTERN 4.1

	0	1	2
0	A	-	-
1	A	-	-
2	C	C	A

203 | ID 3623
PATTERN 3.1

	0	1	2
0	A	A	B
1	-	-	-
2	A	A	B

204 | ID 3628
PATTERN 3.1

	0	1	2
0	A	A	B
1	-	-	-
2	B	B	A

205 | ID 3645
PATTERN 6.1

	0	1	2
0	-	-	-
1	-	-	-
2	B	B	C

206 | ID 3786
PATTERN 3.1

	0	1	2
0	A	-	C
1	A	-	C
2	A	-	C

207 | ID 3788
PATTERN 3.2

	0	1	2
0	-	B	C
1	B	-	C
2	B	-	C

208 | ID 3790
PATTERN 3.2

	0	1	2
0	A	A	-
1	C	-	A
2	C	-	A

209 | ID 3791
PATTERN 4.2

	0	1	2
0	-	B	-
1	C	-	B
2	C	-	B

210 | ID 3792
PATTERN 3.2

	0	1	2
0	A	B	-
1	A	-	B
2	A	-	B

211 | ID 3796
PATTERN 3.2

	0	1	2
0	A	B	-
1	B	-	A
2	B	-	A

217 | ID 3806
PATTERN 3.2

	0	1	2
0	-	B	B
1	B	-	C
2	B	-	C

223 | ID 3853
PATTERN 3.4

	0	1	2
0	A	A	-
1	C	-	A
2	-	C	A

229 | ID 3907
PATTERN 5.1

	0	1	2
0	-	-	B
1	-	-	C
2	B	C	-

235 | ID 3940
PATTERN 3.4

	0	1	2
0	-	B	B
1	C	-	C
2	B	C	-

212 | ID 3797
PATTERN 4.2

	0	1	2
0	-	B	-
1	B	-	C
2	B	-	C

218 | ID 3840
PATTERN 3.3

	0	1	2
0	-	B	C
1	B	-	C
2	-	B	C

224 | ID 3854
PATTERN 3.4

	0	1	2
0	A	B	-
1	A	-	B
2	-	A	B

230 | ID 3914
PATTERN 3.2

	0	1	2
0	A	B	B
1	-	-	A
2	B	A	-

236 | ID 3948
PATTERN 3.3

	0	1	2
0	A	B	-
1	A	-	B
2	A	B	-

213 | ID 3799
PATTERN 2.1

	0	1	2
0	A	A	B
1	B	-	A
2	B	-	A

219 | ID 3842
PATTERN 3.2

	0	1	2
0	A	-	C
1	A	-	C
2	-	A	C

225 | ID 3856
PATTERN 3.4

	0	1	2
0	A	B	-
1	B	-	A
2	-	B	A

231 | ID 3921
PATTERN 3.3

	0	1	2
0	A	B	-
1	B	-	B
2	A	B	-

237 | ID 3950
PATTERN 3.3

	0	1	2
0	A	B	-
1	B	-	A
2	B	A	-

214 | ID 3800
PATTERN 3.2

	0	1	2
0	-	B	C
1	C	-	B
2	C	-	B

220 | ID 3844
PATTERN 2.2

	0	1	2
0	A	A	B
1	B	-	A
2	-	B	A

226 | ID 3894
PATTERN 4.2

	0	1	2
0	A	B	-
1	-	-	B
2	A	B	-

232 | ID 3926
PATTERN 3.2

	0	1	2
0	A	-	A
1	A	-	A
2	C	A	-

238 | ID 3954
PATTERN 3.2

	0	1	2
0	A	-	C
1	A	-	C
2	A	C	-

215 | ID 3802
PATTERN 3.1

	0	1	2
0	A	-	C
1	C	-	A
2	C	-	A

221 | ID 3846
PATTERN 3.3

	0	1	2
0	-	B	C
1	C	-	B
2	-	C	B

227 | ID 3896
PATTERN 4.2

	0	1	2
0	A	B	-
1	-	-	A
2	B	A	-

233 | ID 3932
PATTERN 2.2

	0	1	2
0	A	B	A
1	A	-	A
2	B	A	-

239 | ID 3958
PATTERN 3.4

	0	1	2
0	-	B	C
1	B	-	C
2	B	C	-

216 | ID 3805
PATTERN 2.1

	0	1	2
0	A	B	B
1	B	-	A
2	B	-	A

222 | ID 3850
PATTERN 3.2

	0	1	2
0	A	-	C
1	C	-	A
2	-	C	A

228 | ID 3898
PATTERN 5.1

	0	1	2
0	-	-	B
1	-	-	B
2	C	B	-

234 | ID 3937
PATTERN 3.4

	0	1	2
0	-	B	C
1	B	-	B
2	C	B	-

240 | ID 3962
PATTERN 3.2

	0	1	2
0	A	-	C
1	C	-	A
2	C	A	-

241 | ID 3964
PATTERN 3.4

	0	1	2
0	-	B	C
1	C	-	B
2	C	B	-

247 | ID 4066
PATTERN 3.2

	0	1	2
0	A	-	C
1	A	-	C
2	-	C	A

253 | ID 4096
PATTERN 3.4

	0	1	2
0	A	A	-
1	C	-	C
2	-	C	A

259 | ID 4150
PATTERN 5.1

	0	1	2
0	-	-	B
1	-	-	B
2	B	C	-

265 | ID 4180
PATTERN 3.4

	0	1	2
0	-	B	C
1	B	-	C
2	C	B	-

242 | ID 3967
PATTERN 3.4

	0	1	2
0	-	B	B
1	B	-	C
2	B	C	-

248 | ID 4070
PATTERN 3.4

	0	1	2
0	A	B	-
1	B	-	A
2	-	A	B

254 | ID 4099
PATTERN 3.4

	0	1	2
0	A	B	-
1	B	-	B
2	-	B	A

260 | ID 4164
PATTERN 3.3

	0	1	2
0	A	B	-
1	B	-	A
2	A	B	-

266 | ID 4218
PATTERN 3.1

	0	1	2
0	A	-	C
1	C	-	A
2	A	-	C

243 | ID 3968
PATTERN 2.2

	0	1	2
0	A	B	B
1	B	-	A
2	B	A	-

249 | ID 4072
PATTERN 3.4

	0	1	2
0	A	B	-
1	A	-	B
2	-	B	A

255 | ID 4137
PATTERN 4.2

	0	1	2
0	A	B	-
1	-	-	A
2	A	B	-

261 | ID 4166
PATTERN 3.3

	0	1	2
0	A	B	-
1	A	-	B
2	B	A	-

267 | ID 4220
PATTERN 3.2

	0	1	2
0	-	B	C
1	C	-	B
2	B	-	C

244 | ID 4056
PATTERN 3.3

	0	1	2
0	-	B	C
1	C	-	B
2	-	B	C

250 | ID 4083
PATTERN 3.3

	0	1	2
0	-	B	C
1	B	-	B
2	-	B	C

256 | ID 4139
PATTERN 4.2

	0	1	2
0	A	B	-
1	-	-	B
2	B	A	-

262 | ID 4170
PATTERN 3.2

	0	1	2
0	A	-	C
1	C	-	A
2	A	C	-

268 | ID 4223
PATTERN 4.2

	0	1	2
0	-	B	-
1	B	-	C
2	C	-	B

245 | ID 4058
PATTERN 3.2

	0	1	2
0	A	-	C
1	C	-	A
2	-	A	C

251 | ID 4088
PATTERN 2.2

	0	1	2
0	A	B	A
1	A	-	A
2	-	A	B

257 | ID 4141
PATTERN 5.1

	0	1	2
0	-	-	B
1	-	-	C
2	C	B	-

263 | ID 4174
PATTERN 3.4

	0	1	2
0	-	B	C
1	C	-	B
2	B	C	-

269 | ID 4224
PATTERN 3.2

	0	1	2
0	A	B	-
1	B	-	A
2	A	-	B

246 | ID 4062
PATTERN 3.3

	0	1	2
0	-	B	C
1	B	-	C
2	-	C	B

252 | ID 4094
PATTERN 3.2

	0	1	2
0	A	-	A
1	A	-	A
2	-	A	C

258 | ID 4148
PATTERN 3.2

	0	1	2
0	A	B	A
1	-	-	B
2	B	A	-

264 | ID 4178
PATTERN 3.2

	0	1	2
0	A	-	C
1	A	-	C
2	C	A	-

270 | ID 4228
PATTERN 3.2

	0	1	2
0	A	B	-
1	A	-	B
2	B	-	A

271 | ID 4229
PATTERN 4.2

	0	1	2
0	-	B	-
1	C	-	B
2	B	-	C

---

272 | ID 4232
PATTERN 3.2

	0	1	2
0	-	B	C
1	B	-	C
2	C	-	B

---

273 | ID 4234
PATTERN 3.1

	0	1	2
0	A	-	C
1	A	-	C
2	C	-	A

---

274 | ID 4299
PATTERN 4.2

	0	1	2
0	-	B	C
1	C	-	-
2	-	B	C

---

275 | ID 4304
PATTERN 3.2

	0	1	2
0	A	B	A
1	B	-	-
2	-	A	B

---

276 | ID 4305
PATTERN 4.2

	0	1	2
0	-	B	C
1	B	-	-
2	-	C	B

---

277 | ID 4318
PATTERN 5.1

	0	1	2
0	A	-	-
1	A	-	-
2	-	C	A

---

278 | ID 4319
PATTERN 5.1

	0	1	2
0	A	-	-
1	C	-	-
2	-	A	C

---

279 | ID 4326
PATTERN 4.2

	0	1	2
0	-	B	C
1	B	-	-
2	-	B	C

---

280 | ID 4330
PATTERN 3.2

	0	1	2
0	A	A	B
1	B	-	-
2	-	B	A

---

281 | ID 4332
PATTERN 4.2

	0	1	2
0	-	B	C
1	C	-	-
2	-	C	B

---

282 | ID 4345
PATTERN 5.1

	0	1	2
0	A	-	-
1	C	-	-
2	-	C	A

---

283 | ID 4346
PATTERN 5.1

	0	1	2
0	A	-	-
1	A	-	-
2	-	A	C

Appendice B

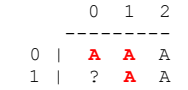
Dimostrazione per sottomatrici 2×3

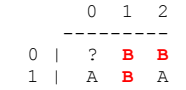
Si riporta di seguito l'output della variante di programma sviluppata per dimostrare che ogni sottomatrice vietata 2×3 contiene almeno una sottomatrice 2×2.

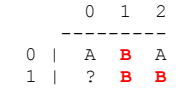
L'elenco contiene tutte le casistiche uniche di matrici vietate 2×3 generate dal programma, con relativa evidenziazione in rosso del pattern della sottomatrice vietata 2×2 contenuta.

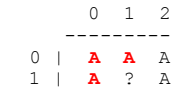
Il simbolo ? sta ad indicare bit ininfluenti nel raggiungimento dell'equivalenza di riga e di colonna della matrice rappresentata. Tali bit possono pertanto essere ignorati per lo scopo dimostrativo che ci si prefigge.

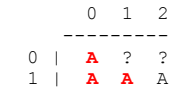
Da notare che la presente dimostrazione è valida anche per sottomatrici 3×2, è sufficiente invertire righe e colonne.

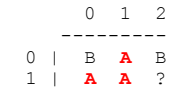

0 1 2
0 | A A A
1 | ? A A

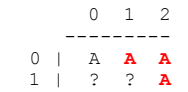

0 1 2
0 | ? B B
1 | A B A

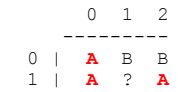

0 1 2
0 | A B A
1 | ? B B

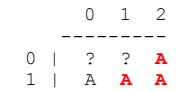

0 1 2
0 | A A A
1 | A ? A

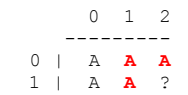

0 1 2
0 | A ? ?
1 | A A A

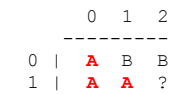

0 1 2
0 | B A B
1 | A A ?

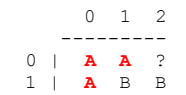

0 1 2
0 | A A A
1 | ? ? A

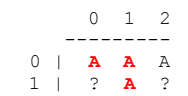

0 1 2
0 | A B B
1 | A ? A

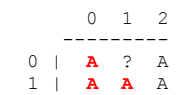

0 1 2
0 | ? ? A
1 | A A A

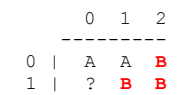

0 1 2
0 | A A A
1 | A A ?

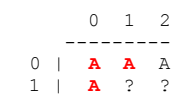

0 1 2
0 | A B B
1 | A A ?

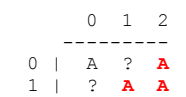

0 1 2
0 | A A ?
1 | A B B

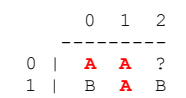

0 1 2
0 | A A A
1 | ? A ?

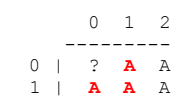

0 1 2
0 | A ? A
1 | A A A

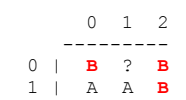

0 1 2
0 | A A B
1 | ? B B

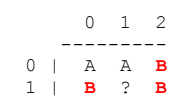

0 1 2
0 | A A A
1 | A ? ?

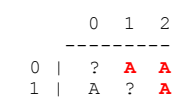

0 1 2
0 | A ? A
1 | ? A A

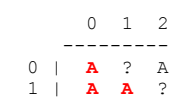

0 1 2
0 | A A ?
1 | B A B

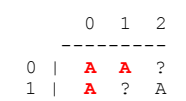

0 1 2
0 | ? A A
1 | A A A

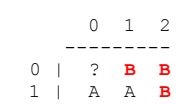

0 1 2
0 | B ? B
1 | A A B

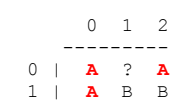

0 1 2
0 | A A B
1 | B ? B

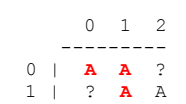

0 1 2
0 | ? A A
1 | A ? A

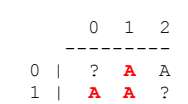

0 1 2
0 | A ? A
1 | A A ?

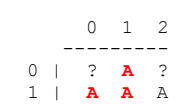

0 1 2
0 | A A ?
1 | A ? A

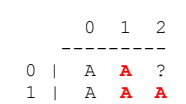

0 1 2
0 | ? B B
1 | A A B


0 1 2
0 | A ? A
1 | A B B


0 1 2
0 | A A ?
1 | ? A A


0 1 2
0 | ? A A
1 | A A ?


0 1 2
0 | ? A ?
1 | A A A


0 1 2
0 | A A ?
1 | A A A

Appendice C

Dimostrazione per sottomatrici 3×3

Si riporta di seguito l'output della variante di programma sviluppata per dimostrare che ogni sottomatrice vietata 3×3 contiene almeno una sottomatrice 2×2.

L'elenco contiene tutte le casistiche uniche di matrici vietate 3×3 generate dal programma, con relativa evidenziazione in rosso del pattern della sottomatrice vietata 2×2 contenuta.

Considerato l'elevato numero di combinazioni tramite cui è possibile ottenere l'equivalenza di riga e di colonna, si è provveduto a sviluppare una signature *ad hoc*. Una tale firma, esprimibile come un numero intero di 4 cifre, è suddivisa come segue:

- le prime due cifre (da 10 a 65) indicano la modalità con cui è stata raggiunta l'equivalenza di colonna. Questa prima parte della firma viene anche sfruttata per raggruppare e presentare in maniera sistematica e ordinata le matrici raccolte in questa appendice;
- la terza e quarta cifra (da 10 a 65), indicano la modalità con cui è stata raggiunta l'equivalenza di riga.

A parità di signature è sufficiente procedere all'analisi di una sola matrice, ignorando le altre contrassegnate con la medesima firma. In tal modo, partendo da un insieme in ingresso di 2025 matrici vietate in formato 3×3 (si vedano i risultati esposti nella sezione 4.1) è stato qui possibile arrivare a completare la dimostrazione presentando un elenco compatto di 379 matrici vietate, consultabile alle pagine seguenti.

SIGNATURE 10

SIGN 1010

	0	1	2
0	A	A	A
1	A	?	?
2	A	?	?

SIGN 1035

	0	1	2
0	A	?	?
1	A	B	B
2	A	A	?

SIGN 1011

	0	1	2
0	A	A	?
1	A	A	A
2	A	?	?

SIGN 1041

	0	1	2
0	A	B	B
1	A	?	A
2	A	?	?

SIGN 1012

	0	1	2
0	A	A	?
1	A	?	?
2	A	A	A

SIGN 1042

	0	1	2
0	A	B	B
1	A	?	?
2	A	?	A

SIGN 1014

	0	1	2
0	A	?	?
1	A	A	A
2	A	?	?

SIGN 1045

	0	1	2
0	A	?	?
1	A	B	B
2	A	?	A

SIGN 1015

	0	1	2
0	A	?	?
1	A	A	?
2	A	A	A

SIGN 1051

	0	1	2
0	A	?	A
1	A	A	?
2	A	?	?

SIGN 1018

	0	1	2
0	A	?	?
1	A	?	?
2	A	A	A

SIGN 1052

	0	1	2
0	A	?	A
1	A	?	?
2	A	A	?

SIGN 1021

	0	1	2
0	A	A	?
1	A	?	A
2	A	?	?

SIGN 1055

	0	1	2
0	A	?	?
1	A	?	A
2	A	A	?

SIGN 1022

	0	1	2
0	A	A	?
1	A	?	?
2	A	?	A

SIGN 1061

	0	1	2
0	A	?	A
1	A	B	B
2	A	?	?

SIGN 1025

	0	1	2
0	A	?	?
1	A	A	?
2	A	?	A

SIGN 1062

	0	1	2
0	B	?	B
1	B	?	?
2	B	A	A

SIGN 1031

	0	1	2
0	A	B	B
1	A	A	?
2	A	?	?

SIGN 1065

	0	1	2
0	B	?	?
1	B	?	B
2	B	A	A

SIGN 1032

	0	1	2
0	A	B	B
1	A	?	?
2	A	A	?

SIGNATURE 11

SIGN 1110

	0	1	2
0	B	B	B
1	B	A	?
2	?	A	?

SIGN 1135

	0	1	2
0	B	?	?
1	B	A	A
2	A	A	?

SIGN 1111

	0	1	2
0	B	B	?
1	B	A	A
2	?	A	?

SIGN 1141

	0	1	2
0	B	A	A
1	B	A	B
2	?	A	?

SIGN 1112

	0	1	2
0	C	C	?
1	C	A	?
2	?	A	A

SIGN 1142

	0	1	2
0	C	A	A
1	C	B	?
2	A	B	A

SIGN 1114

	0	1	2
0	A	?	?
1	A	A	A
2	?	A	?

SIGN 1145

	0	1	2
0	C	?	?
1	C	B	B
2	A	B	A

SIGN 1115

	0	1	2
0	A	?	?
1	A	A	?
2	?	A	A

SIGN 1151

	0	1	2
0	A	?	A
1	A	A	?
2	?	A	?

SIGN 1118

	0	1	2
0	C	?	?
1	C	A	?
2	A	A	A

SIGN 1152

	0	1	2
0	B	?	B
1	B	A	?
2	A	A	?

SIGN 1121

	0	1	2
0	B	B	?
1	B	A	B
2	?	A	?

SIGN 1155

	0	1	2
0	B	?	?
1	B	A	B
2	A	A	?

SIGN 1122

	0	1	2
0	B	B	?
1	B	B	?
2	A	B	A

SIGN 1161

	0	1	2
0	B	?	B
1	B	A	A
2	?	A	?

SIGN 1125

	0	1	2
0	B	?	?
1	B	B	?
2	A	B	A

SIGN 1162

	0	1	2
0	C	?	C
1	C	A	?
2	?	A	A

SIGN 1131

	0	1	2
0	A	B	B
1	A	A	?
2	?	A	?

SIGN 1165

	0	1	2
0	C	?	?
1	C	A	C
2	?	A	A

SIGN 1132

	0	1	2
0	B	A	A
1	B	A	?
2	A	A	?

SIGNATURE 12

SIGN 1210

	0	1	2
0	B	B	B
1	B	?	A
2	?	?	A

SIGN 1235

	0	1	2
0	C	?	?
1	C	A	A
2	B	B	A

SIGN 1211

	0	1	2
0	C	C	?
1	C	A	A
2	?	?	A

SIGN 1241

	0	1	2
0	A	B	B
1	A	?	A
2	?	?	A

SIGN 1212

	0	1	2
0	B	B	?
1	B	?	A
2	?	A	A

SIGN 1242

	0	1	2
0	B	A	A
1	B	?	A
2	A	?	A

SIGN 1214

	0	1	2
0	A	?	?
1	A	A	A
2	?	?	A

SIGN 1245

	0	1	2
0	B	?	?
1	B	A	A
2	A	?	A

SIGN 1215

	0	1	2
0	B	?	?
1	B	B	A
2	?	A	A

SIGN 1251

	0	1	2
0	B	?	B
1	B	B	A
2	?	?	A

SIGN 1218

	0	1	2
0	C	?	?
1	C	?	A
2	A	A	A

SIGN 1252

	0	1	2
0	C	?	C
1	C	?	A
2	A	A	A

SIGN 1221

	0	1	2
0	A	A	?
1	A	?	A
2	?	?	A

SIGN 1255

	0	1	2
0	A	?	?
1	A	?	A
2	B	B	A

SIGN 1222

	0	1	2
0	C	C	?
1	C	?	A
2	A	?	A

SIGN 1261

	0	1	2
0	B	?	B
1	B	A	A
2	?	?	A

SIGN 1225

	0	1	2
0	C	?	?
1	C	C	A
2	A	?	A

SIGN 1262

	0	1	2
0	A	?	A
1	A	?	A
2	?	A	A

SIGN 1231

	0	1	2
0	B	C	C
1	B	B	A
2	?	?	A

SIGN 1265

	0	1	2
0	A	?	?
1	A	?	A
2	?	A	A

SIGN 1232

	0	1	2
0	C	B	B
1	C	?	A
2	A	A	A

SIGNATURE 14

SIGN 1410

	0	1	2
0	**A**	**A**	A
1	?	**A**	?
2	?	A	?

SIGN 1411

	0	1	2
0	**A**	**A**	?
1	?	**A**	A
2	?	A	?

SIGN 1412

	0	1	2
0	**A**	**A**	?
1	?	**A**	?
2	?	A	A

SIGN 1414

	0	1	2
0	?	**A**	?
1	**A**	**A**	A
2	?	A	?

SIGN 1415

	0	1	2
0	?	**A**	?
1	**A**	**A**	?
2	?	A	A

SIGN 1418

	0	1	2
0	?	A	?
1	?	**A**	?
2	**A**	**A**	A

SIGN 1421

	0	1	2
0	**A**	**A**	?
1	C	**A**	C
2	?	A	?

SIGN 1422

	0	1	2
0	**B**	**B**	?
1	?	**B**	?
2	A	B	A

SIGN 1425

	0	1	2
0	?	**B**	?
1	**B**	**B**	?
2	A	B	A

SIGN 1431

	0	1	2
0	?	**A**	A
1	**A**	**A**	?
2	?	A	?

SIGN 1432

	0	1	2
0	?	A	A
1	?	**A**	?
2	**A**	**A**	?

SIGN 1435

	0	1	2
0	?	A	?
1	?	**A**	A
2	**A**	**A**	?

SIGN 1441

	0	1	2
0	?	**A**	**A**
1	B	**A**	B
2	?	A	?

SIGN 1451

	0	1	2
0	C	**A**	C
1	**A**	**A**	?
2	?	A	?

SIGN 1452

	0	1	2
0	C	A	C
1	?	**A**	?
2	**A**	**A**	?

SIGN 1455

	0	1	2
0	?	A	?
1	B	**A**	B
2	**A**	**A**	?

SIGN 1461

	0	1	2
0	C	**A**	C
1	?	**A**	**A**
2	?	A	?

SIGNATURE 15

SIGN 1510

	0	1	2
0	**B**	**B**	B
1	?	**B**	A
2	?	?	A

SIGN 1511

	0	1	2
0	**A**	**A**	?
1	?	**A**	A
2	?	?	A

SIGN 1512

	0	1	2
0	**B**	**B**	?
1	?	**B**	A
2	?	A	A

SIGN 1514

	0	1	2
0	?	**A**	?
1	**A**	**A**	A
2	?	?	A

SIGN 1515

	0	1	2
0	?	**B**	?
1	**B**	**B**	A
2	?	A	A

SIGN 1521

	0	1	2
0	**C**	**C**	?
1	A	**C**	A
2	?	?	A

SIGN 1522

	0	1	2
0	**C**	**C**	?
1	?	**C**	A
2	A	?	A

SIGN 1525

	0	1	2
0	?	**C**	?
1	**C**	**C**	A
2	A	?	A

SIGN 1531

	0	1	2
0	?	**B**	B
1	**B**	**B**	A
2	?	?	A

SIGN 1541

	0	1	2
0	?	**B**	**B**
1	A	**B**	A
2	?	?	A

SIGN 1542

	0	1	2
0	?	A	A
1	?	**A**	**A**
2	A	?	**A**

SIGN 1545

	0	1	2
0	?	**A**	?
1	?	**A**	**A**
2	A	?	A

SIGN 1551

	0	1	2
0	C	**B**	C
1	**B**	**B**	A
2	?	?	A

SIGN 1561

	0	1	2
0	C	**A**	C
1	?	**A**	**A**
2	?	?	A

SIGN 1562

	0	1	2
0	C	B	C
1	?	B	**A**
2	?	**A**	**A**

SIGN 1565

	0	1	2
0	?	B	?
1	A	B	**A**
2	?	**A**	**A**

SIGNATURE 18

SIGN 1810

	0	1	2
0	A	**A**	**A**
1	?	?	**A**
2	?	?	A

SIGN 1811

	0	1	2
0	C	C	**A**
1	?	**A**	**A**
2	?	?	A

SIGN 1812

	0	1	2
0	C	C	A
1	?	?	**A**
2	?	**A**	**A**

SIGN 1814

	0	1	2
0	?	?	**A**
1	A	**A**	**A**
2	?	?	A

SIGN 1818

	0	1	2
0	?	?	A
1	?	?	**A**
2	A	**A**	**A**

SIGN 1821

	0	1	2
0	C	C	**A**
1	**A**	?	**A**
2	?	?	A

SIGN 1831

	0	1	2
0	?	**A**	**A**
1	C	C	**A**
2	?	?	A

SIGN 1841

	0	1	2
0	?	**A**	**A**
1	A	?	**A**
2	?	?	A

SIGN 1842

	0	1	2
0	?	**A**	**A**
1	?	?	**A**
2	A	?	A

SIGN 1845

	0	1	2
0	?	?	**A**
1	?	**A**	**A**
2	A	?	A

SIGN 1851

	0	1	2
0	**A**	?	**A**
1	C	C	**A**
2	?	?	A

SIGN 1861

	0	1	2
0	A	?	**A**
1	?	**A**	**A**
2	?	?	A

SIGN 1862

	0	1	2
0	A	?	A
1	?	?	**A**
2	?	**A**	**A**

SIGN 1865

	0	1	2
0	?	?	A
1	A	?	**A**
2	?	**A**	**A**

SIGNATURE 21

SIGN 2110

	0	1	2
0	**A**	**A**	A
1	**A**	?	?
2	?	**A**	?

SIGN 2111

	0	1	2
0	**A**	**A**	?
1	**A**	B	B
2	?	A	?

SIGN 2112

	0	1	2
0	**A**	**A**	?
1	**A**	?	?
2	?	A	A

SIGN 2114

	0	1	2
0	**B**	A	?
1	**B**	**B**	B
2	?	A	?

SIGN 2115

	0	1	2
0	**B**	A	?
1	**B**	**B**	?
2	?	A	A

SIGN 2118

	0	1	2
0	C	**A**	?
1	C	?	?
2	**A**	**A**	A

SIGN 2121

	0	1	2
0	**A**	**A**	?
1	**A**	?	A
2	?	A	?

SIGN 2122

	0	1	2
0	**B**	**B**	?
1	**B**	?	?
2	A	B	A

SIGN 2125

	0	1	2
0	**C**	B	?
1	**C**	**C**	?
2	A	B	A

SIGN 2131

	0	1	2
0	**B**	A	A
1	**B**	**B**	?
2	?	A	?

SIGN 2132

	0	1	2
0	C	**A**	A
1	C	?	?
2	**A**	**A**	?

SIGN 2135

	0	1	2
0	C	**A**	?
1	C	B	B
2	**A**	**A**	?

SIGN 2141

	0	1	2
0	**B**	A	A
1	**B**	?	**B**
2	?	A	?

SIGN 2142

	0	1	2
0	C	**B**	**B**
1	C	?	?
2	A	**B**	A

SIGN 2145

	0	1	2
0	C	B	?
1	C	**A**	**A**
2	A	B	**A**

SIGN 2151

	0	1	2
0	**B**	A	B
1	**B**	**B**	?
2	?	A	?

SIGN 2152

	0	1	2
0	**C**	A	**C**
1	**C**	?	?
2	A	A	?

SIGN 2155

	0	1	2
0	**B**	A	?
1	**B**	?	**B**
2	A	A	?

SIGN 2161

	0	1	2
0	**C**	A	**C**
1	**C**	B	B
2	?	A	?

SIGN 2162

	0	1	2
0	**C**	A	**C**
1	**C**	?	?
2	?	A	A

SIGN 2165

	0	1	2
0	**C**	A	?
1	**C**	?	**C**
2	?	A	A

SIGNATURE 22

SIGN 2210

	0	1	2
0	**A**	**A**	**A**
1	**A**	?	?
2	?	?	**A**

SIGN 2211

	0	1	2
0	B	**B**	A
1	B	**B**	**B**
2	?	?	A

SIGN 2212

	0	1	2
0	**C**	**C**	A
1	**C**	?	?
2	?	A	A

SIGN 2214

	0	1	2
0	**B**	?	A
1	**B**	**B**	B
2	?	?	A

SIGN 2215

	0	1	2
0	**C**	?	A
1	**C**	**C**	?
2	?	A	A

SIGN 2221

	0	1	2
0	**B**	**B**	A
1	**B**	?	B
2	?	?	A

SIGN 2222

	0	1	2
0	**C**	**C**	A
1	**C**	?	?
2	A	?	A

SIGN 2231

	0	1	2
0	**C**	A	A
1	**C**	**C**	?
2	?	?	A

SIGN 2232

	0	1	2
0	C	**A**	**A**
1	C	?	?
2	B	B	**A**

SIGN 2235

	0	1	2
0	C	?	**A**
1	C	B	B
2	A	**A**	**A**

SIGN 2241

	0	1	2
0	**B**	A	A
1	**B**	?	**B**
2	?	?	A

SIGN 2242

	0	1	2
0	C	**A**	**A**
1	C	?	?
2	A	?	**A**

SIGN 2245

	0	1	2
0	B	?	**A**
1	B	C	C
2	**A**	?	**A**

SIGN 2251

	0	1	2
0	**A**	?	A
1	**A**	**A**	?
2	?	?	A

SIGN 2252

	0	1	2
0	**A**	?	**A**
1	**A**	?	?
2	B	B	A

SIGN 2255

	0	1	2
0	**B**	?	A
1	**B**	?	**B**
2	A	A	A

SIGN 2261

	0	1	2
0	**A**	?	**A**
1	**A**	B	B
2	?	?	A

SIGN 2262

	0	1	2
0	**A**	?	**A**
1	**A**	?	?
2	?	A	A

SIGN 2265

	0	1	2
0	**C**	?	A
1	**C**	?	**C**
2	?	A	A

SIGNATURE 25

SIGN 2510

	0	1	2
0	**A**	**A**	**A**
1	?	**A**	?
2	?	?	**A**

SIGN 2511

	0	1	2
0	**B**	**B**	A
1	?	**B**	B
2	?	?	A

SIGN 2512

	0	1	2
0	**C**	**C**	A
1	?	**C**	?
2	?	A	A

SIGN 2514

	0	1	2
0	?	**B**	A
1	**B**	**B**	B
2	?	?	A

SIGN 2515

	0	1	2
0	?	**C**	A
1	**C**	**C**	?
2	?	A	A

SIGN 2521

	0	1	2
0	**C**	**C**	A
1	B	**C**	B
2	?	?	A

SIGN 2525

	0	1	2
0	?	**C**	A
1	**C**	**C**	?
2	A	?	A

SIGN 2531

	0	1	2
0	?	**A**	A
1	**A**	**A**	?
2	?	?	A

SIGN 2541

	0	1	2
0	?	**A**	**A**
1	B	**A**	B
2	?	?	A

SIGN 2542

	0	1	2
0	?	**A**	**A**
1	?	**A**	?
2	A	?	A

SIGN 2551

	0	1	2
0	A	**C**	A
1	**C**	**C**	?
2	?	?	A

SIGN 2561

	0	1	2
0	A	**B**	A
1	?	**B**	**B**
2	?	?	A

SIGN 2562

	0	1	2
0	A	B	**A**
1	?	B	?
2	?	**A**	**A**

SIGNATURE 31

SIGN 3110

	0	1	2
0	**B**	**B**	B
1	A	**B**	?
2	A	?	?

SIGN 3111

	0	1	2
0	**B**	**B**	?
1	A	**B**	B
2	A	?	?

SIGN 3112

	0	1	2
0	**C**	**C**	?
1	A	**C**	?
2	A	A	A

SIGN 3114

	0	1	2
0	?	**A**	?
1	**A**	**A**	A
2	A	?	?

SIGN 3115

	0	1	2
0	?	**A**	?
1	**A**	**A**	?
2	A	A	A

SIGN 3118

	0	1	2
0	?	C	?
1	**A**	C	?
2	**A**	**A**	A

SIGN 3121

	0	1	2
0	**B**	**B**	?
1	A	**B**	A
2	A	?	?

SIGN 3122

	0	1	2
0	**C**	**C**	?
1	A	**C**	?
2	A	?	A

SIGN 3125

	0	1	2
0	?	**A**	?
1	**A**	**A**	?
2	A	?	A

SIGN 3131

	0	1	2
0	?	**A**	A
1	**A**	**A**	?
2	A	?	?

SIGN 3132

	0	1	2
0	?	B	B
1	**A**	B	?
2	**A**	**A**	?

SIGN 3135

	0	1	2
0	?	B	?
1	**A**	B	B
2	**A**	**A**	?

SIGN 3141

	0	1	2
0	?	**B**	**B**
1	A	**B**	A
2	A	?	?

SIGN 3142

	0	1	2
0	?	**B**	**B**
1	A	**B**	?
2	A	?	A

SIGN 3145

	0	1	2
0	?	**B**	?
1	A	**B**	**B**
2	A	?	A

SIGN 3151

	0	1	2
0	B	**A**	B
1	**A**	**A**	?
2	A	?	?

SIGN 3152

	0	1	2
0	C	B	C
1	**A**	B	?
2	**A**	**A**	?

SIGN 3155

	0	1	2
0	?	B	?
1	**A**	B	A
2	**A**	**A**	?

SIGN 3161

	0	1	2
0	C	**B**	C
1	A	**B**	**B**
2	A	?	?

SIGN 3162

	0	1	2
0	A	C	**A**
1	B	C	?
2	B	**A**	**A**

SIGN 3165

	0	1	2
0	?	A	?
1	B	**A**	B
2	B	**A**	**A**

SIGNATURE 32

SIGN 3210

	0	1	2
0	B	**B**	**B**
1	A	?	**B**
2	A	?	?

SIGN 3211

	0	1	2
0	B	B	A
1	**A**	**A**	A
2	**A**	?	?

SIGN 3212

	0	1	2
0	C	C	B
1	**A**	?	B
2	**A**	**A**	A

SIGN 3214

	0	1	2
0	?	?	A
1	**A**	**A**	A
2	**A**	?	?

SIGN 3221

	0	1	2
0	C	C	A
1	**A**	?	**A**
2	**A**	?	?

SIGN 3222

	0	1	2
0	C	C	B
1	**A**	?	B
2	**A**	?	**A**

SIGN 3231

	0	1	2
0	?	B	B
1	**A**	**A**	B
2	**A**	?	?

SIGN 3232

	0	1	2
0	?	B	B
1	**A**	?	B
2	**A**	**A**	?

SIGN 3241

	0	1	2
0	?	**B**	**B**
1	B	?	**B**
2	B	?	?

SIGN 3242

	0	1	2
0	?	**C**	**C**
1	A	?	**C**
2	A	?	A

SIGN 3245

	0	1	2
0	?	?	**B**
1	A	**B**	**B**
2	A	?	A

SIGN 3251

	0	1	2
0	B	?	B
1	**A**	**A**	B
2	**A**	?	?

SIGN 3255

	0	1	2
0	?	?	A
1	**A**	?	A
2	**A**	**A**	?

SIGN 3261

	0	1	2
0	B	?	**B**
1	A	**B**	**B**
2	A	?	?

SIGN 3262

	0	1	2
0	A	?	A
1	B	?	**A**
2	B	**A**	**A**

SIGN 3265

	0	1	2
0	?	?	B
1	**B**	?	**B**
2	**B**	A	A

SIGNATURE 35

SIGN 3510

	0	1	2
0	B	**B**	**B**
1	?	A	**B**
2	?	A	?

SIGN 3511

	0	1	2
0	C	C	A
1	?	**A**	**A**
2	?	**A**	?

SIGN 3512

	0	1	2
0	C	C	B
1	?	**A**	B
2	?	**A**	**A**

SIGN 3514

	0	1	2
0	?	?	A
1	**A**	**A**	A
2	?	**A**	?

SIGN 3521

	0	1	2
0	C	C	**B**
1	**B**	A	**B**
2	?	A	?

SIGN 3522

	0	1	2
0	C	C	A
1	?	B	**A**
2	**A**	B	**A**

SIGN 3531

	0	1	2
0	?	B	B
1	**A**	**A**	B
2	?	**A**	?

SIGN 3535

	0	1	2
0	?	?	A
1	?	**A**	A
2	**A**	**A**	?

SIGN 3541

	0	1	2
0	?	**B**	**B**
1	B	A	**B**
2	?	A	?

SIGN 3542

	0	1	2
0	?	**A**	**A**
1	?	B	**A**
2	A	B	A

SIGN 3551

	0	1	2
0	C	?	C
1	**A**	**A**	C
2	?	**A**	?

SIGN 3552

	0	1	2
0	B	?	B
1	?	**A**	B
2	**A**	**A**	?

SIGN 3561

	0	1	2
0	A	?	A
1	?	**A**	**A**
2	?	**A**	?

SIGN 3565

	0	1	2
0	?	?	C
1	C	**A**	C
2	?	**A**	**A**

SIGNATURE 41

SIGN 4110

	0	1	2
0	**A**	**A**	A
1	B	?	?
2	B	**A**	?

SIGN 4111

	0	1	2
0	A	A	?
1	**B**	**B**	B
2	**B**	A	?

SIGN 4112

	0	1	2
0	**A**	**A**	?
1	B	?	?
2	B	**A**	A

SIGN 4114

	0	1	2
0	?	A	?
1	**B**	**B**	B
2	**B**	A	?

SIGN 4115

	0	1	2
0	?	A	?
1	**B**	**B**	?
2	**B**	A	A

SIGN 4118

	0	1	2
0	?	A	?
1	**A**	?	?
2	**A**	**A**	A

SIGN 4121

	0	1	2
0	**A**	**A**	?
1	B	?	B
2	B	**A**	?

SIGN 4122

	0	1	2
0	**B**	**B**	?
1	A	?	?
2	A	**B**	A

SIGN 4125

	0	1	2
0	?	B	?
1	**A**	**A**	?
2	**A**	B	A

SIGN 4131

	0	1	2
0	?	A	A
1	**B**	**B**	?
2	**B**	A	?

SIGN 4132

	0	1	2
0	?	A	A
1	**A**	?	?
2	**A**	**A**	?

SIGN 4135

	0	1	2
0	?	A	?
1	**A**	B	B
2	**A**	**A**	?

SIGN 4141

	0	1	2
0	?	A	A
1	**A**	?	A
2	**A**	**A**	?

SIGN 4142

	0	1	2
0	?	B	B
1	**A**	?	?
2	**A**	B	**A**

SIGN 4145

	0	1	2
0	?	B	?
1	**A**	C	C
2	**A**	B	**A**

SIGN 4151

	0	1	2
0	C	A	C
1	**B**	**B**	?
2	**B**	A	?

SIGN 4152

	0	1	2
0	B	A	B
1	**A**	?	?
2	**A**	**A**	?

SIGN 4155

	0	1	2
0	?	A	?
1	**A**	?	A
2	**A**	**A**	?

SIGN 4161

	0	1	2
0	B	A	B
1	**A**	B	B
2	**A**	**A**	?

SIGN 4162

	0	1	2
0	C	**A**	C
1	B	?	?
2	B	**A**	**A**

SIGN 4165

	0	1	2
0	?	A	?
1	**B**	?	**B**
2	**B**	A	A

SIGNATURE 42

SIGN 4210

	0	1	2
0	A	**A**	**A**
1	B	?	?
2	B	?	**A**

SIGN 4211

	0	1	2
0	C	C	A
1	**A**	B	B
2	**A**	?	**A**

SIGN 4212

	0	1	2
0	B	B	A
1	**A**	?	?
2	**A**	**A**	A

SIGN 4215

	0	1	2
0	?	?	A
1	**A**	A	?
2	**A**	A	**A**

SIGN 4218

	0	1	2
0	?	?	A
1	**A**	?	?
2	**A**	**A**	A

SIGN 4221

	0	1	2
0	C	C	A
1	**B**	?	**B**
2	**B**	?	A

SIGN 4222

	0	1	2
0	C	C	A
1	**A**	?	?
2	**A**	?	**A**

SIGN 4225

	0	1	2
0	?	?	A
1	**A**	**A**	?
2	**A**	?	A

SIGN 4231

	0	1	2
0	?	A	A
1	**B**	**B**	?
2	**B**	?	A

SIGN 4232

	0	1	2
0	?	A	A
1	**B**	?	?
2	**B**	**B**	A

SIGN 4235

	0	1	2
0	?	?	A
1	**A**	B	B
2	**A**	**A**	A

SIGN 4241

	0	1	2
0	?	A	A
1	**B**	?	**B**
2	**B**	?	A

SIGN 4242

	0	1	2
0	?	A	A
1	**A**	?	?
2	**A**	?	**A**

SIGN 4245

	0	1	2
0	?	?	A
1	**A**	B	B
2	**A**	?	**A**

SIGN 4251

	0	1	2
0	A	?	A
1	**B**	**B**	?
2	**B**	?	A

SIGN 4252

	0	1	2
0	A	?	A
1	**B**	?	?
2	**B**	**B**	A

SIGN 4261

	0	1	2
0	**A**	?	**A**
1	B	C	C
2	B	?	**A**

SIGN 4262

	0	1	2
0	A	?	**A**
1	B	?	?
2	B	**A**	**A**

SIGN 4265

	0	1	2
0	?	?	A
1	**B**	?	**B**
2	**B**	A	A

SIGNATURE 45

SIGN 4510

	0	1	2
0	A	**A**	**A**
1	?	B	?
2	?	B	**A**

SIGN 4511

	0	1	2
0	C	C	A
1	?	**B**	**B**
2	?	**B**	A

SIGN 4512

	0	1	2
0	B	B	A
1	?	**A**	?
2	?	**A**	**A**

SIGN 4515

	0	1	2
0	?	?	A
1	**A**	**A**	?
2	?	**A**	A

SIGN 4518

	0	1	2
0	?	?	A
1	?	**A**	?
2	**A**	**A**	A

SIGN 4521

	0	1	2
0	C	C	A
1	B	**A**	B
2	?	**A**	**A**

SIGN 4522

	0	1	2
0	C	C	**A**
1	?	B	?
2	**A**	B	**A**

SIGN 4531

	0	1	2
0	?	A	A
1	**B**	**B**	?
2	?	**B**	A

SIGN 4532

	0	1	2
0	?	A	A
1	?	**B**	?
2	**B**	**B**	A

SIGN 4541

	0	1	2
0	?	**A**	**A**
1	C	B	C
2	?	B	**A**

SIGN 4542

	0	1	2
0	?	**A**	**A**
1	?	B	?
2	A	B	**A**

SIGN 4545

	0	1	2
0	?	?	A
1	?	**B**	**B**
2	A	**B**	A

SIGN 4551

	0	1	2
0	A	?	A
1	**B**	**B**	?
2	?	**B**	A

SIGN 4552

	0	1	2
0	A	?	A
1	?	**B**	?
2	**B**	**B**	A

SIGN 4555

	0	1	2
0	?	?	A
1	B	**A**	B
2	A	**A**	**A**

SIGN 4561

	0	1	2
0	A	?	A
1	?	**B**	**B**
2	?	**B**	A

SIGN 4562

	0	1	2
0	A	?	A
1	?	**A**	?
2	?	**A**	**A**

SIGNATURE 51

SIGN 5110

	0	1	2
0	A	A	A
1	?	A	?
2	A	?	?

SIGN 5111

	0	1	2
0	A	A	?
1	?	A	A
2	A	?	?

SIGN 5112

	0	1	2
0	A	A	?
1	?	A	?
2	A	A	A

SIGN 5114

	0	1	2
0	A	B	?
1	B	B	B
2	A	?	?

SIGN 5115

	0	1	2
0	A	B	?
1	B	B	?
2	A	A	A

SIGN 5118

	0	1	2
0	A	B	?
1	?	B	?
2	A	A	A

SIGN 5121

	0	1	2
0	A	A	?
1	B	A	B
2	A	?	?

SIGN 5122

	0	1	2
0	A	A	?
1	?	A	?
2	A	?	A

SIGN 5125

	0	1	2
0	A	B	?
1	B	B	?
2	A	?	A

SIGN 5131

	0	1	2
0	A	B	B
1	B	B	?
2	A	?	?

SIGN 5132

	0	1	2
0	A	B	B
1	?	B	?
2	A	A	?

SIGN 5135

	0	1	2
0	A	B	?
1	?	B	B
2	A	A	?

SIGN 5141

	0	1	2
0	A	C	C
1	B	C	B
2	A	?	?

SIGN 5142

	0	1	2
0	A	B	B
1	?	B	?
2	A	?	A

SIGN 5145

	0	1	2
0	A	B	?
1	?	B	B
2	A	?	A

SIGN 5151

	0	1	2
0	A	B	A
1	B	B	?
2	A	?	?

SIGN 5152

	0	1	2
0	A	B	A
1	?	B	?
2	A	A	?

SIGN 5155

	0	1	2
0	A	C	?
1	B	C	B
2	A	A	?

SIGN 5161

	0	1	2
0	A	B	A
1	?	B	B
2	A	?	?

SIGN 5162

	0	1	2
0	B	C	B
1	?	C	?
2	B	A	A

SIGN 5165

	0	1	2
0	B	C	?
1	A	C	A
2	B	A	A

SIGNATURE 52

SIGN 5210

	0	1	2
0	A	A	A
1	?	?	A
2	A	?	?

SIGN 5211

	0	1	2
0	A	A	B
1	?	B	B
2	A	?	?

SIGN 5212

	0	1	2
0	A	A	B
1	?	?	B
2	A	A	A

SIGN 5214

	0	1	2
0	A	?	B
1	B	B	B
2	A	?	?

SIGN 5221

	0	1	2
0	A	A	B
1	B	?	B
2	A	?	?

SIGN 5222

	0	1	2
0	A	A	B
1	?	?	B
2	A	?	A

SIGN 5231

	0	1	2
0	A	B	B
1	C	C	B
2	A	?	?

SIGN 5235

	0	1	2
0	A	?	B
1	?	B	B
2	A	A	?

SIGN 5241

	0	1	2
0	A	B	B
1	B	?	B
2	A	?	?

SIGN 5242

	0	1	2
0	A	B	B
1	?	?	B
2	A	?	A

SIGN 5245

	0	1	2
0	A	?	C
1	?	C	C
2	A	?	A

SIGN 5251

	0	1	2
0	A	?	A
1	B	B	A
2	A	?	?

SIGN 5252

	0	1	2
0	A	?	A
1	?	?	A
2	A	A	?

SIGN 5261

	0	1	2
0	B	?	B
1	?	B	B
2	B	?	?

SIGN 5262

	0	1	2
0	B	?	B
1	?	?	B
2	B	A	A

SIGN 5265

	0	1	2
0	B	?	A
1	A	?	A
2	B	A	A

SIGNATURE 55

SIGN 5510

	0	1	2
0	A	A	A
1	?	?	A
2	?	A	?

SIGN 5511

	0	1	2
0	A	A	B
1	?	B	B
2	?	A	?

SIGN 5512

	0	1	2
0	A	A	B
1	?	?	B
2	?	A	A

SIGN 5514

	0	1	2
0	?	A	B
1	B	B	B
2	?	A	?

SIGN 5521

	0	1	2
0	A	A	C
1	C	?	C
2	?	A	?

SIGN 5522

	0	1	2
0	B	B	A
1	?	?	A
2	A	B	A

SIGN 5531

	0	1	2
0	?	A	A
1	B	B	A
2	?	A	?

SIGN 5532

	0	1	2
0	?	A	A
1	?	?	A
2	A	A	?

SIGN 5541

	0	1	2
0	?	A	A
1	A	?	A
2	?	A	?

SIGN 5545

	0	1	2
0	?	B	A
1	?	A	A
2	A	B	A

SIGN 5551

	0	1	2
0	B	A	B
1	C	C	B
2	?	A	?

SIGN 5555

	0	1	2
0	?	A	B
1	B	?	B
2	A	A	?

SIGN 5561

	0	1	2
0	B	A	B
1	?	B	B
2	?	A	?

SIGN 5562

	0	1	2
0	C	A	C
1	?	?	C
2	?	A	A

SIGNATURE 61

SIGN 6110

	0	1	2
0	B	B	B
1	?	A	?
2	B	A	?

SIGN 6111

	0	1	2
0	B	B	?
1	?	A	A
2	B	A	?

SIGN 6112

	0	1	2
0	B	B	?
1	?	A	?
2	B	A	A

SIGN 6114

	0	1	2
0	B	?	?
1	A	A	A
2	B	A	?

SIGN 6115

	0	1	2
0	B	?	?
1	A	A	?
2	B	A	A

SIGN 6118

	0	1	2
0	A	?	?
1	?	A	?
2	A	A	A

SIGN 6121

	0	1	2
0	B	B	?
1	C	A	C
2	B	A	?

SIGN 6122

	0	1	2
0	A	A	?
1	?	B	?
2	A	B	A

SIGN 6125

	0	1	2
0	A	?	?
1	B	B	?
2	A	B	A

SIGN 6131

	0	1	2
0	B	C	C
1	A	A	?
2	B	A	?

SIGN 6132

	0	1	2
0	A	B	B
1	?	A	?
2	A	A	?

SIGN 6135

	0	1	2
0	A	?	?
1	?	A	A
2	A	A	?

SIGN 6141

	0	1	2
0	A	B	B
1	B	A	B
2	A	A	?

SIGN 6142

	0	1	2
0	A	C	C
1	?	B	?
2	A	B	A

SIGN 6145

	0	1	2
0	A	?	?
1	?	B	B
2	A	B	A

SIGN 6151

	0	1	2
0	B	?	B
1	A	A	?
2	B	A	?

SIGN 6152

	0	1	2
0	A	?	A
1	?	A	?
2	A	A	?

SIGN 6155

	0	1	2
0	A	?	?
1	B	A	B
2	A	A	?

SIGN 6161

	0	1	2
0	A	?	A
1	?	A	A
2	A	A	?

SIGN 6162

	0	1	2
0	B	?	B
1	?	A	?
2	B	A	A

SIGN 6165

	0	1	2
0	B	?	?
1	C	A	C
2	B	A	A

SIGNATURE 62

SIGN 6210

	0	1	2
0	B	B	B
1	?	?	A
2	B	?	A

SIGN 6211

	0	1	2
0	B	B	?
1	?	A	A
2	B	?	A

SIGN 6212

	0	1	2
0	A	A	?
1	?	?	A
2	A	A	A

SIGN 6215

	0	1	2
0	A	?	?
1	B	B	A
2	A	A	A

SIGN 6218

	0	1	2
0	A	?	?
1	?	?	A
2	A	A	A

SIGN 6221

	0	1	2
0	B	B	?
1	A	?	A
2	B	?	A

SIGN 6222

	0	1	2
0	A	A	?
1	?	?	A
2	A	?	A

SIGN 6225

	0	1	2
0	A	?	?
1	C	C	A
2	A	?	A

SIGN 6231

	0	1	2
0	A	C	C
1	B	B	A
2	A	?	A

SIGN 6232

	0	1	2
0	A	C	C
1	?	?	A
2	A	A	A

SIGN 6241

	0	1	2
0	B	C	C
1	A	?	A
2	B	?	A

SIGN 6242

	0	1	2
0	A	B	B
1	?	?	A
2	A	?	A

SIGN 6245

	0	1	2
0	A	?	?
1	?	A	A
2	A	?	A

SIGN 6251

	0	1	2
0	B	?	B
1	C	C	A
2	B	?	A

SIGN 6252

	0	1	2
0	B	?	B
1	?	?	A
2	B	B	A

SIGN 6255

	0	1	2
0	B	?	?
1	A	?	A
2	B	B	A

SIGN 6261

	0	1	2
0	B	?	B
1	?	A	A
2	B	?	A

SIGN 6262

	0	1	2
0	B	?	B
1	?	?	A
2	B	A	A

SIGN 6265

	0	1	2
0	B	?	?
1	A	?	A
2	B	A	A

SIGNATURE 65

SIGN 6510

	0	1	2
0	B	B	B
1	?	?	A
2	?	B	A

SIGN 6511

	0	1	2
0	B	B	?
1	?	A	A
2	?	B	A

SIGN 6512

	0	1	2
0	A	A	?
1	?	?	A
2	?	A	A

SIGN 6515

	0	1	2
0	?	A	?
1	B	B	A
2	?	A	A

SIGN 6518

	0	1	2
0	?	A	?
1	?	?	A
2	A	A	A

SIGN 6521

	0	1	2
0	B	B	?
1	A	?	A
2	?	B	A

SIGN 6522

	0	1	2
0	B	B	?
1	?	?	A
2	A	B	A

SIGN 6531

	0	1	2
0	?	B	B
1	A	A	A
2	?	B	A

SIGN 6532

	0	1	2
0	?	B	B
1	?	?	A
2	B	B	A

SIGN 6535

	0	1	2
0	?	B	?
1	?	A	A
2	B	B	A

SIGN 6541

	0	1	2
0	?	B	B
1	A	?	A
2	?	B	A

SIGN 6542

	0	1	2
0	?	B	B
1	?	?	A
2	A	B	A

SIGN 6551

	0	1	2
0	C	A	C
1	B	B	A
2	?	A	A

SIGN 6552

	0	1	2
0	B	A	B
1	?	?	A
2	A	A	A

SIGN 6561

	0	1	2
0	C	B	C
1	?	A	A
2	?	B	A

SIGN 6562

	0	1	2
0	C	A	C
1	?	?	A
2	?	A	A

SIGN 6565

	0	1	2
0	?	A	?
1	A	?	A
2	?	A	A

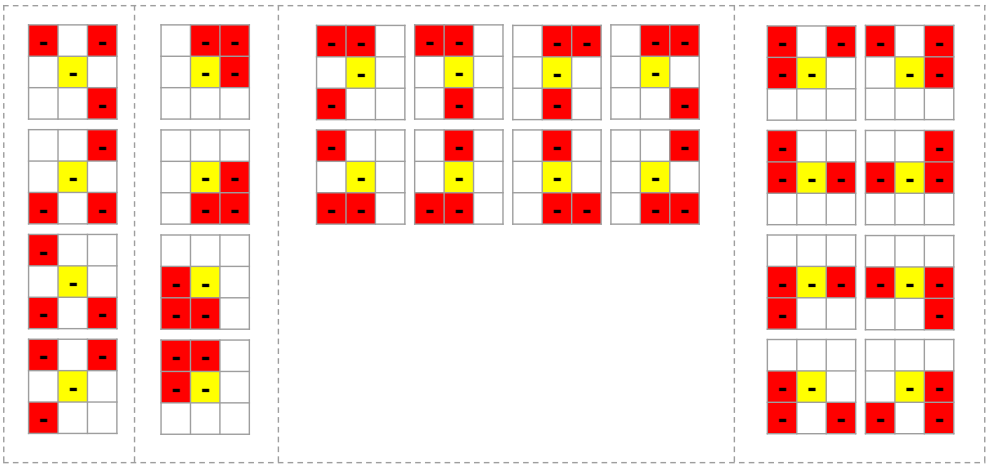
Appendice D

Combinazioni di pattern necessari e pattern negativi

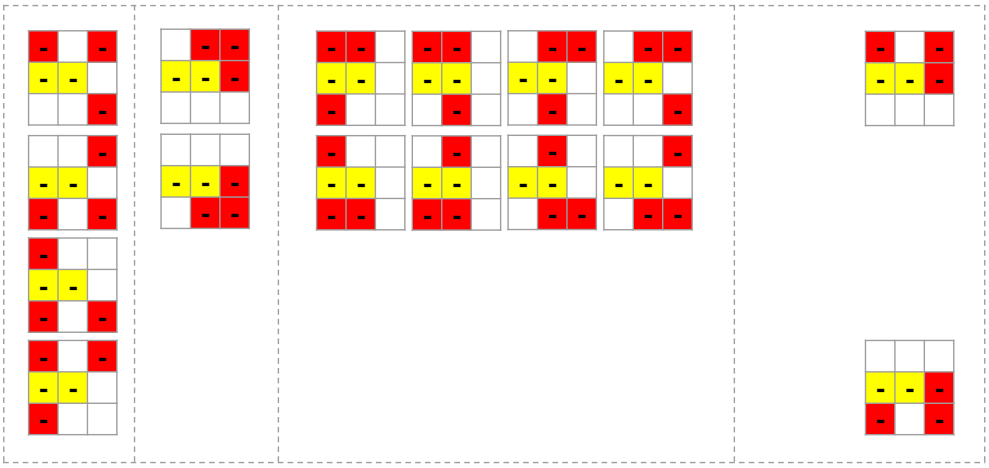
Per ciascuno dei 9 pattern necessari individuati alla sezione 4.2.2 viene di seguito riportato un prospetto che raccoglie le possibili combinazioni tra il pattern necessario e i pattern negativi sovrapponibili.

Una matrice appartenente ad un pattern necessario sarà vietata se riconducibile ad uno degli schemi indicati nei prospetti che seguono, ammessa altrimenti.

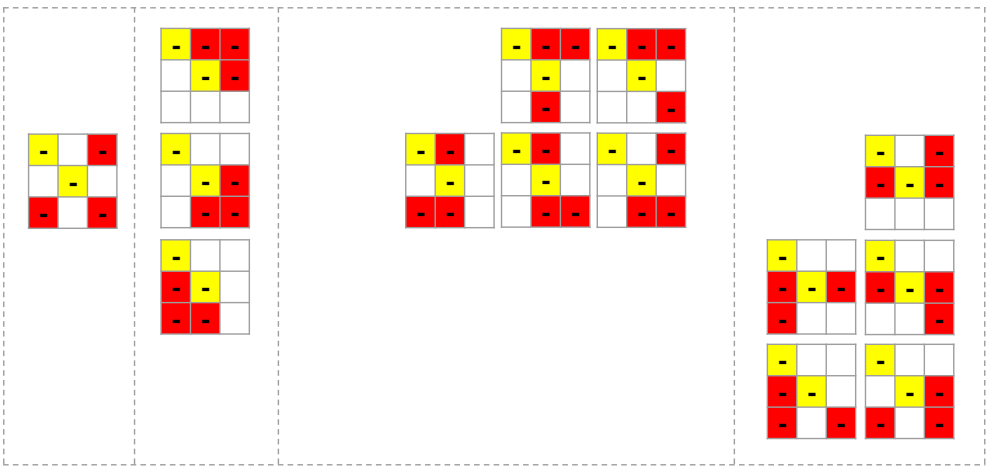
1.1



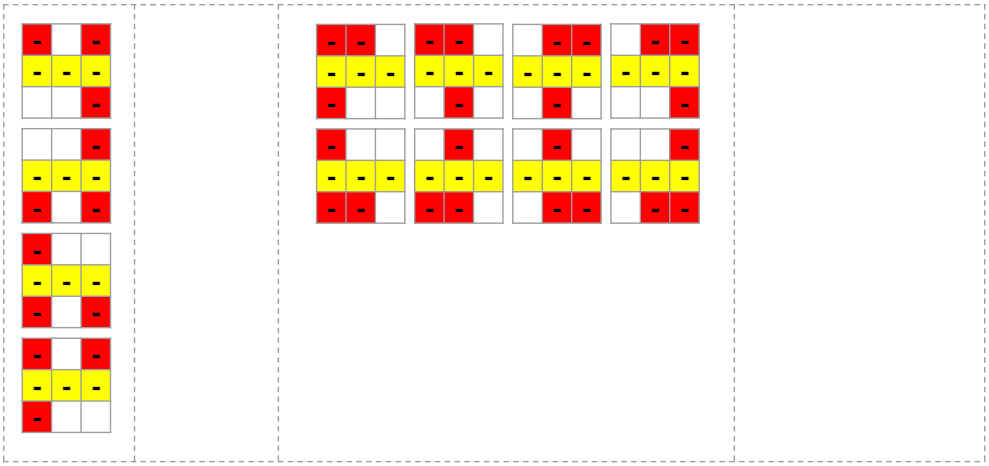
2.1



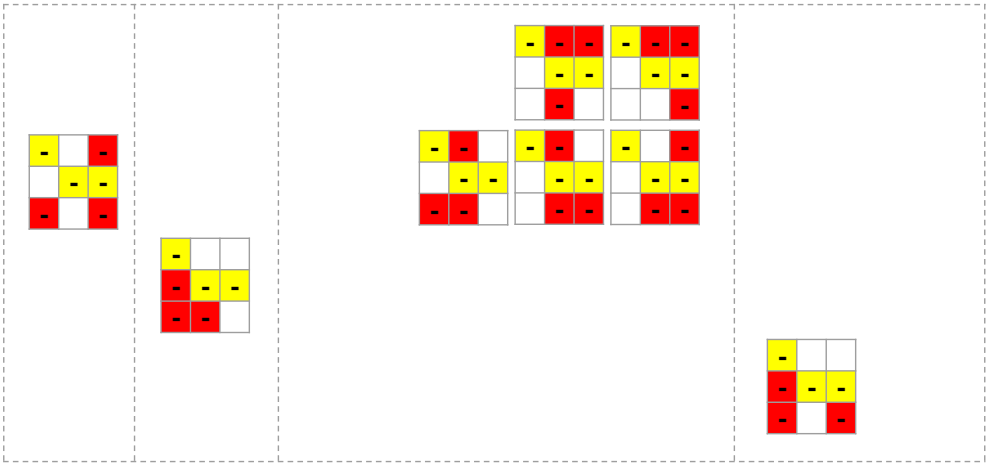
2.2



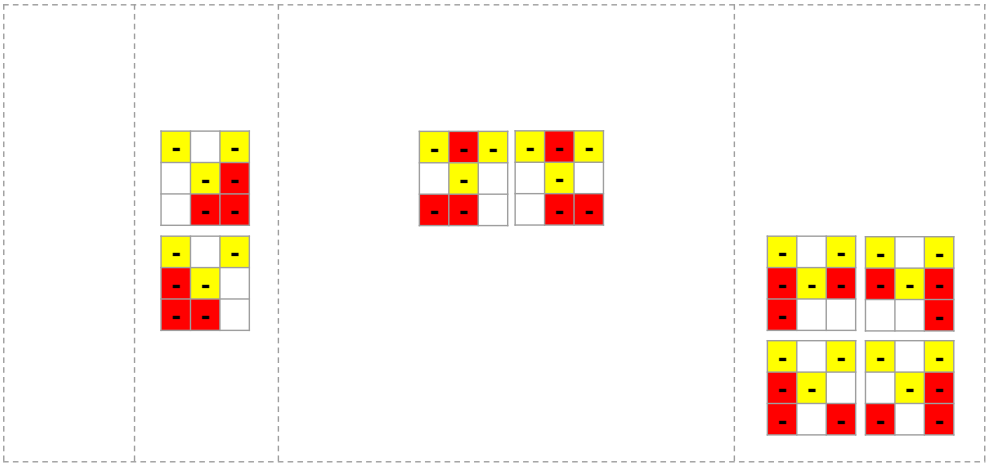
3.1



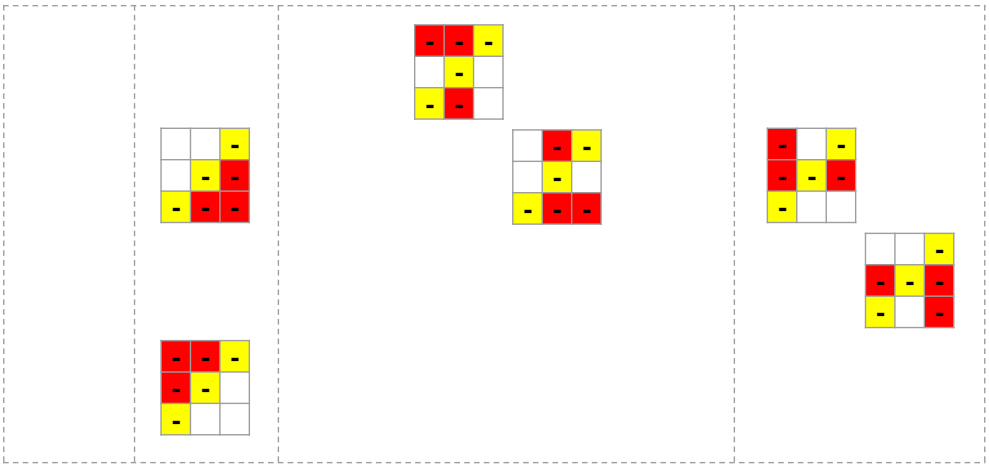
3.2



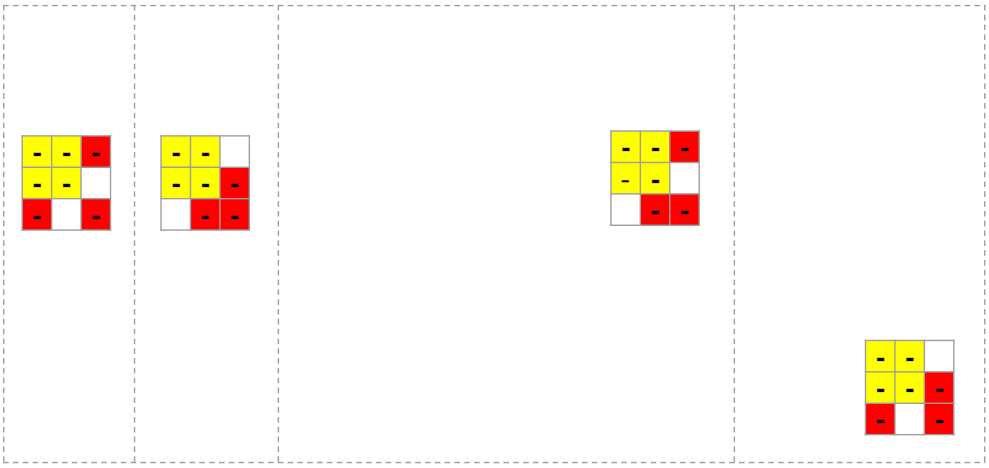
3.3



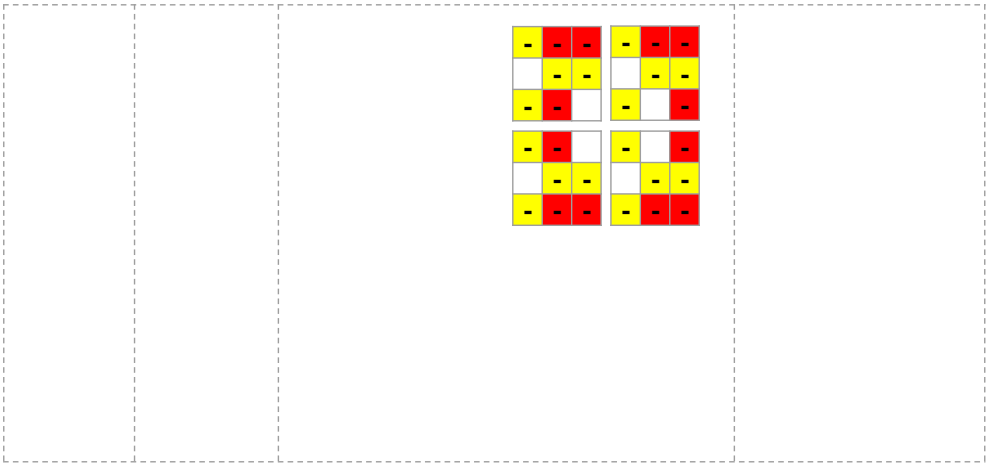
3.4



4.1



4.2



Appendice E

Dettaglio dei pattern individuati

Si riepilogano di seguito i 12 pattern sotto i quali ricadono tutte le matrici rappresentanti funzioni privatamente computabili (il cui elenco completo è consultabile all'appendice A).

Ogni pattern viene qui suddiviso in varianti, individuabili contando le occorrenze dei bit alternativi al bit centrale. Tali varianti agevolano la classificazione delle funzioni in logica ternaria prese in esame, come mostrato nella sezione 4.3.

Si riportano, per ciascuna variante di ciascun pattern, gli ID delle matrici ammesse che appartengono a quella variante.

PATTERN 1.1 (x7)

VARIANTE A1 (x2)
ID 433 | ID 2353

	0	1	2
0	A	B	A
1	A	-	A
2	A	B	A

VARIANTE B1 (x4)
ID 1136 | ID 2360
ID 2630 | ID 3326

	0	1	2
0	A	B	A
1	B	-	B
2	B	A	B

VARIANTE C1 (x1)
ID 2623

	0	1	2
0	A	B	A
1	B	-	B
2	A	B	A

FINE PATTERN 1.1

PATTERN 2.1 (x28)

VARIANTE A1 (x4)
ID 379 | ID 460
ID 2275 | ID 2356

	0	1	2
0	A	A	A
1	-	-	C
2	A	A	A

VARIANTE B1 (x4)
ID 1379 | ID 2289
ID 2636 | ID 3380

	0	1	2
0	A	A	B
1	-	-	A
2	A	A	B

VARIANTE C1 (x4)
ID 1190 | ID 2366
ID 2829 | ID 3569

	0	1	2
0	A	B	B
1	-	-	A
2	A	B	B

VARIANTE C2 (x4)
ID 2626 | ID 2650
ID 2815 | ID 2866

	0	1	2
0	A	B	A
1	-	-	B
2	A	B	A

VARIANTE C3 (x4)
ID 449 | ID 1406
ID 2295 | ID 3137

	0	1	2
0	A	A	B
1	-	-	B
2	A	A	B

VARIANTE C4 (x4)
ID 1390 | ID 3142
ID 3337 | ID 3805

	0	1	2
0	A	A	B
1	-	-	A
2	B	B	A

VARIANTE C5 (x4)
ID 1150 | ID 1417
ID 3385 | ID 3799

	0	1	2
0	A	A	B
1	-	-	B
2	B	B	A

FINE PATTERN 2.1

PATTERN 2.2 (x12)

VARIANTE A1 (x4)
ID 1137 | ID 2369
ID 3236 | ID 4088

	0	1	2
0	-	B	B
1	B	-	C
2	C	B	B

VARIANTE A2 (x4)
ID 1299 | ID 2361
ID 3344 | ID 3932

	0	1	2
0	-	B	C
1	B	-	B
2	B	C	B

VARIANTE B1 (x4)
ID 1156 | ID 3333
ID 3844 | ID 3968

	0	1	2
0	-	B	C
1	B	-	C
2	C	C	B

FINE PATTERN 2.2

PATTERN 3.1 (x16)

VARIANTE A1 (x2)
ID 352 | ID 2272

	0	1	2
0	A	A	A
1	-	-	-
2	A	A	A

VARIANTE B1 (x2)
ID 1433 | ID 2292

	0	1	2
0	A	B	B
1	-	-	-
2	A	B	B

VARIANTE B2 (x2)
ID 2812 | ID 2893

	0	1	2
0	A	B	A
1	-	-	-
2	A	B	A

VARIANTE B3 (x2)
ID 2832 | ID 3623

	0	1	2
0	A	A	B
1	-	-	-
2	A	A	B

VARIANTE C1 (x2)
ID 378 | ID 3786

	0	1	2
0	A	A	A
1	-	-	-
2	C	C	C

VARIANTE C2 (x2)
ID 1444 | ID 3802

	0	1	2
0	A	B	B
1	-	-	-
2	B	A	A

VARIANTE C3 (x2)
ID 2900 | ID 4218

	0	1	2
0	A	B	A
1	-	-	-
2	B	A	B

VARIANTE C4 (x2)
ID 3628 | ID 4234

	0	1	2
0	A	A	B
1	-	-	-
2	B	B	A

FINE PATTERN 3.1

PATTERN 3.2 (x56)

VARIANTE A1 (x8)
ID 1056 | ID 1380
ID 2280 | ID 2288
ID 3182 | ID 3398
ID 3926 | ID 4094

	0	1	2
0	-	B	B
1	B	-	-
2	C	B	B

VARIANTE B1 (x8)
ID 1110 | ID 1374
ID 1407 | ID 2286
ID 2294 | ID 3155
ID 3425 | ID 3477

	0	1	2
0	-	B	B
1	C	-	-
2	C	B	B

VARIANTE B2 (x8)
ID 1078 | ID 1318
ID 1419 | ID 3151
ID 3177 | ID 3225
ID 3790 | ID 3806

	0	1	2
0	-	B	B
1	B	-	-
2	B	C	C

VARIANTE C1 (x8)
ID 404 | ID 482
ID 1080 | ID 3186
ID 3788 | ID 3792
ID 3842 | ID 3954

	0	1	2
0	-	B	B
1	B	-	-
2	C	C	C

VARIANTE C2 (x8)
ID 1132 | ID 1392
ID 3394 | ID 3420
ID 3796 | ID 3800
ID 4066 | ID 4178

	0	1	2
0	-	B	B
1	C	-	-
2	B	C	C

VARIANTE C3 (x8)
ID 2666 | ID 2874
ID 4058 | ID 4148
ID 4170 | ID 4220
ID 4224 | ID 4304

	0	1	2
0	-	B	C
1	C	-	-
2	B	C	B

VARIANTE C4 (x8)
ID 1210 | ID 3576
ID 3850 | ID 3914
ID 3962 | ID 4228
ID 4232 | ID 4330

	0	1	2
0	-	B	C
1	B	-	-
2	C	C	B

FINE PATTERN 3.2

PATTERN 3.3 (x28)

VARIANTE A1 (x4)
ID 1298 | ID 2370
ID 2474 | ID 3218

	0	1	2
0	-	B	-
1	B	-	B
2	B	C	B

VARIANTE B1 (x4)
ID 1352 | ID 2373
ID 2751 | ID 3461

	0	1	2
0	-	B	-
1	C	-	C
2	C	B	C

VARIANTE B2 (x4)
ID 2481 | ID 2643
ID 3921 | ID 4083

	0	1	2
0	-	B	-
1	B	-	B
2	C	B	C

VARIANTE C1 (x4)
ID 456 | ID 2484
ID 3840 | ID 3948

	0	1	2
0	-	B	-
1	B	-	B
2	C	C	C

VARIANTE C2 (x4)
ID 1158 | ID 2718
ID 3950 | ID 4062

	0	1	2
0	-	B	-
1	C	-	B
2	B	C	C

VARIANTE C3 (x4)
ID 2640 | ID 2744
ID 4056 | ID 4164

	0	1	2
0	-	B	-
1	C	-	C
2	B	C	B

VARIANTE C4 (x4)
ID 2510 | ID 3342
ID 3846 | ID 4166

	0	1	2
0	-	B	-
1	B	-	C
2	C	C	B

FINE PATTERN 3.3

PATTERN 3.4 (x20)

VARIANTE A1 (x2)
ID 1312 | ID 3967

	0	1	2
0	A	B	-
1	A	-	B
2	-	A	A

VARIANTE A2 (x2)
ID 3229 | ID 3853

	0	1	2
0	A	A	-
1	C	-	A
2	-	C	A

VARIANTE A3 (x2)
ID 3937 | ID 4099

	0	1	2
0	A	B	-
1	B	-	B
2	-	B	A

VARIANTE B1 (x2)
ID 1314 | ID 3958

	0	1	2
0	A	A	-
1	C	-	A
2	-	C	C

VARIANTE B2 (x2)
ID 1366 | ID 3964

	0	1	2
0	A	B	-
1	B	-	B
2	-	A	A

VARIANTE B3 (x2)
ID 3238 | ID 3854

	0	1	2
0	A	B	-
1	A	-	B
2	-	A	B

VARIANTE B4 (x2)
ID 3472 | ID 3856

	0	1	2
0	A	B	-
1	B	-	A
2	-	B	A

VARIANTE B5 (x2)
ID 4070 | ID 4174

	0	1	2
0	A	B	-
1	B	-	A
2	-	A	B

VARIANTE B6 (x2)
ID 3940 | ID 4072

	0	1	2
0	A	B	-
1	A	-	B
2	-	B	A

VARIANTE B7 (x2)
ID 4096 | ID 4180

	0	1	2
0	A	A	-
1	C	-	C
2	-	C	A

FINE PATTERN 3.4

PATTERN 4.1 (x28)

VARIANTE A1 (x4)
ID 771 | ID 1402
ID 3043 | ID 3393

	0	1	2
0	-	-	B
1	-	-	B
2	B	B	C

VARIANTE B5 (x4)
ID 798 | ID 3049
ID 3150 | ID 3589

	0	1	2
0	-	-	B
1	-	-	C
2	B	B	C

VARIANTE B1 (x4)
ID 477 | ID 783
ID 1430 | ID 3033

	0	1	2
0	-	-	B
1	-	-	B
2	C	C	C

VARIANTE B6 (x4)
ID 809 | ID 2888
ID 3045 | ID 3384

	0	1	2
0	-	-	B
1	-	-	C
2	C	C	B

VARIANTE B2 (x4)
ID 774 | ID 1206
ID 1429 | ID 3529

	0	1	2
0	-	-	B
1	-	-	B
2	B	C	C

FINE PATTERN 4.1

VARIANTE B3 (x4)
ID 780 | ID 1403
ID 2664 | ID 3519

	0	1	2
0	-	-	B
1	-	-	B
2	C	B	C

VARIANTE B4 (x4)
ID 393 | ID 782
ID 3051 | ID 3141

	0	1	2
0	-	-	B
1	-	-	B
2	C	C	B

PATTERN 4.2 (x32)

VARIANTE A1 (x4)
ID 1055 | ID 2279
ID 2468 | ID 3164

	0	1	2
0	-	B	B
1	B	-	-
2	-	B	B

VARIANTE C4 (x4)
ID 2712 | ID 3896
ID 4229 | ID 4305

	0	1	2
0	-	B	C
1	B	-	-
2	-	C	B

VARIANTE B1 (x4)
ID 1109 | ID 2748
ID 2285 | ID 3407

	0	1	2
0	-	B	B
1	C	-	-
2	-	B	B

VARIANTE C5 (x4)
ID 2478 | ID 2825
ID 3894 | ID 4326

	0	1	2
0	-	B	C
1	B	-	-
2	-	B	C

VARIANTE C1 (x4)
ID 1079 | ID 2496
ID 3411 | ID 3797

	0	1	2
0	-	B	B
1	C	-	-
2	-	C	C

VARIANTE C6 (x4)
ID 2504 | ID 4139
ID 4223 | ID 4332

	0	1	2
0	-	B	C
1	C	-	-
2	-	C	B

FINE PATTERN 4.2

VARIANTE C2 (x4)
ID 1133 | ID 2720
ID 3168 | ID 3791

	0	1	2
0	-	B	B
1	B	-	-
2	-	C	C

VARIANTE C3 (x4)
ID 2738 | ID 2819
ID 4137 | ID 4299

	0	1	2
0	-	B	C
1	C	-	-
2	-	B	C

PATTERN 5.1 (x32)

ID 769	ID 772
ID 778	ID 781
ID 796	ID 799
ID 805	ID 808
ID 1066	ID 1068
ID 1120	ID 1122
ID 3034	ID 3036
ID 3040	ID 3042
ID 3169	ID 3178
ID 3412	ID 3421
ID 3520	ID 3522
ID 3526	ID 3528
ID 3898	ID 3907
ID 4141	ID 4150
ID 4318	ID 4319
ID 4345	ID 4346

	0	1	2
0	-	?	?
1	?	-	-
2	?	-	-

FINE PATTERN 5.1

PATTERN 5.2 (x8)

ID 2461	ID 2464
ID 2488	ID 2491
ID 2704	ID 2707
ID 2731	ID 2734

	0	1	2
0	-	?	-
1	?	-	?
2	-	?	-

FINE PATTERN 5.2

PATTERN 6.1 (x16)

ID 14	ID 15
ID 17	ID 18
ID 365	ID 758
ID 759	ID 785
ID 786	ID 1458
ID 2916	ID 3029
ID 3047	ID 3515
ID 3533	ID 3645

	0	1	2
0	?	?	?
1	-	-	-
2	-	-	-

FINE PATTERN 6.1

Bibliografia

[1] Bacci, E.: *Applicazione di crittografia visuale per computazioni omomorfiche*. Università di Pisa (2023)

[2] Beaver, D.: *Perfect Privacy for Two-Party Protocols*. TR-11-89, Harvard University (1989)

[3] Benaloh, J. C.: *Secret sharing homomorphisms: keeping shares of a secret secret* (extended abstract). In: Odlyzko, A.M. (ed.) Crypto 1986. LNCS, vol. 263, pp. 251–260. Springer, Heidelberg (1987)

[4] Cimato, S., De Prisco, R., De Santis, A.: *Probabilistic visual cryptography schemes*. Comput. J. 49(1), 97–107 (2006)

[5] Cimato, S., Yang, C.-N.: *Visual Cryptography and Secret Image Sharing*. CRC Press, Boca Raton (2012)

[6] D'Arco, P., De Prisco, R., Desmedt, Y.: *Private visual share homomorphic computation and randomness reduction in visual cryptography*. In Anderson C.A. Nascimento and Paulo Barreto, editors, Information Theoretic Security, pages 95–113, Cham, 2016. Springer International Publishing (2016)

[7] D'Arco, P., De Prisco, R.: *Secure two-party computation: A visual way*. In Carles Padró, editor, Information Theoretic Security - 7th International Conference, ICITS 2013, Singapore, November 28-30, 2013, Proceedings, volume 8317 of Lecture Notes in Computer Science, pages 18–38. Springer (2013)

[8] De Bonis, A., De Santis, A.: *Randomness in secret sharing and visual cryptography schemes*. Theor. Comput. Sci. 314(3), 351–374 (2004)

[9] De Prisco, R., De Santis, A.: *On the relation of random grid and deterministic visual cryptography*. IEEE Trans. Inf. Forensics Secur. 9(4), 653–665 (2014)

[10] Dieci, A.: *Studio di funzioni privatamente computabili in logica multivalore per la crittografia visuale*. Università degli Studi di Milano (2023)

[11] Dyala, R. I., Je, S. T., Rosni A.: *An overview of visual cryptography techniques.* Multim. Tools Appl., 80(21), 31927–31952 (2021)

[12] Kafri, O., Keren, E.: *Encryption of pictures and shapes by random grids.* Opt. Lett. 12(6), 377–379 (1987)

[13] Kushilevitz, E.: *Privacy and communication complexity.* In: Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS), pp. 416–421 (1989)

[14] Lisi, G.: *Secure Two-Party Computation: Analisi delle funzioni privatamente computabili su domini multivalore.* Università degli Studi di Milano (2023)

[15] Naor, M., Shamir, A.: *Visual cryptography.* In: Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 1–12. Springer, Heidelberg (1995)

[16] Narad, S. K., Sayankar, M. R., Alone, S. V., Mahiskar, P. S.: *Secret sharing scheme for group authentication – a review.* In 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), volume 1, pages 12–16 (2017).

[17] Punithavathi, P., Geetha, S.: *Visual cryptography: A brief survey.* Inf. Secur. J. A Glob. Perspect., 26(6):305–317 (2017)

[18] Sahni, G. K., Vege, H. K.: *Schemes and Applications of Visual Cryptography.* International Journal of Emerging Technologies in Engineering Research (IJETER) Volume 8, Issue 6, June (2020)

[19] Shamir, A.: *How to share a secret.* Communications of the ACM. 22(11), 612–613 (1979)

[20] Snowden, E.: *Errore di sistema.* Longanesi (2019)

[21] Yadav, S., Shrivas, B.: *A survey on visual cryptography techniques and their applications.* International Journal of Computer Science and Information Technologies (2015)

[22] Yang, C.-N.: *New visual secret sharing schemes using probabilistic method.* Pattern Recogn. Lett. 25, 481–494 (2004)

[23] Yao, A. C.: *Protocols for Secure Computations.* Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, pp. 160-164 (1982)