GENERALISED IF-THEN-ELSE OPERATOR FOR COMPACT POLYNOMIAL REPRESENTATION OF MULTI OUTPUT FUNCTIONS

Ilya Levin, Osnat Keren

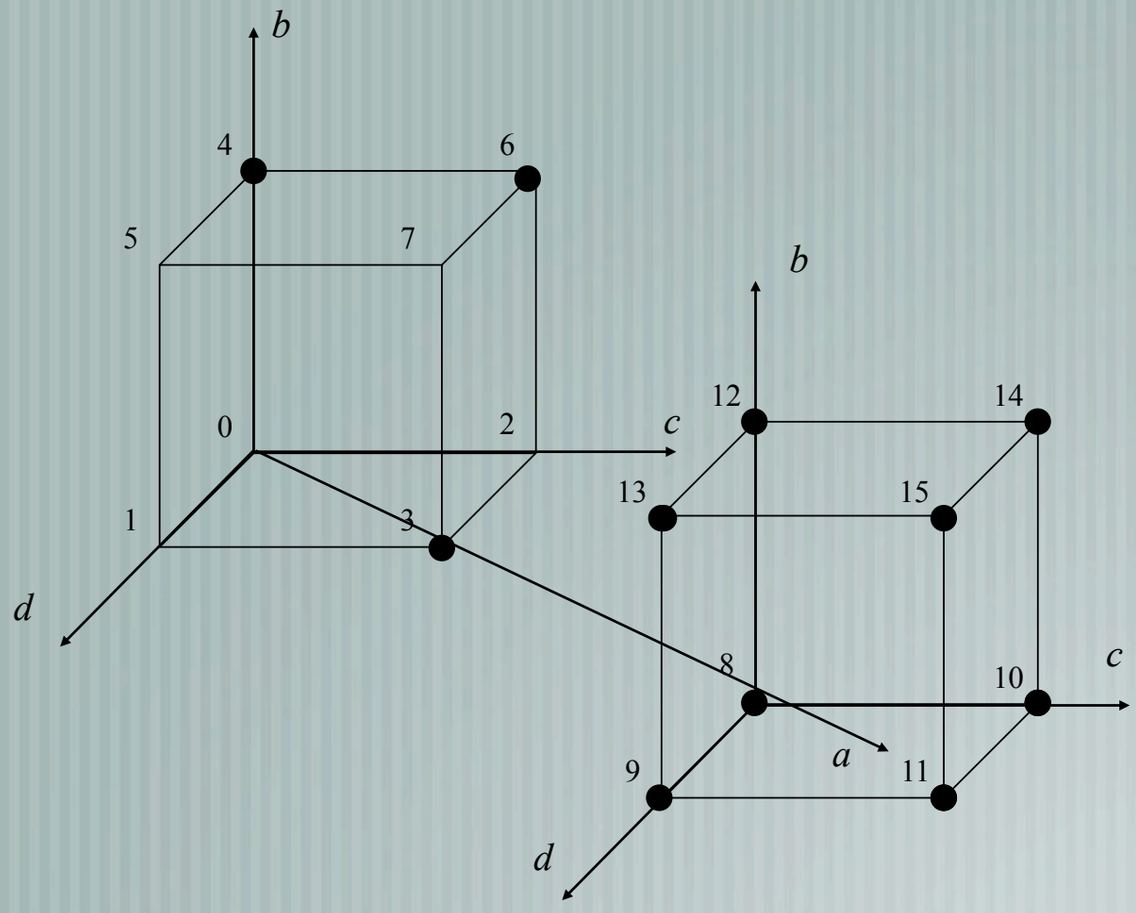
Tel Aviv University, Bar Ilan University, Israel

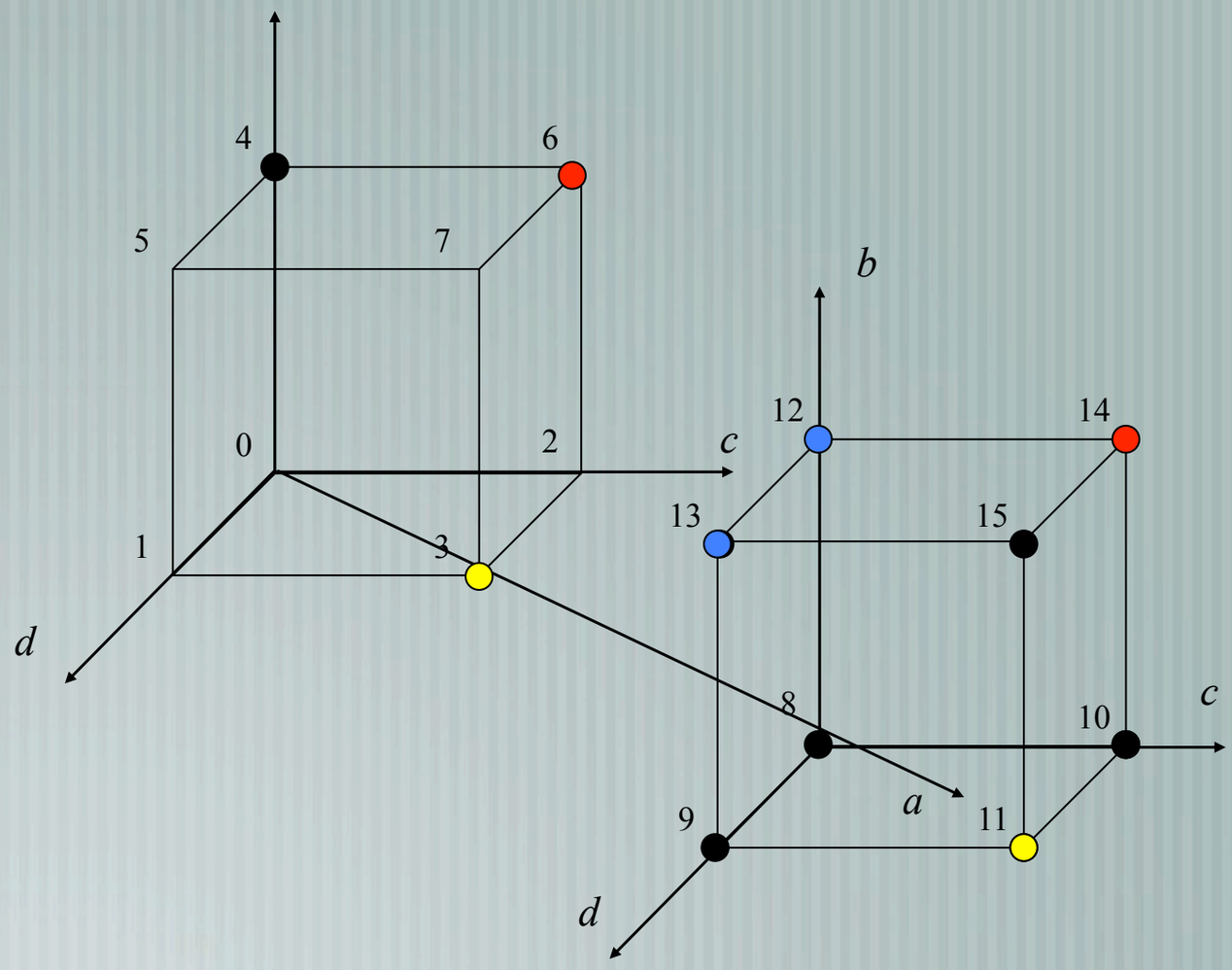
DSD 2011

Outline

- *Logic functions vs. System of Logic functions*
- *Preliminaries*
- *Generalised ITE (GITE) operator*
- *EvP and ExP*
- *Partition algebra of EvP*
- *Boolean algebra of ExP*
- *Dichotomy property*
- *Decomposition*
- *Conclusions*

Logic Function vs. System of Logic Functions

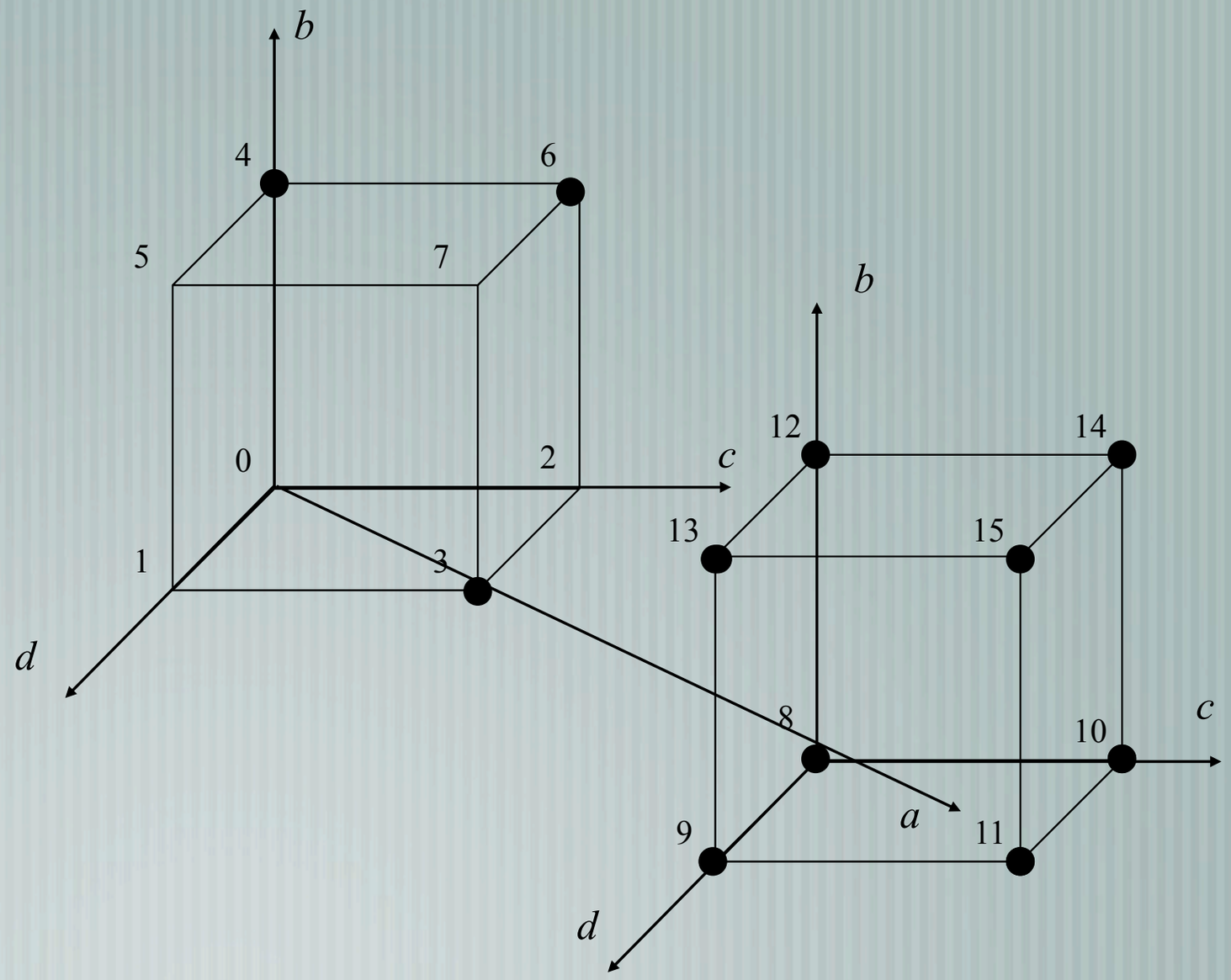




Logic Function

I. Single Logic function

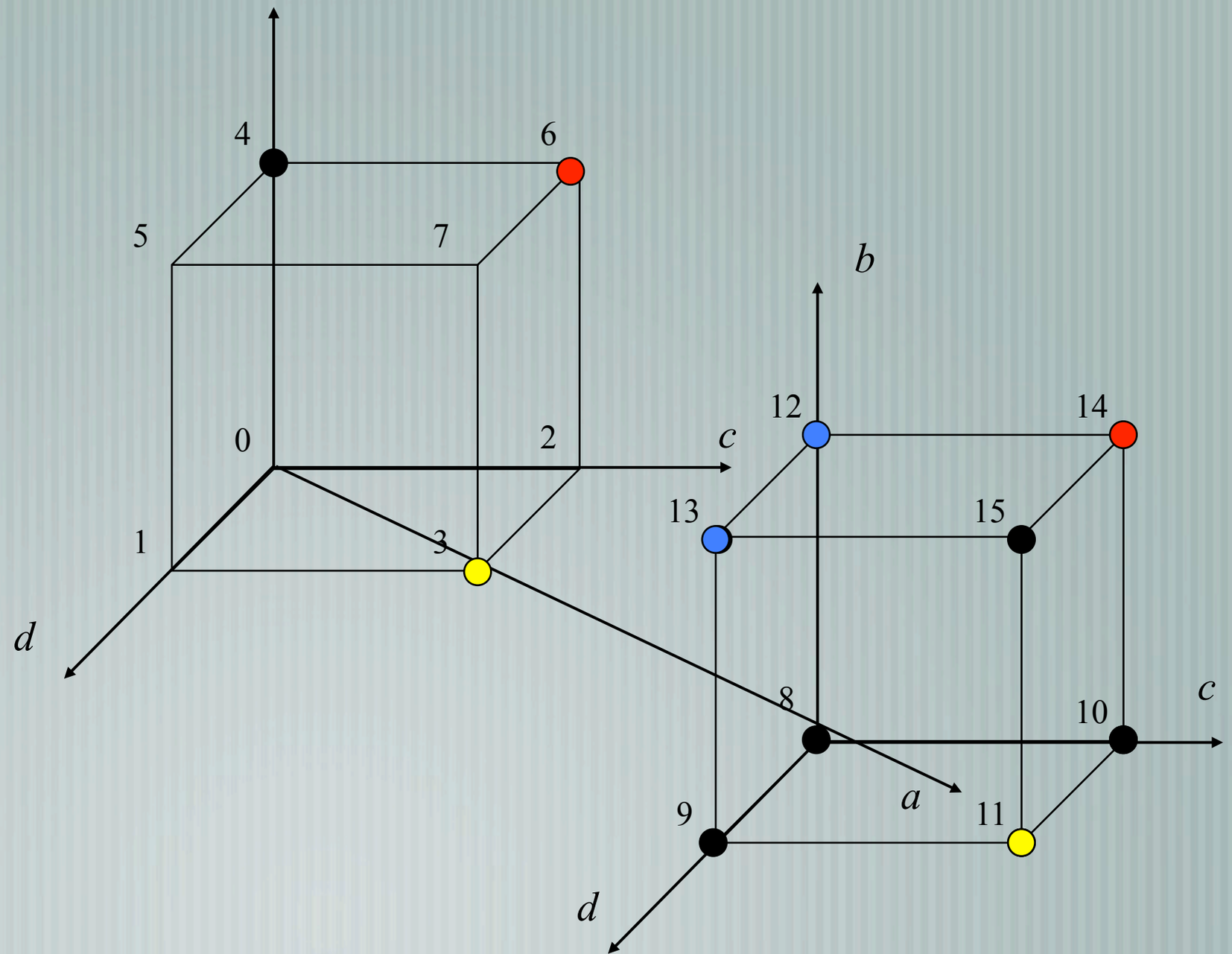
II. Two-block Partition of Boolean Cube



System of Logic Functions

I. n Logic functions

II. n-block Partition of Boolean Cube



Two Domains

- Boolean Algebra of output vectors
- Partitions Algebra of input vectors

Partitions

Definition.

A partition on a set C is a collection of disjoint subsets of C whose set union is C, i.e. $\pi = \{B_\alpha\}$ such that:

$$B_\alpha \cap B_\beta = \varnothing \ (\alpha \neq \beta) \text{and } \bigcup \{B_\alpha\} = C.$$

Example: $S = \{1,2,3,4,5,6,7,8\}$

$$\pi_1 = \{\{1\},\{2\},\{3,4,7\},\{5,6,8\}\} = \{\bar{1}; \bar{2}; \overline{3,4,7}; \overline{5,6,8}\}$$

Partitions

A product $\pi_{prd} = \pi_1 \bullet \pi_2$ of partitions π_1 and π_2 is a partition comprising intersections of blocks π_1 and π_2:

$s \equiv t(\pi_1 \bullet \pi_2)$ *iff* $s \equiv t(\pi_2) \,\&\, s \equiv t(\pi_1)$.

A sum $(\pi_1 + \pi_2)$ of the partitions π_1 and π_2 defined as follows:

$s \equiv t(\pi_1 + \pi_2)$ *iff* a chain $s_0, s_1, \ldots, s_n$ exists in C such as:

$s = s_0, s_1, \ldots, s_n = t$, for which either $s_i \equiv s_{i+1}(\pi_1)$ or

$s_i \equiv s_{i+1}(\pi_2)$, $0 \leq i \geq n-1$.

Algebra of Partitions

The algebraic structure of partitions is known as a lattice.

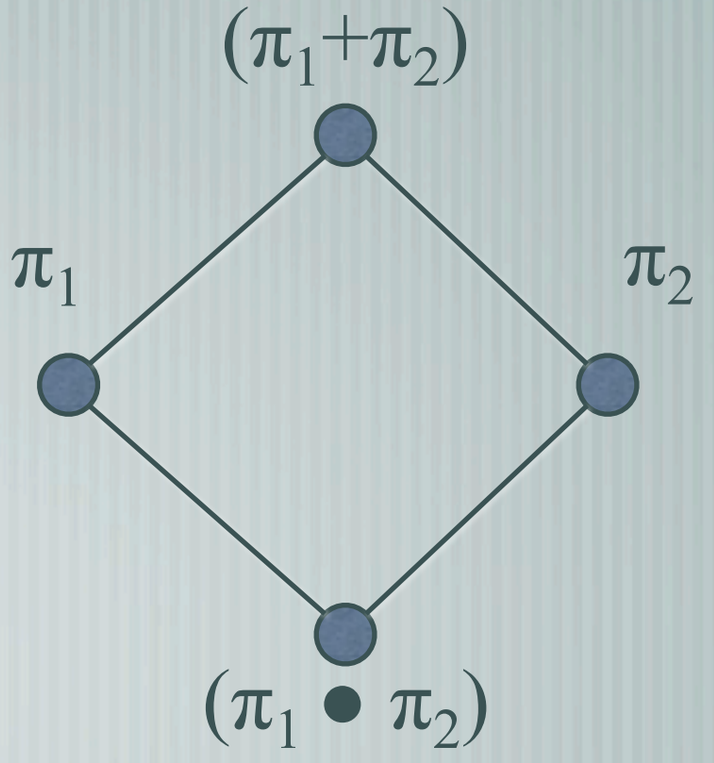
This lattice has both

Zero (the smallest partition π^0) and

One (the biggest partition π^1) elements defined as follows:

$$\pi^0 = \left\{\overline{s_1}; \ldots; \overline{s_m}\right\};$$

$$\pi^1 = \left\{\overline{s_1, \ldots, s_m}\right\}.$$



Algebraic Decision Diagrams (ADD)

✓Proposed in 1993 by R. Baher, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Parvo, F. Somenzi

✓Multi Output Functions as ADD

✓Different forms of representation of ADDs

✓Operations: Apply and If-Then-Else operation

✓Used for: matrix multiplication, shortest path algorithms, and numerical linear algebra.

Algebraic Decision Diagram

An ADD is a function:

$$f : \{0,1\}^n \rightarrow S$$

where S is the finite carrier of the algebraic structure.

ADD is a form for representation of Multi Output Functions (MOF).

Algebraic Decision Diagram

- ✓ADDs representations
  - MTBDD
  - Matrix

- ✓ADD operations
  - Apply
  - If-Then-Else (ITE)

Apply operation

$$\boldsymbol{Apply}(f, g, op) = f \ \boldsymbol{op}\ g$$

$$f = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}; \quad g = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix}$$

$$\boldsymbol{Apply}(f, g, +) = \begin{pmatrix} 5 & 5 & 4 & 4 \\ 5 & 5 & 4 & 4 \\ 2 & 2 & 3 & 3 \\ 2 & 2 & 3 & 3 \end{pmatrix}$$

If-Then-Else (ITE) operation

$$\boldsymbol{ITE}(f,g,h) = f \bullet g + \bar{f} \bullet h$$

$$f = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}; g = \begin{pmatrix} 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \end{pmatrix}; h = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix}$$

$$\boldsymbol{ITE}(f,g,h) = \begin{pmatrix} 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \\ 2 & 2 & 3 & 3 \\ 2 & 2 & 3 & 3 \end{pmatrix}$$

The ITE comprises a Boolean function operation as a “binary condition” being a two-block partition of the Boolean space

ITE vs. GITE

- ITE - two-block partition on the Boolean cube
- GITE - n-block partition on the Boolean cube

Generalised ITE operation

Definition. Generalized ITE (GITE) is

$\boldsymbol{GITE}(\pi, Y)$

where: $\pi = \{B_1; \ldots; B_m\}$ is a partition on the Boolean space;

$Y = \{Y_1, \ldots, Y_m\}$ – a set of operators - binary vectors.

$Y_i \ (i = 1, \ldots, m)$ corresponds to a certain block B_i of the partition π.

Thus, GITE consists of a partition portion (Evolution Part) and an operator portion (Execution Part).

Multi Output Function as GITE

Definition

A Multi – Output Function (MOF)

is a mapping $f : \{0,1\}^n \to Y$,

which is ***GITE***(π, Y) defined on two sets:

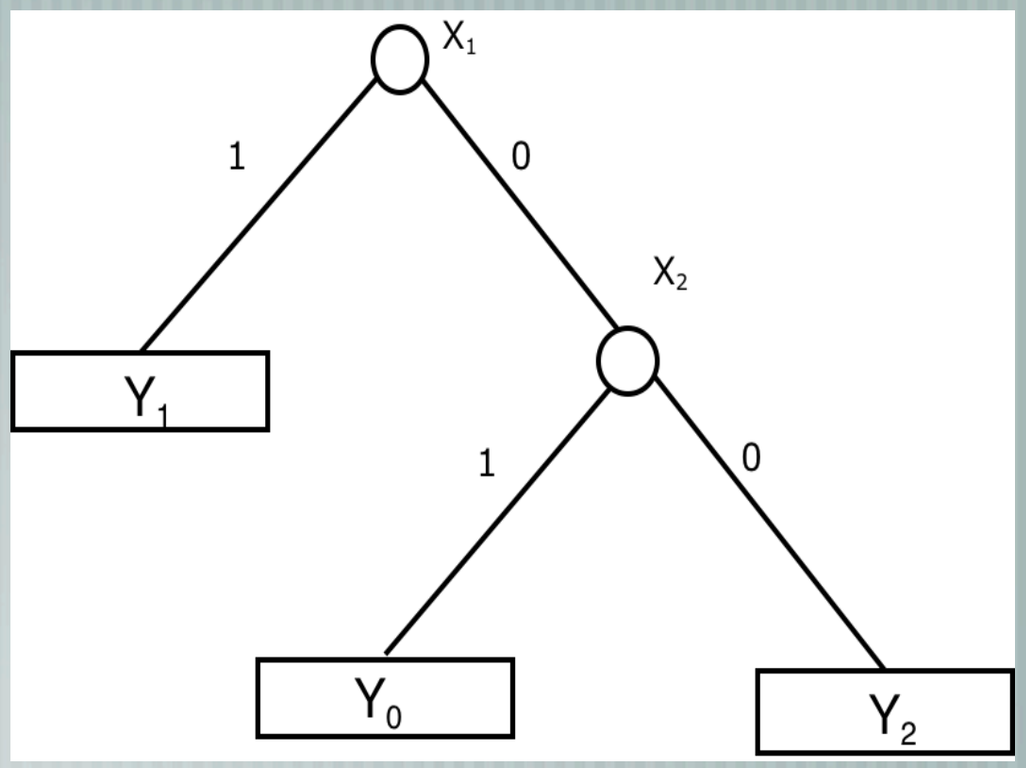
a) the partitions π and

b) the set Y of operators.

Polynomial representation of GITE

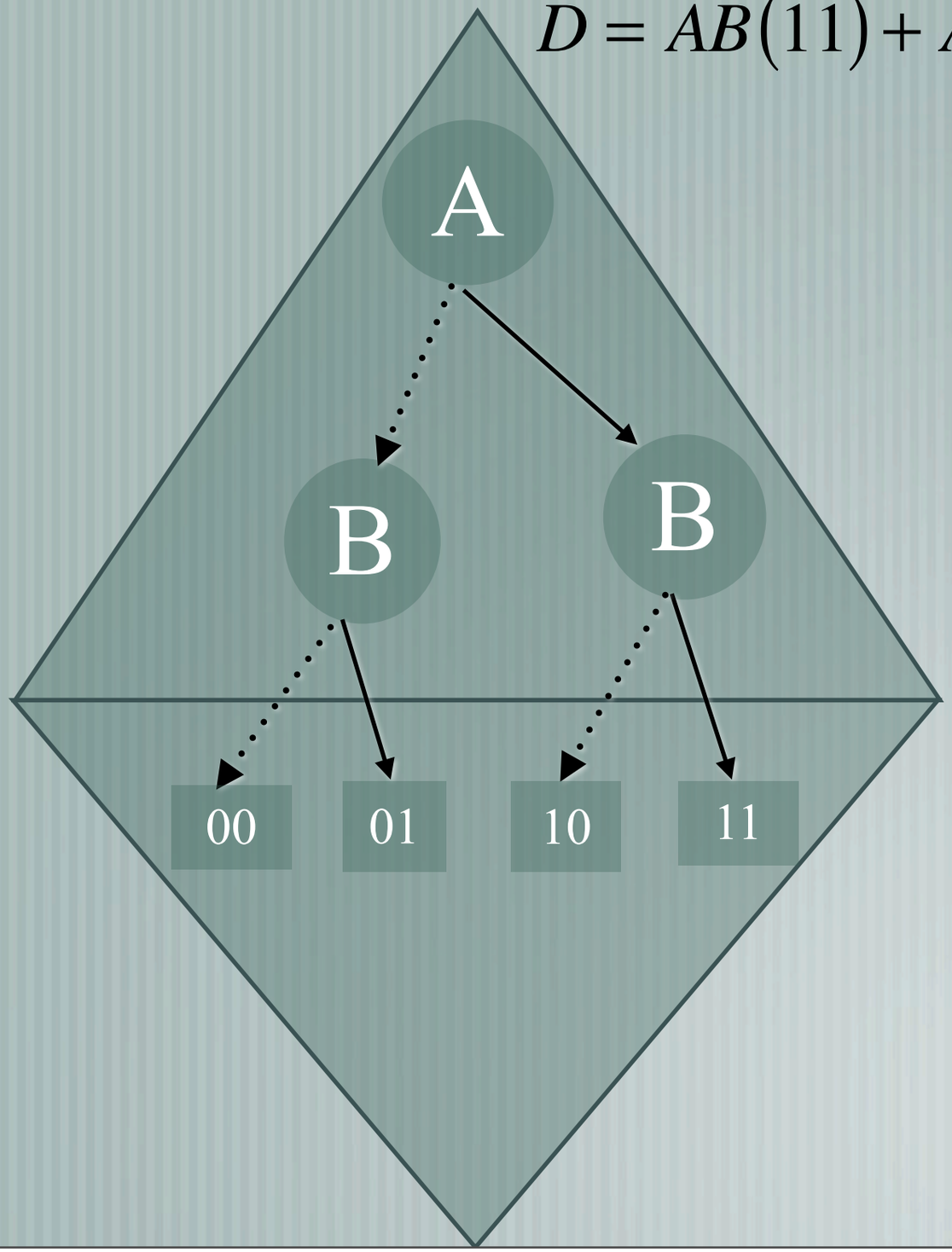
Example : $D = B_1 Y_1 + B_2 Y_2 + B_0 Y_0$

$B_1 = x_1;\ B_2 = \overline{x}_1 \overline{x}_2;\ B_0 = \overline{\overline{B_1 + B_2}}$



GITE

$$D = AB(11) + A\bar{B}(10) + \bar{A}B(01) + \bar{A}\bar{B}(00)$$



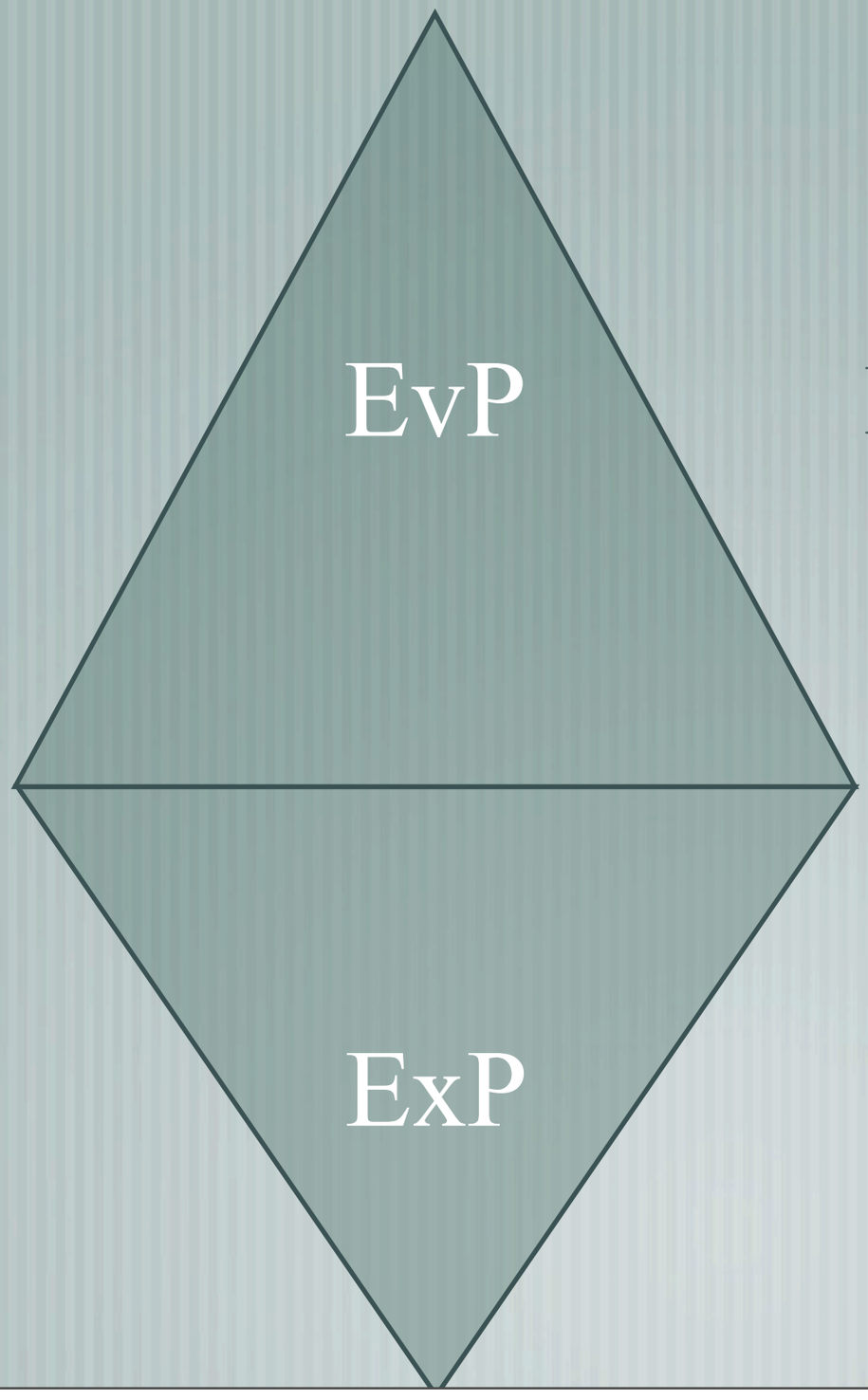
X

EvP

ExP

Y

GITE algebra is a product of two algebras



Partitions Algebra

Boolean Algebra

GITE algebra

The GITE algebra is a product of two algebras:

- the algebra of partitions on the Evolution Part
- the Boolean algebra on the Execution Part

Partitions Algebra of GITE Evolution Part

GITE Apply operation

Definition : GITE Apply operation

$$\boldsymbol{Apply}\left(D_a, D_b, op\right) = D_a op D_b =$$

$$= \boldsymbol{GITE}\left(\pi_a \bullet \pi_b; Y_{a1} op Y_{b1}, Y_{a1} op Y_{b2} \ldots, Y_{am} op Y_{bm}\right),$$

GITE Apply operation is performed by multiplying partitions π and by pair-wise op operation on operators Y.

GITE Apply operation

Definition. Apply operation on GITE-polynomial
we call *Product* of GITE-polynomials and define as follows:

Let $D_1 = \sum_{i=1}^{m} Y_i + B_0^1 Y_0,\ D_2 = \sum_{k=1}^{l} Y_k + B_0^2 Y_0.$

$$D_1 \circ D_2 = Apply\left(D_1,\ D_2, op\right) = \Sigma\left(B_{ij}^1 \cdot B_{kl}^2\right)\left\{Y_i op Y_k\right\},$$

for each pair of terms from D_1 and D_2,

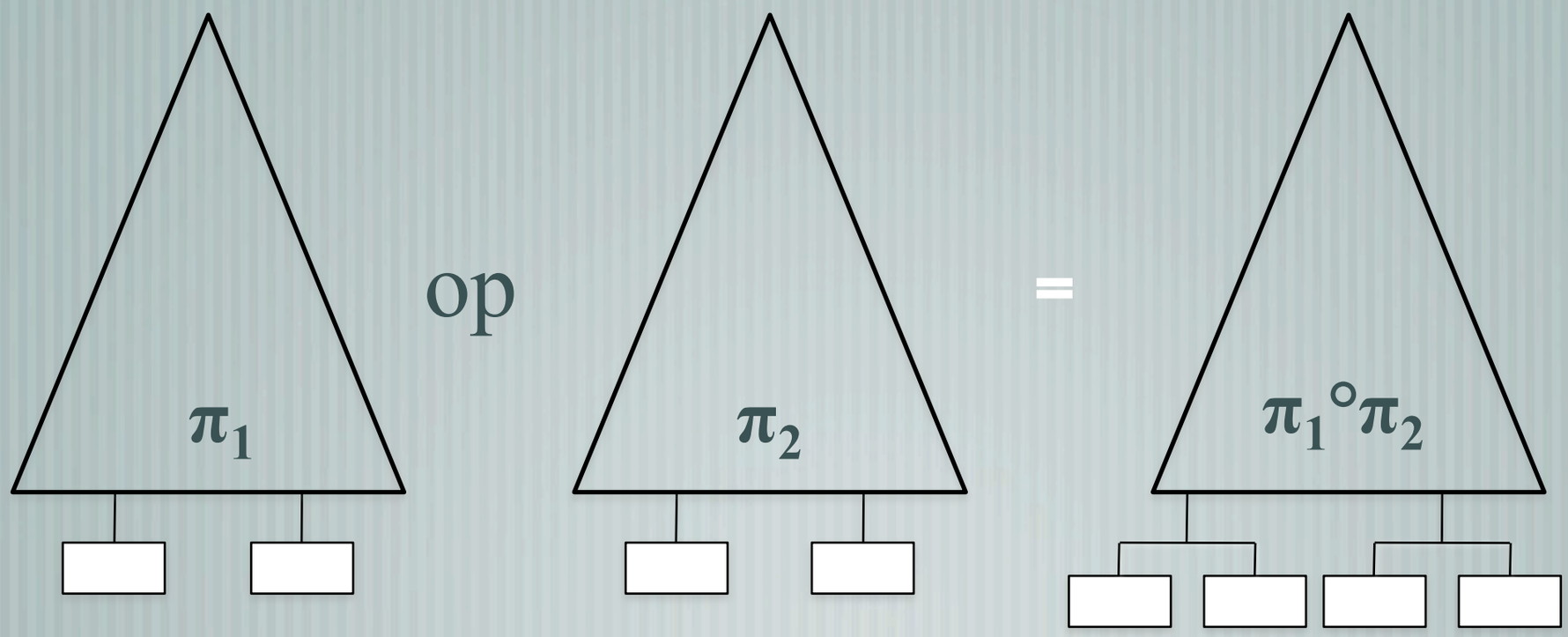
$B_{ij}^1 \cdot B_{kl}^2$ is a logic product (AND) of *B*-functions;

$Y_i op Y_j$ - is the Apply operation between Y_i and Y_k

GITE Apply operation

The Apply operation between $D_1 = \boldsymbol{GITE}\left(\pi_1; Y_{11},\ldots,Y_{1m}\right)$ and $D_2 = \boldsymbol{GITE}\left(\pi_2; Y_{21},\ldots,Y_{2m}\right)$:

$$D_{op} = \boldsymbol{Apply}\left(D_1, D_2\right) = D_1\, op\, D_2 = \boldsymbol{GITE}\left(\pi_1 \bullet \pi_2; Y_{11}\, op\, Y_{21},\ldots,Y_{1m}\, op\, Y_{2m}\right).$$



Factorization of GITE expressions

The product of GITE partitions corresponds to the Apply operation.

The sum of GITE partitions corresponds to *factorization* of GITEs.

Define the factorization of GITEs as follows:

$$D = D_i \, op \, D_j = \boldsymbol{GITE}\left(\pi_i + \pi_j; D_1, \ldots, D_f\right),$$

where f is a number of blocks in $\left(\pi_i + \pi_j\right)$

$D_1, \ldots, D_f$ stand for GITEs representing remaining functions.

Example

Let $D_1 = x_1 Y_1 + \bar{x}_1 \bar{x}_2 Y_2 + \bar{x}_1 x_2 Y_3, D_2 = \bar{x}_1 Y_4 + x_1 \bar{x}_3 Y_5 + x_1 x_3 Y_6.$

$$D_1 \circ D_2 = \boldsymbol{GITE}\left(\pi_1; Y_1, Y_2, Y_3\right) \circ \boldsymbol{GITE}\left(\pi_2; Y_4, Y_5, Y_6\right) =$$

$$= \boldsymbol{GATE}\left(\pi_1 + \pi_2; D_{23}, D_{456}\right);$$

$$D_1 \circ D_2 = D_3\left(D_{23}, D_{456}\right) = x_1 D_{23} + \bar{x}_1 D_{456};$$

where:

$$D_{23} = Y_1 \circ \left(\bar{x}_3 Y_5 + x_3 Y_6\right)$$

$$D_{456} = \left(\bar{x}_2 Y_2 + x_2 Y_3\right) \circ Y_4$$

Substitution

Let D_1, D_2, D_3 be:

$$D_1 = \boldsymbol{GITE}\left(\pi_1; Y_{11}, Y_{12}\right), D_2 = \boldsymbol{GITE}\left(\pi_2; Y_{21}, \ldots, Y_{2m}\right),$$

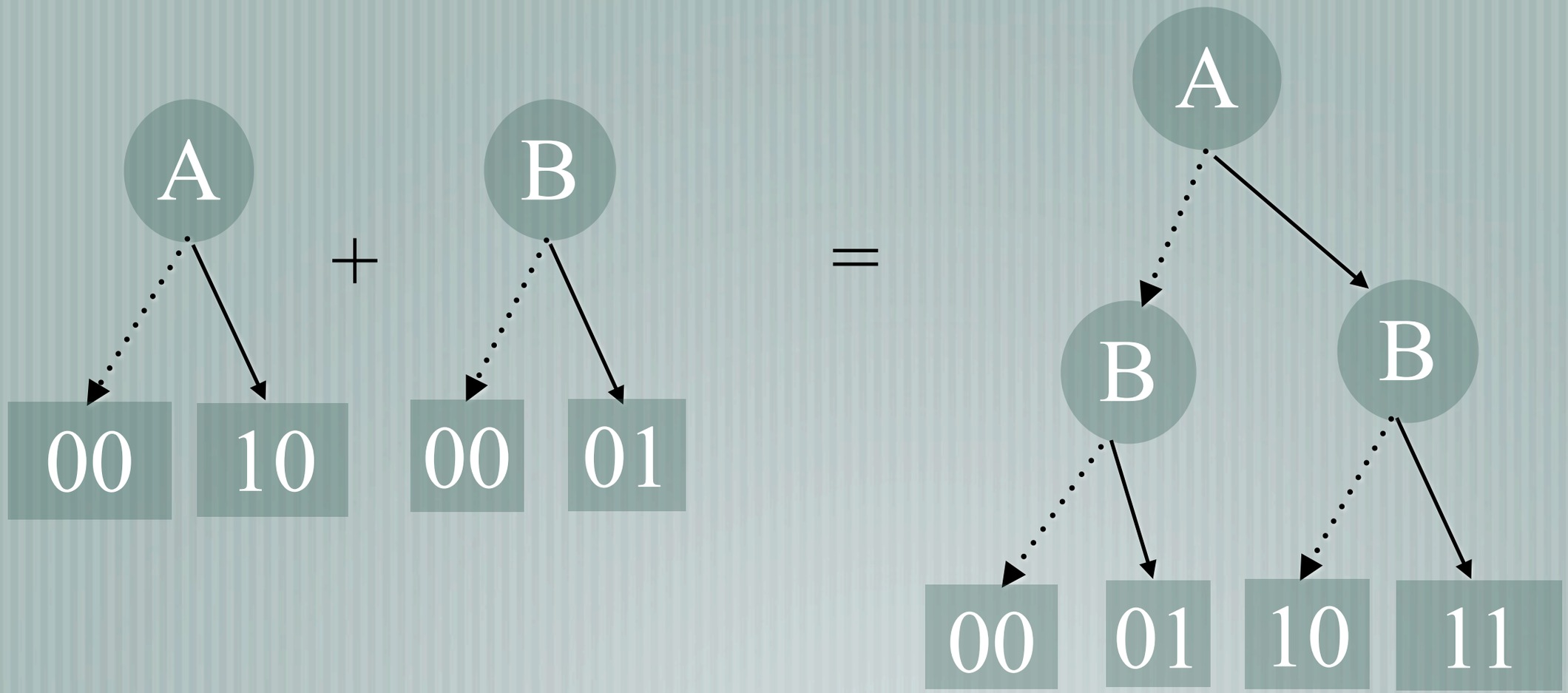
$$D_3 = \boldsymbol{GITE}\left(\pi_3; Y_{31}, \ldots, Y_{3m}\right)$$

After substitution: $Y_{11} \leftarrow D_2 \quad Y_{12} \leftarrow D_3$, we have:

$$D_1 = \boldsymbol{GITE}\left(\pi_1; D_2, D_3\right)$$

Boolean algebra of GITE Execution Part

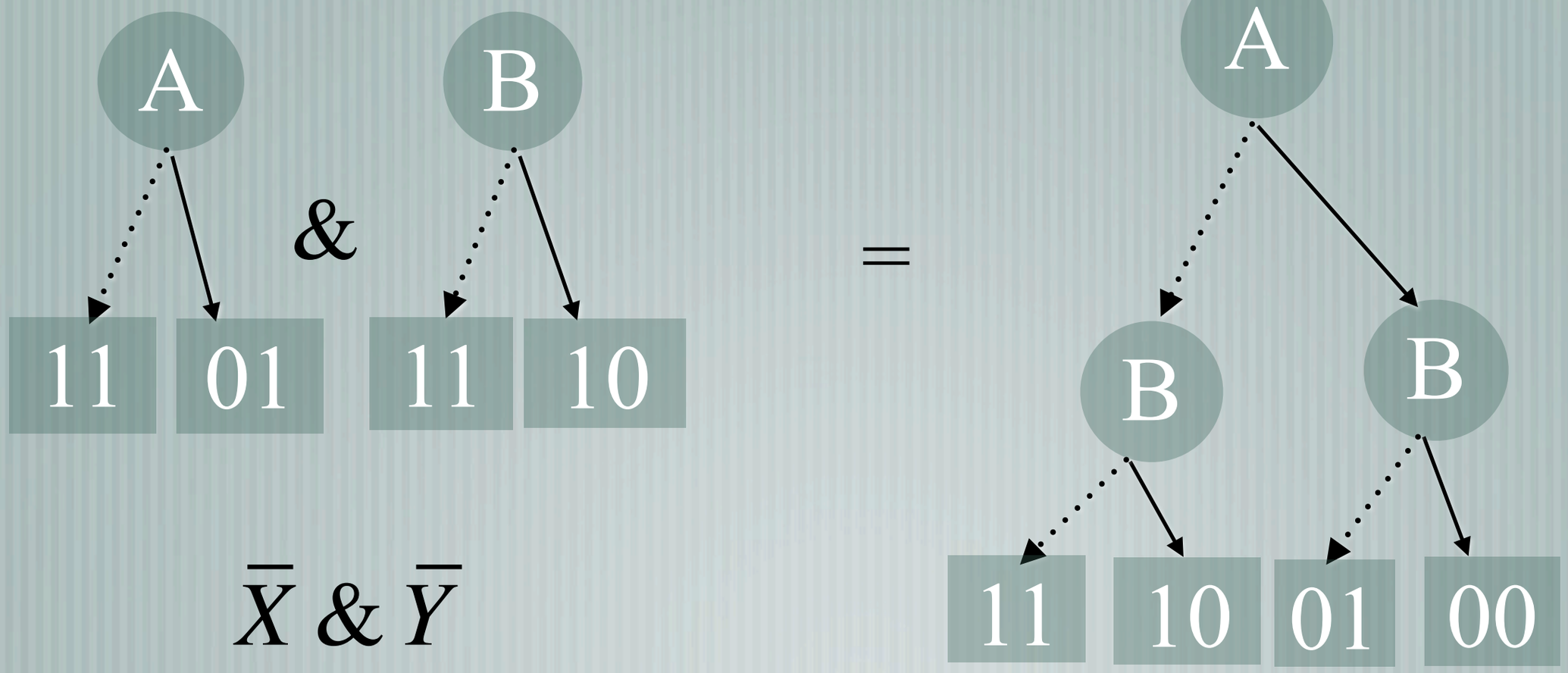
Boolean algebra of GITE Execution Part



$$X + Y$$

Boolean algebra of GITE Execution Part

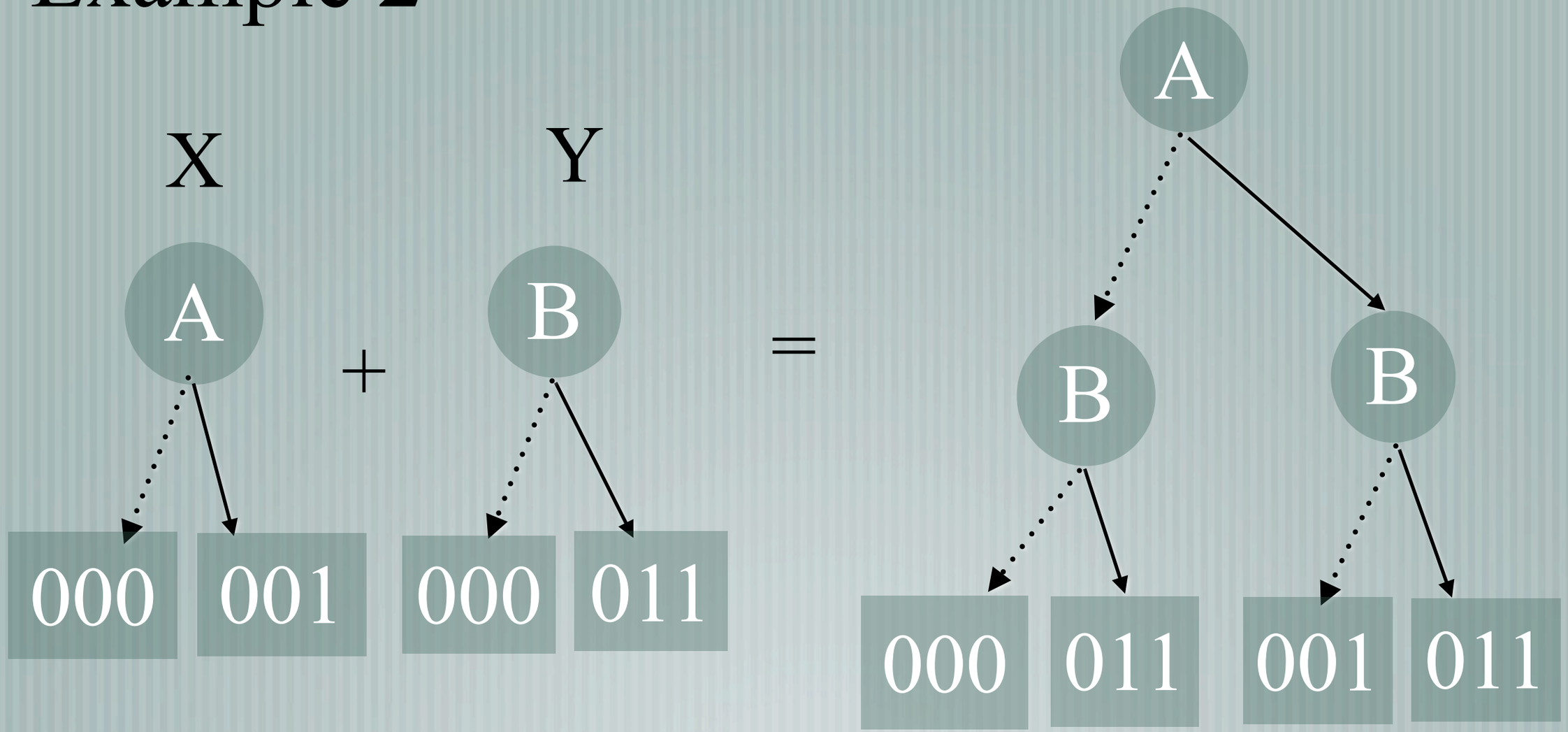
Example 1: $X + Y = \overline{(\bar{X} \,\&\, \bar{Y})}$



$\bar{X} \,\&\, \bar{Y}$

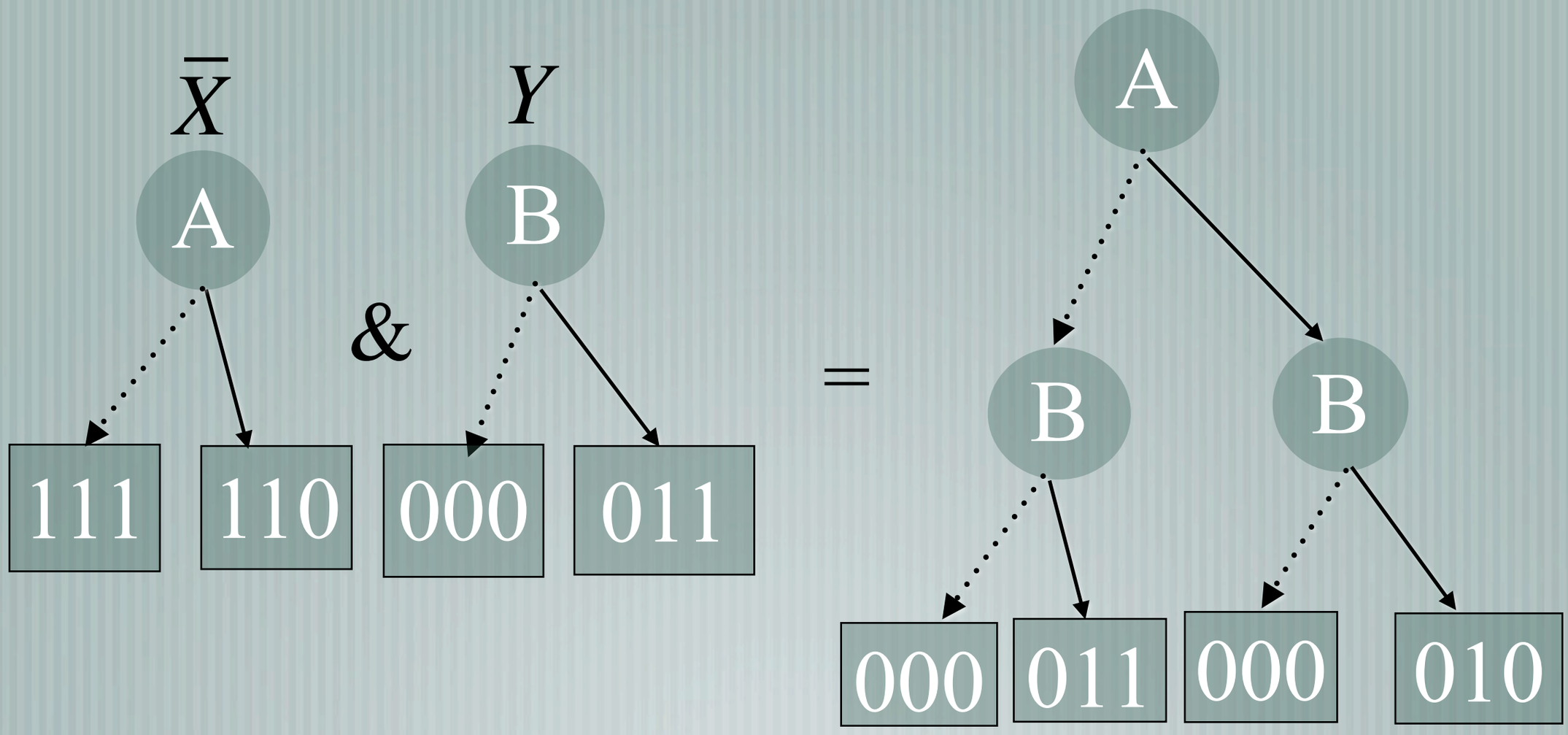
Boolean algebra of GITE Execution Part

Example 2



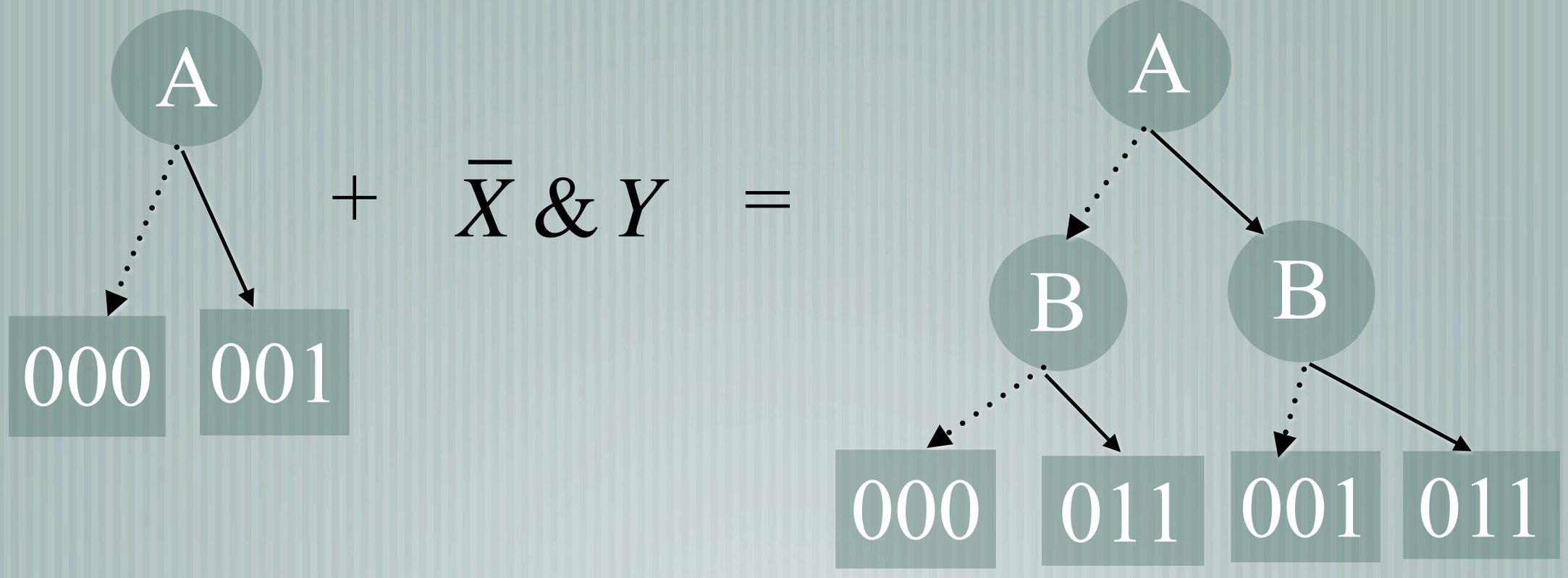
Boolean algebra of GITE Execution Part

Example 2



Boolean algebra of GITE Execution Part

Example 2: $X + \bar{X}\,\&\,Y = X + Y$



Decomposition

- Beginning from the initial implicant table to construct a network consisting of a number of component GITE
- Minimise component independently
- Each of the components have to be dichotomic

Dichotomic Fragment

We say that a set of product terms forms a *dichotomic* fragment,
if the set is straightforwardly mappable into an MTBDD.

The dichotomic property guarantees that there exists a Shannon expansion
that will not bring additional product terms to the initial GITE.

The dichotomic property means that the paths of the MTBDD
are in one-to-one correspondence with product terms of the GITE.

Dichotomy Property

We study cases where GITE is represented by a MTBDD.
Our hypothesis is that the GITE can be more efficiently represented by a set of dichotomic fragments.

The whole GITE would be considered a set of sub-GITE, functionally equal to the initial GITE.

Any GITE can be decomposed into a network of dichotomic fragments connected by the Apply and the Substitution operations.

Algebraic Decomposition Method

The proposed decomposition algorithm is based on grouping of the set of cubes representing the function to a set of blocks.

The algorithm is algebraic decomposition method.

Function F is represented as:

$F = D \circ Q + R$

where D, Q and R, are the divisor, quotient and remainder.

Algebraic Decomposition Method

Decomposition is performed simultaneously on the set
of functions that are represented as a single GITE - polynomial.
Our algebraic decomposition has the form:

$$D = \mathrm{GITE}\left(\pi_h, D_1, \ldots D_j\right) \circ R.$$

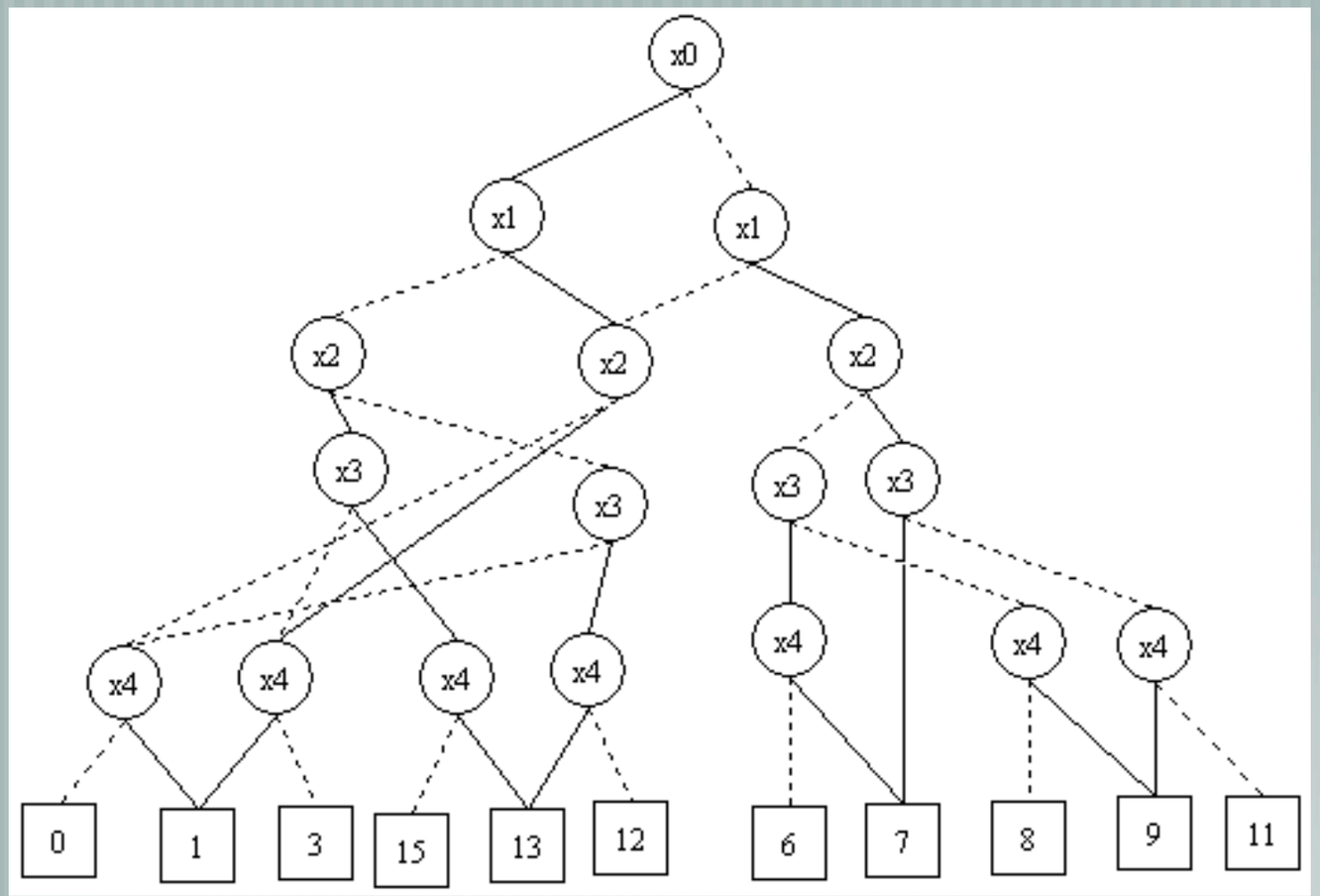
Where: divisor π_h is a block header,

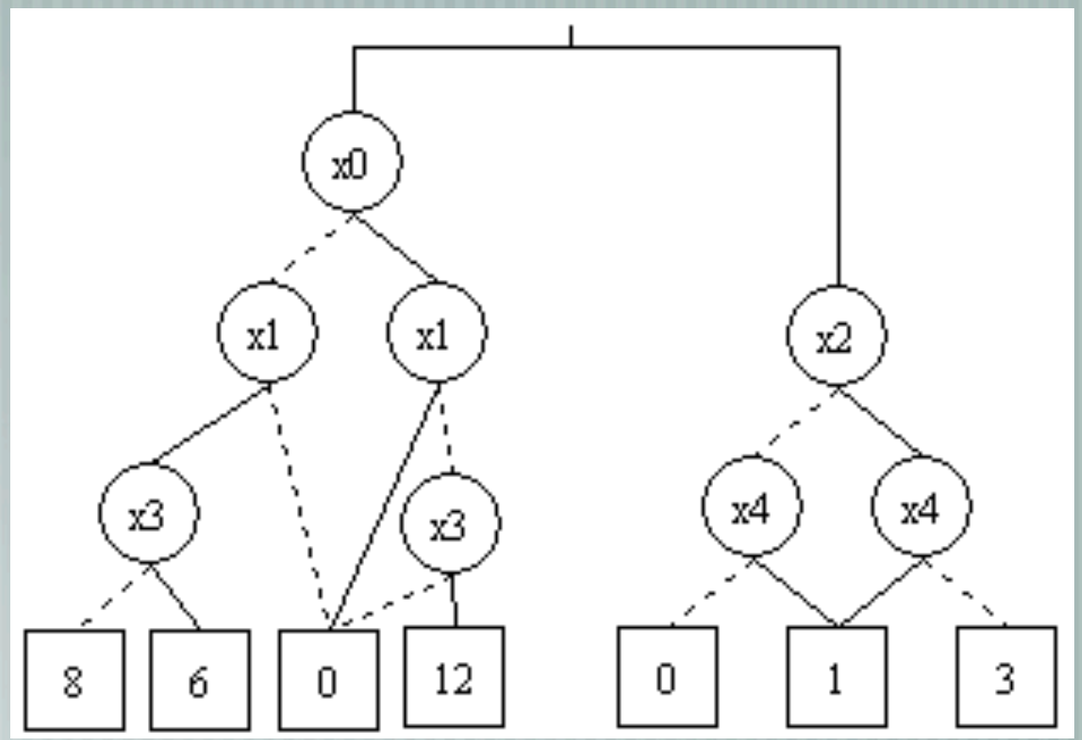
quotient $\left(D_1, \ldots D_j\right)$, D_i, $i = 1, \ldots, j$, is a block fragment,

Reminder R consists of the remaining cubes
that were not included in the block.

The partition π_h together with the GITE-polynomials D_i
form a block.

Decomposition





Two Algorithms of Decomposition

✓Two algorithms have been developed and studied: a “density” algorithm and a “dichotomy” algorithm

✓Both algorithms use one and the same general decomposition method

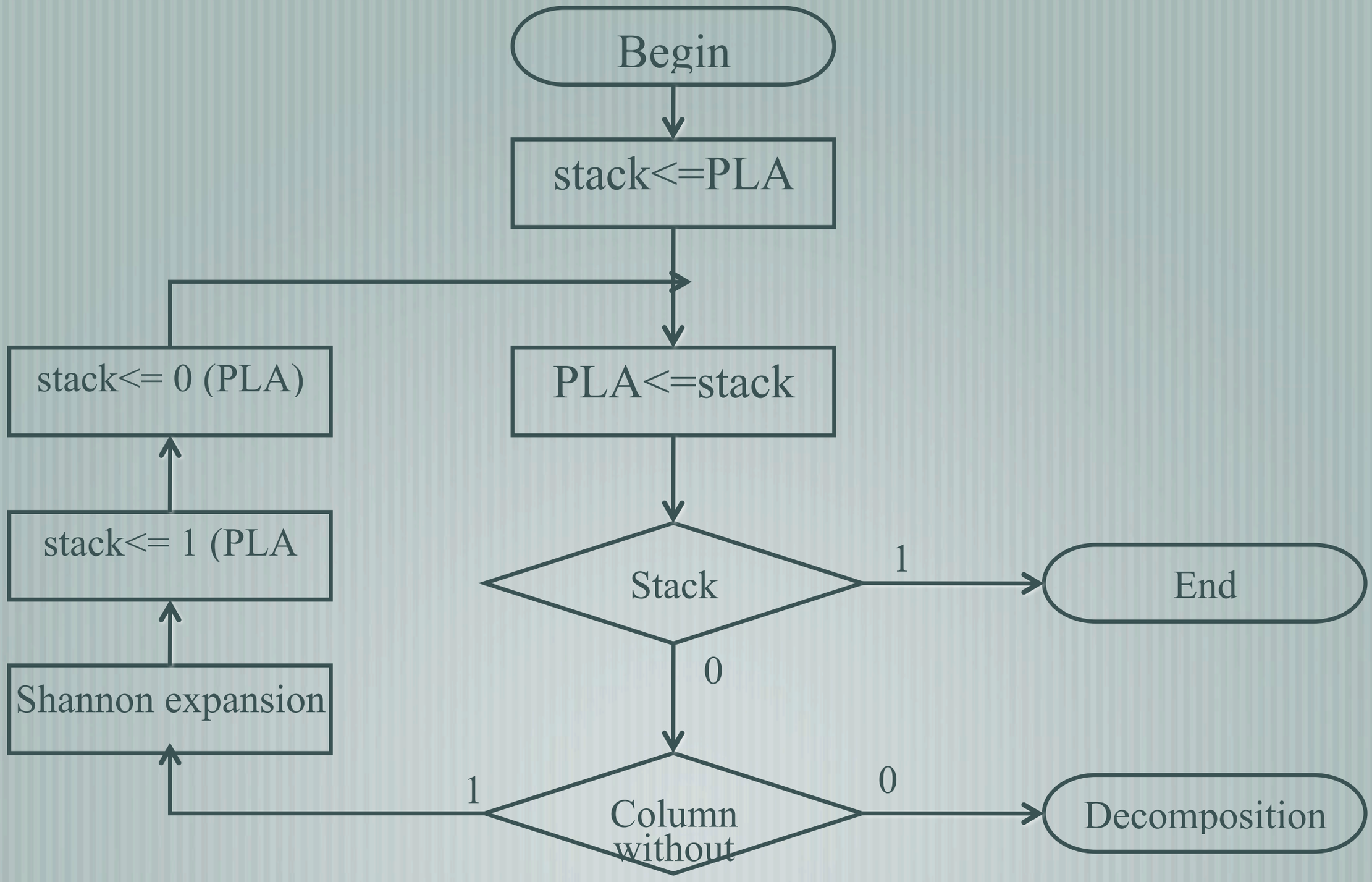
✓The general method is the partitioning of the set of cubes into a number of components. This partitioning is performed recursively

✓On each step of the recursive procedure, the corresponding component is partitioned into two subsets: a common header and a remainder

✓Each common header is implemented as a conventional MTBDD

✓The main concern of the general decomposition method is searching for optimal “common headers”, for obtaining optimal resulting MTBDD

Dichotomy Oriented Decomposition



Density

Block density corresponds to a number of literals in the block's cubes normalised by the maximal possible number of literal in this block. The success of the decomposition strongly depends on the density.

Experimental Results - Low Density

Title	|X|	D%	Nmon	Nnet	ratio
ALU1	12	18	982	25	0.02
B12	15	29	155	145	0.93
DK48	15	31	3428	58	0.02
DK27	9	34	79	22	0.28
CON1	7	37	16	15	0.94
ALU2	10	39	264	150	0.57
DUKE2	22	40	1435	326	0.23
ALU3	10	42	278	151	0.54
MISEX3C	14	43	10875	705	0.06
WIM	4	50	15	10	0.67
F51M	8	53	255	155	0.61
DK17	10	57	160	55	0.34
APLA	10	64	128	85	0.66
INC	7	79	39	35	0.9

Experimental Results - High Density

Title	|X|	D%	Nmon	Nnet	ratio
ADD6	12	52	504	731	1.45
RADD	8	57	90	143	1.59
CLIP	9	59	189	376	1.99
Z4	7	61	52	101	1.94
ROOT	8	65	72	134	1.86
SQR6	6	67	63	85	1.35
SQN	7	69	81	116	1.43
MLP4	8	73	240	345	1.44
SAO2	10	73	95	157	1.65
DIST	8	73	125	326	2.61
BW	5	80	25	58	2.32
RD53	5	90	15	53	3.53

Conclusions

- ✓Generalised ITE (GITE) operation is introduced
- ✓Multi output functions can be expressed by the GITE
- ✓GITE comprises Evolution Part (EvP) and Execution Part (ExP)
- ✓GITE algebra is a product of two algebras: Partition algebra of ExP and Boolean algebra of EvP
- ✓The problem of GITE decomposition is formulated
- ✓Mutual effect of the algebras are used as a base of the decomposition algorithm