Lecture 0_2.1

An introduction to Boolean Algebras for Circuit Design

Paolo PRINETTO

Politecnico di Torino (Italy)

University of Illinois at Chicago, IL (USA)

Paolo.Prinetto@polito.it

prinetto@uic.edu

www.testgroup.polito.it

www.comitato-girotondo.org

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Goal

– This lecture presents a global overviews of Boolean Algebras, focusing on issues related to their usage in designing logical circuits.

Prerequisites

– **None**

Rel. 22.12.2017

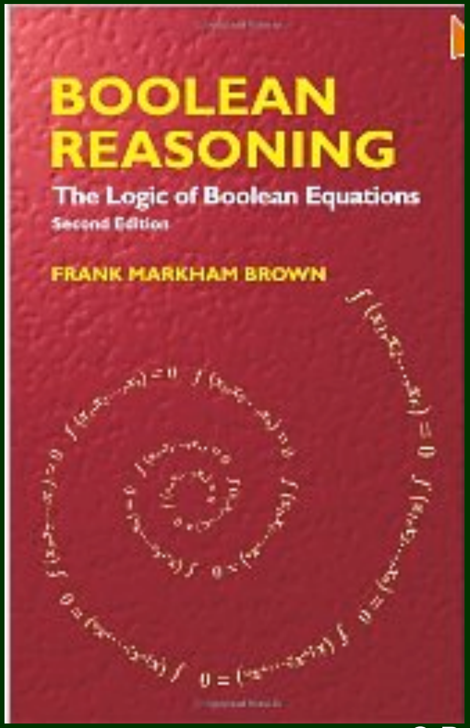
Homework

– Solve the proposed Exercises

Further readings

– **Students interested in a deeper knowledge of the arguments covered in this lecture can refer, for instance, to:**

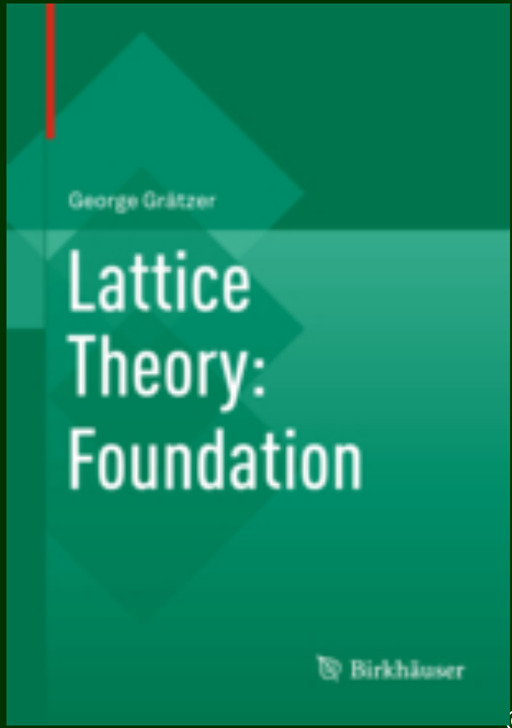
  - **F.M. Brown:**
    "*Boolean reasoning: the logic of boolean equations,*"
    2nd edition, 2012



Further readings

– Details on Lattices can be found, for instance, in:
  - G. Gratzer:
    "*Lattice Theory: Foundation*"
    Springer, 2011

    http://www.springer.com/birkhauser/mathematics/book/978-3-0348-0017-4



Outline

- Boolean Algebras Definitions
- Examples of Boolean Algebras
- Geometric interpretation of Boolean Algebras
- Values taken by functions
- How representing functions?
- Boolean Algebras properties

Outline

- – Boolean Algebras Definitions
- – Examples of Boolean Algebras
- – Geometric interpretation of Boolean Algebras
- – Values taken by functions
- – How representing functions?
- – Boolean Algebras properties

Boolean Algebras Definitions

- Boolean Algebras are defined, in the literature, in many different ways:
  - definition by *lattices*
  - definition by *properties*
  - definition by *postulates* [Huntington]

Boolean Algebras Definitions

- Boolean Algebras are defined, in the literature, in many different ways:
  - definition by *lattices*
  - definition by *properties*
  - definition by *postulates* [Huntington]

Boolean Algebras
definition by postulates

A Boolean Algebra is an algebraic system

$$(B , + , \cdot , 0 , 1)$$

where:

- B is a set
- + and · are binary operations in B
- 0 and 1 are distinct elements in B

satisfying the following postulates:

A1: closure

$\forall$ a, b $\in$ B:

- a + b $\in$ B
- a · b $\in$ B

A2 : commutative

$\forall$ a, b $\in$ B:

– a + b = b + a

– a · b = b · a

A3: distributive

$\forall$ a, b, c $\in$ B:

– $a \cdot (b + c) = a \cdot b + a \cdot c$

– $a + b \cdot c = (a + b) \cdot (a + c)$

A4: identities

$\exists\ 0 \in B \mid \forall a \in B,\ a + 0 = a$

$\exists\ 1 \in B \mid \forall a \in B,\ a \cdot 1 = a$

A5: existence of the complement

$\forall\, a \in B,\ \exists\, a' \in B \ |$

- $a + a' = 1$
- $a \cdot a' = 0.$

Some definitions

- The elements of the carrier set B = {0,1} are called *constants*
- All the symbols that get values $\in$ B are called *variables* (hereinafter they will be referred to as $x_1, x_2, \ldots, x_n$)
- A *letter* is a constant or a variable
- A *literal* is a letter or its complement

Outline

– Boolean Algebras Definitions
– **Examples of Boolean Algebras**
– Geometric interpretation of Boolean Algebras
– Values taken by functions
– How representing functions?
– Boolean Algebras properties

Examples of Boolean Algebras

- Let us consider some examples of Boolean Algebras:
  - *the algebra of classes*
  - *propositional algebra*
  - *arithmetic Boolean Algebras*
  - *binary Boolean Algebra*
  - *quaternary Boolean Algebra*

The algebra of classes

- Suppose that every set of interest is a subset of a fixed nonempty set S.
- We call
  - S a universal set
  - its subsets the *classes* of S.
- The *algebra of classes* consists of the set 2^S (the set of subsets of S) together with two operations on 2^S, namely *union* and *intersection*.

Rel. 22.12.2017

The algebra of classes (2)

This algebra satisfies the postulates for a Boolean Algebra, provided the substitutions:

B	$\leftrightarrow$	2^S
+	$\leftrightarrow$	$\cup$
·	$\leftrightarrow$	$\cap$
0	$\leftrightarrow$	$\varnothing$
1	$\leftrightarrow$	**S**

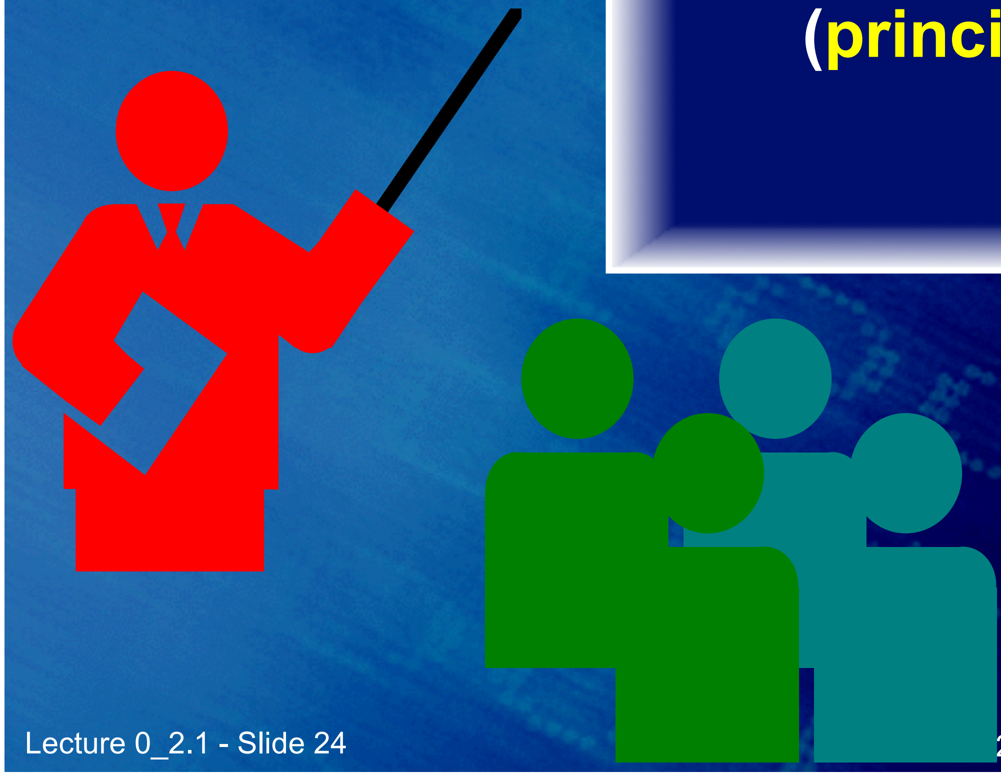
Thus, the algebraic system

$$(2^S, \cup, \cap, \varnothing, S)$$

ia a Boolean Algebra.

Propositions

A **proposition** is a formula which is necessarily TRUE or FALSE (**principle of the excluded third**), but cannot be both (**principle of no contradiction**)



Russell's paradox

As a consequence, Bertrand Russell's paradox:

"*this sentence is false*"

is not a proposition, since if it is assumed to be TRUE its content implies that is is FALSE, and vice-versa.

Propositional algebra

Let:

- **P** a set of propositional functions
- **F** the formula which is always false (contradiction)
- **T** the formula which is always true (tautology)
- $\vee$ the disjunction (or)
- $\wedge$ the conjunction (and)
- $\neg$ the negation (not)

then:

Propositional algebra *(2)*

The system

$$(P, \vee , \wedge , F , T)$$

is a Boolean Algebra:

B $\leftrightarrow$ P

\+ $\leftrightarrow$ $\vee$

· $\leftrightarrow$ $\wedge$

0 $\leftrightarrow$ F

1 $\leftrightarrow$ T

Arithmetic Boolean Algebra

Let:

- *n* be the result of a product of the elements of a set of prime numbers
- *D* the set of all the dividers of n
- *lcm* the operation that evaluates the *lowest common multiple*
- *GCD* the operation that evaluates the *Greatest Common Divisor*

Arithmetic Boolean Algebra (2)

The algebraic system:

$$(D, lcm, GCD, 1, n)$$

is a Boolean Algebra:

- B $\leftrightarrow$ D
- + $\leftrightarrow$ lcm
- · $\leftrightarrow$ GCD
- 0 $\leftrightarrow$ 1
- 1 $\leftrightarrow$ n

Binary Boolean Algebra

The system

$$(\{0,1\} , + , \cdot , 0 , 1)$$

is a Boolean Algebra, provided that the two operations + and · be defined as follows:

+	0	1
0	0	1
1	1	1

·	0	1
0	0	0
1	0	1

Quaternary Boolean Algebra

The system

$$(\{a,b,0,1\} , + , \cdot , 0 , 1)$$

is a Boolean Algebra provided that the two operations + and · be defined as follows:

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

·	0	a	b	1
0	0	0	0	0
a	0	a	0	a
b	0	0	b	b
1	0	a	b	1

Outline

- Boolean Algebras Definitions
- Examples of Boolean Algebras
- **Geometric interpretation of Boolean Algebras**
- Values taken by functions
- How representing functions?
- Boolean Algebras properties

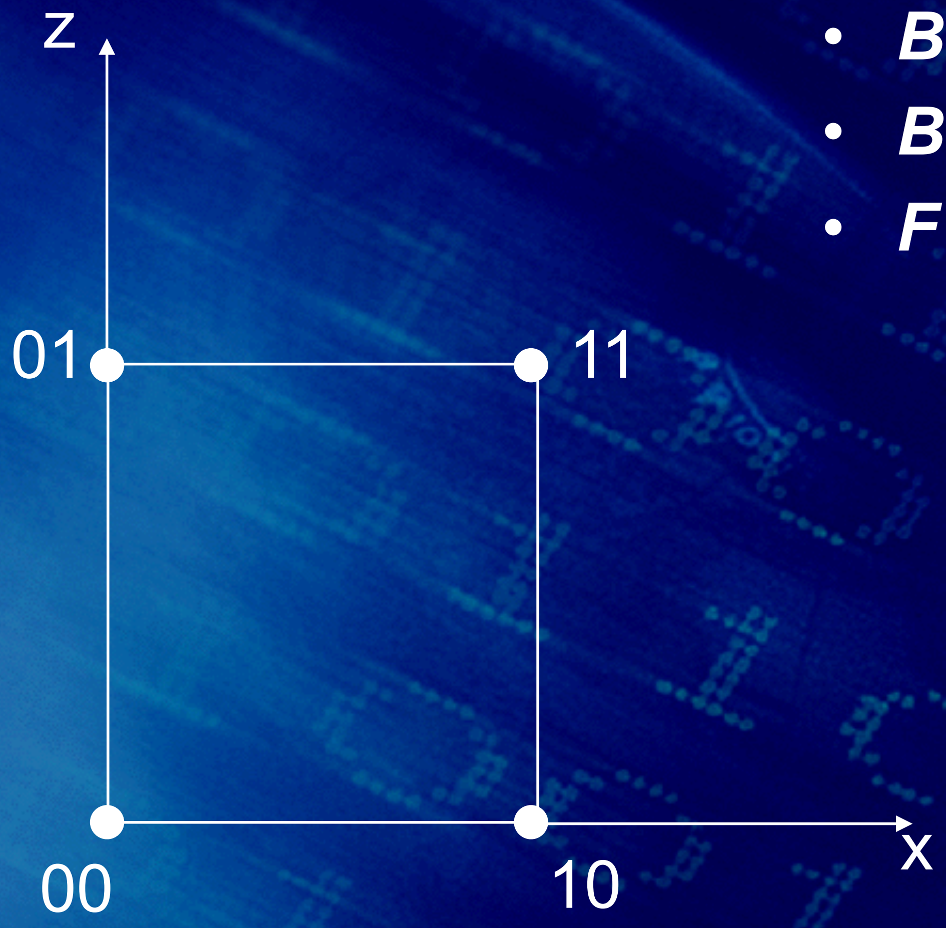
Geometric interpretation

The carrier B^n of a Boolean Algebra can be seen as an *n-dimensional space*, where each generic element $v \in B^n$ (usually called a *vertex*), is represented by a vector of n coordinates, each $\in B$

1-input functions

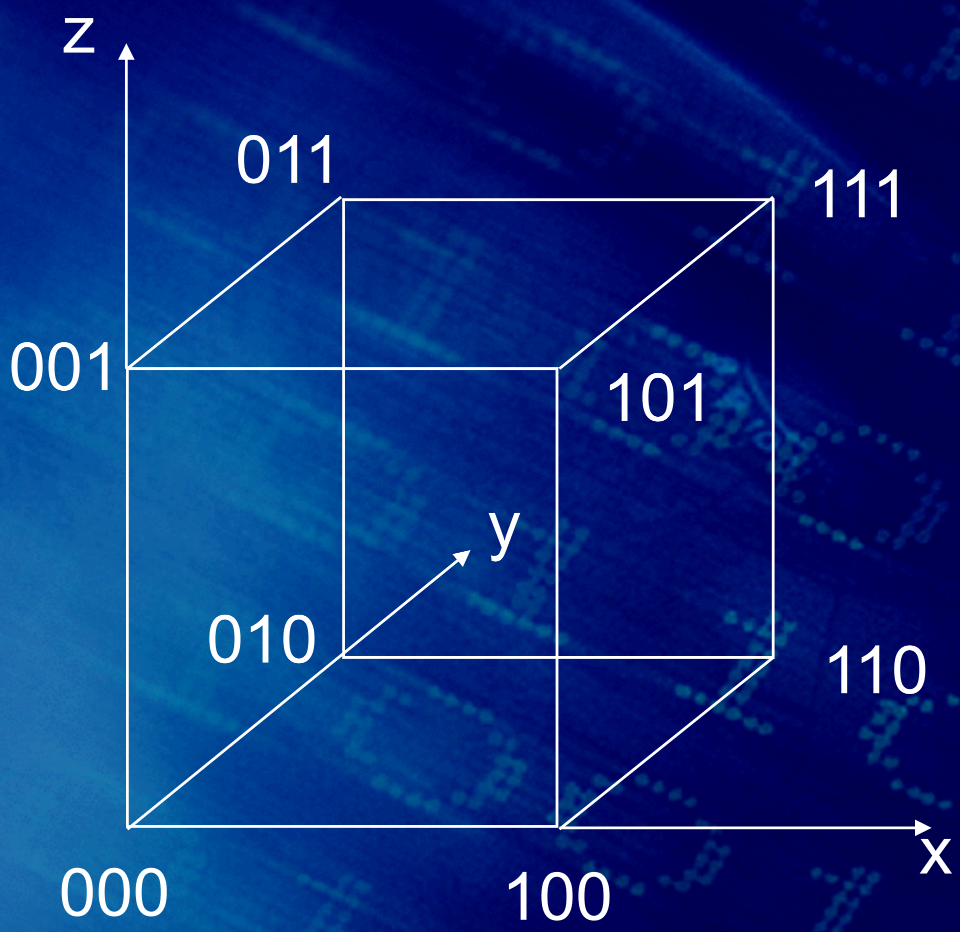
- ***B* = { 0, 1}**
- ***B***
- ***F* = F (x)**

2-input functions



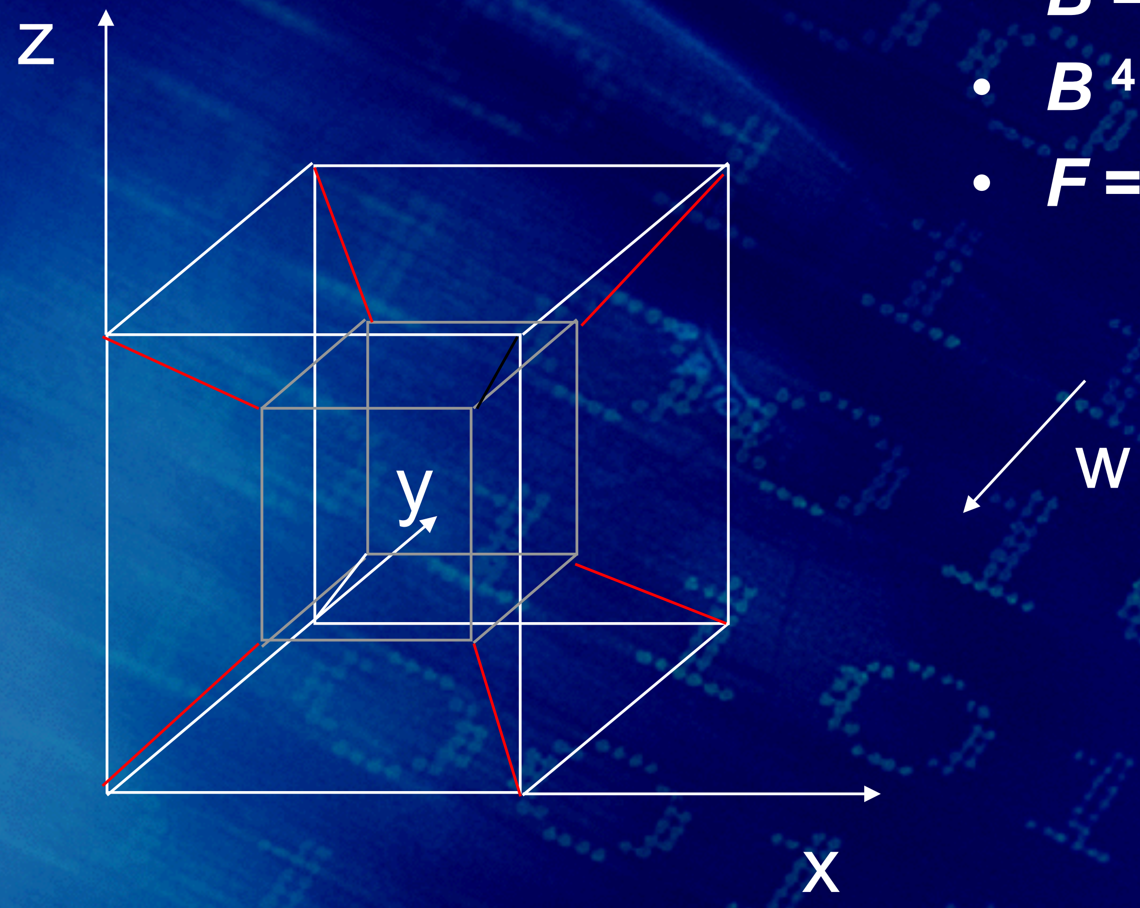
- *B* = { 0, 1}
- B^2
- *F* = F (x, z)

3-input functions



- B = { 0, 1}
- B^3
- F = F (x, y, z)

4-input functions



- ***B*** = { 0, 1 }
- ***B*** ^{4}
- ***F*** = F (x, y, z, w)

Outline

- Boolean Algebras Definitions
- Examples of Boolean Algebras
- Geometric interpretation of Boolean Algebras
- **Values taken by functions**
- How representing functions?
- Boolean Algebras properties

Values taken by functions

– In general, in each vertex V, a function may:

. *get the value 0* $\Rightarrow$ V $\in$ *off-set*

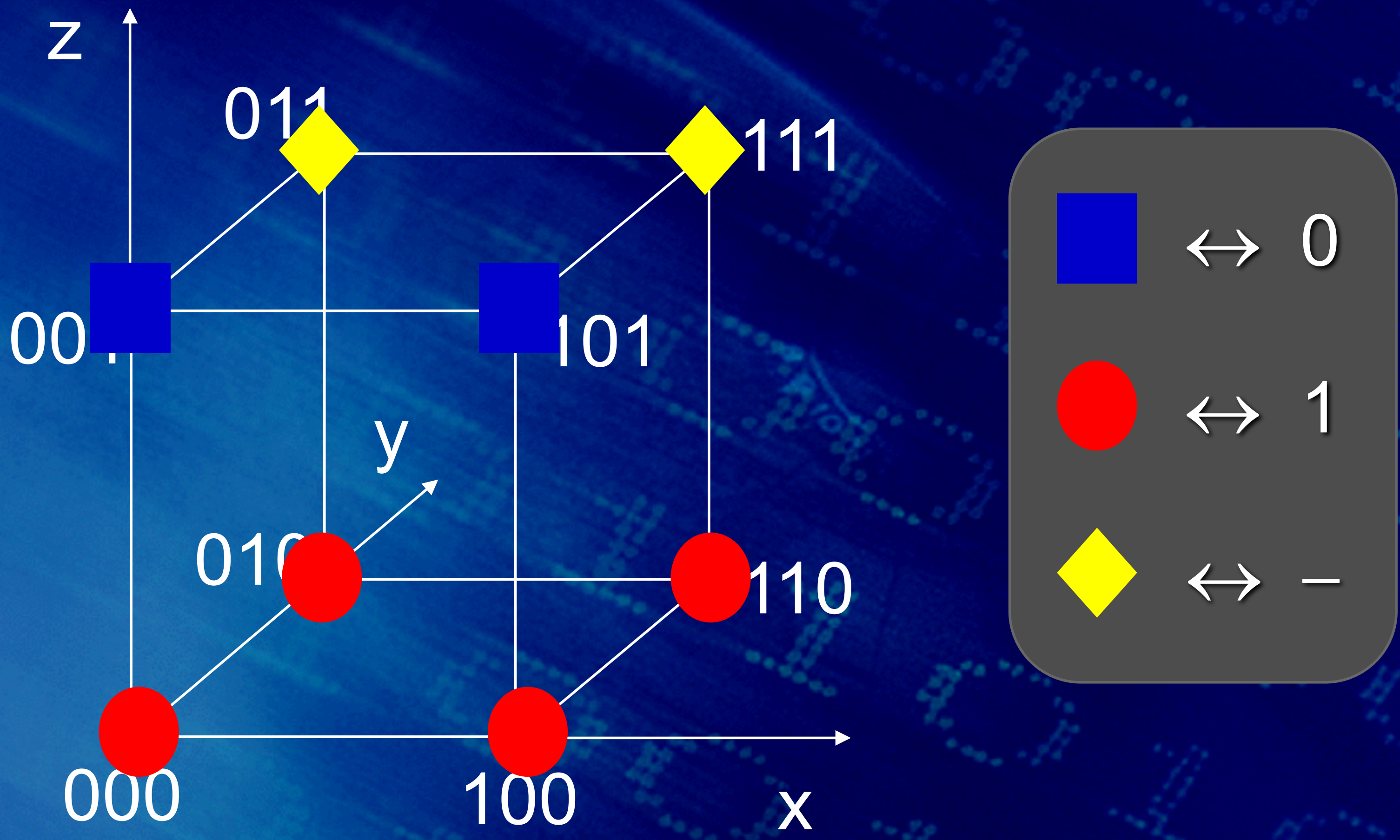
. *get the value 1* $\Rightarrow$ V $\in$ *on-set*

. *be not specified* $\Rightarrow$ V $\in$ *don't-care set*

Values taken by functions

- In general, in each vertex V, a function may:
  - . *get the value 0* $\Rightarrow$ V $\in$ *off-set*
  - . *get the value 1* $\Rightarrow$ V $\in$ *on-set*
  - . *be not specified* $\Rightarrow$ V $\in$ *don't-care set*

Typically used to represent combinations of input variables not relevant in a given design

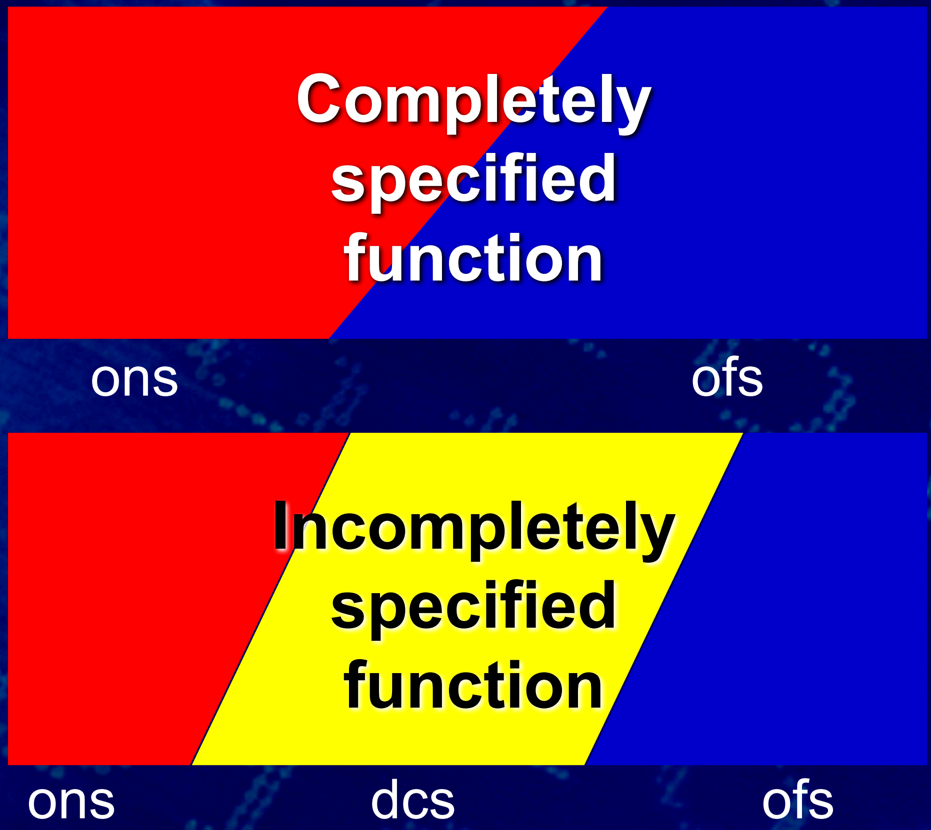


z
011
111
001
101
y
010
110
000
100
x
↔ 0
↔ 1
↔ –

Completely vs. incompletely specified functions

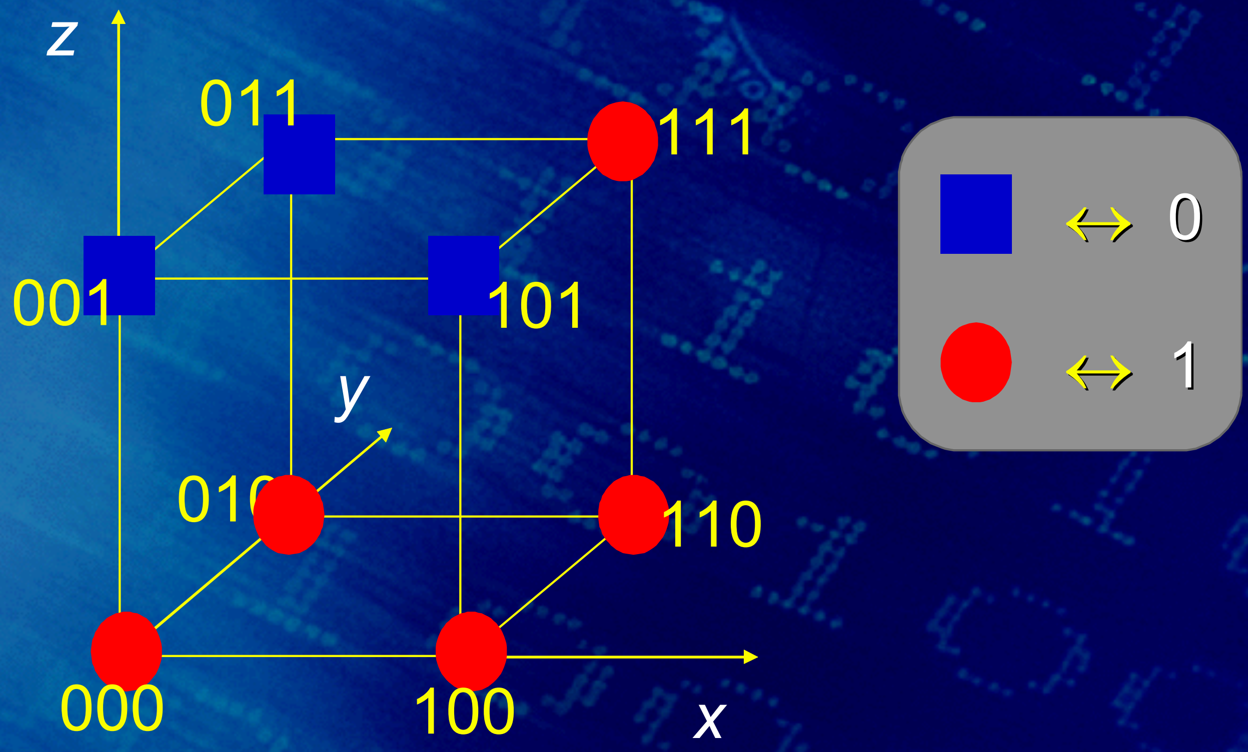
- A function f is ***completely specified*** iff dcs (f) = ∅

- f is ***incompletely specified*** otherwise



Note

- **For sake of simplicity, without loosing generality, in the sequel of the lecture we shall use, as a case study, the following completely specified 3-input function *F = F (x, y, z)* :**



Outline

- – Boolean Algebras Definitions
- – Examples of Boolean Algebras
- – Geometric interpretation of Boolean Algebras
- – Values taken by functions
- – **How representing functions?**
- – Boolean Algebras properties

How representing functions?

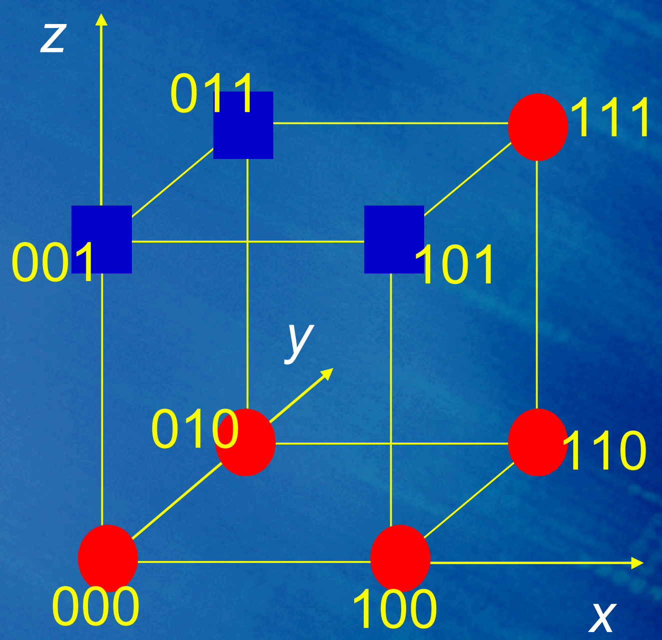
- We need a way "*to represent*" or "*to express*" the value that the function F gets in each vertex, i.e., in each point of its domain.

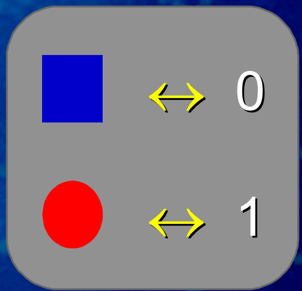
An exhaustive approach

- We could explicitly represent the values assumed by the function for each of the possible combinations of the input variables ⇒ 2^n entries !!!
- Two alternatives:
  - Tabular representation: ***Truth Table***
  - Matrix representation: ***Karnaugh maps***

Truth table

- It specifies the value the function gets for each input combination:





x y z	F
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	1

Usage

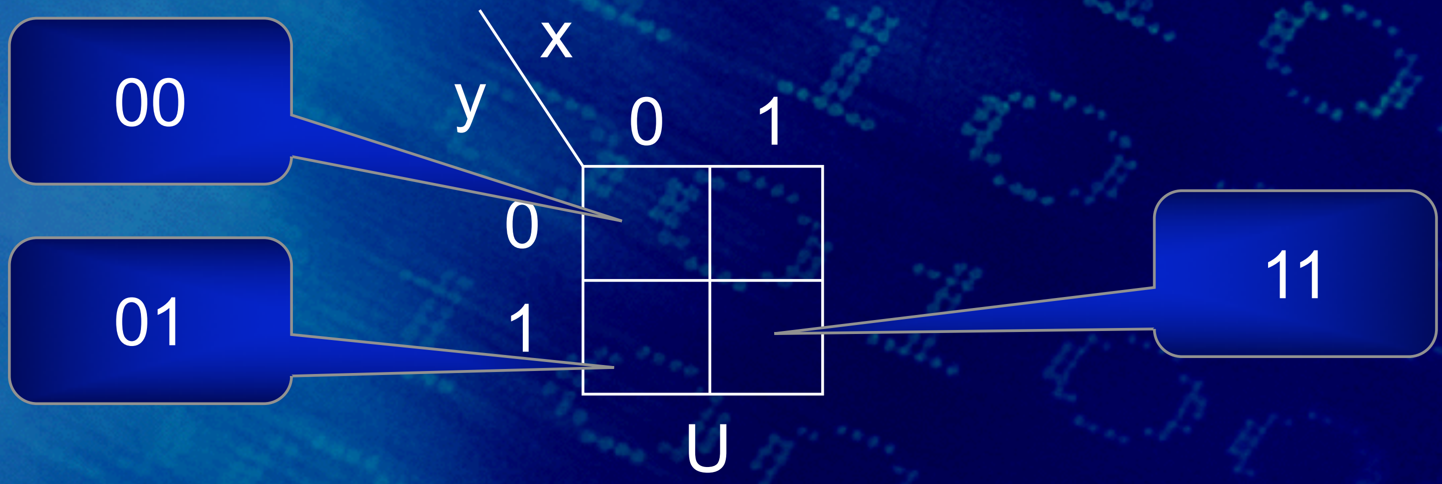
- When dealing with few inputs, they can be used to prove theorems
- Sometimes helpful in manual design, as preliminary step to generate the equivalent Karnaugh maps

Karnaugh maps

Karnaugh maps (***K-map*** **for short) were first introduced by Maurice Karnaugh in 1953.**

An n input variable boolean function is represented by a matrix of 2^n cells.

Each cell specifies the value of the function when its input variables get the values of the corresponding row and column.



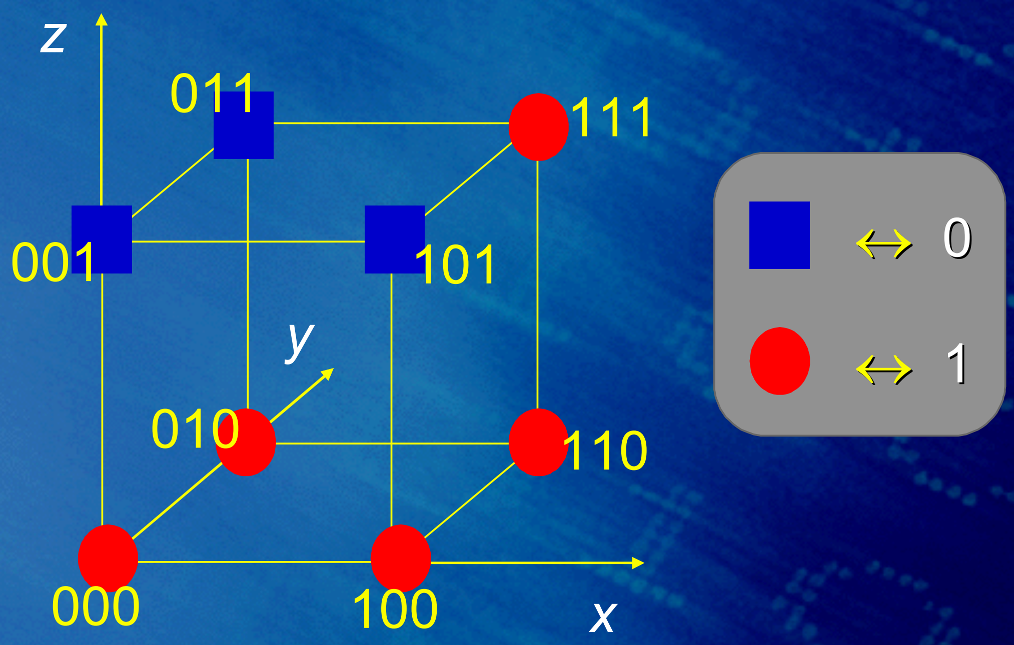
Karnaugh maps

x y z	F
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	1

z \ x y	00	01	11	10
0	1	1	1	1
1	0	0	1	0

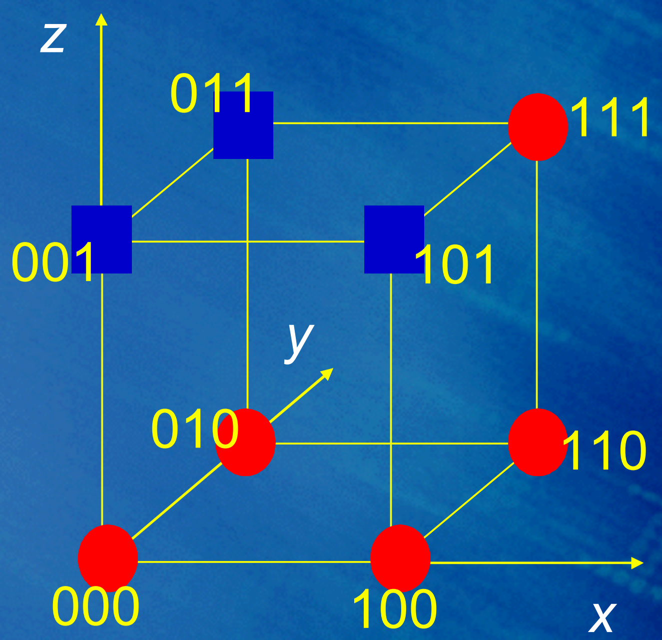
An intuitive approach

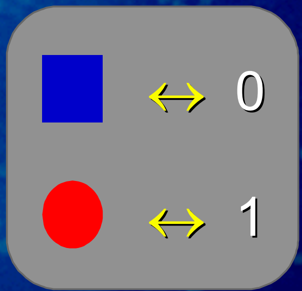
- We could, for instance, deciding of expressing only the vertices in which the function gets the value 1:

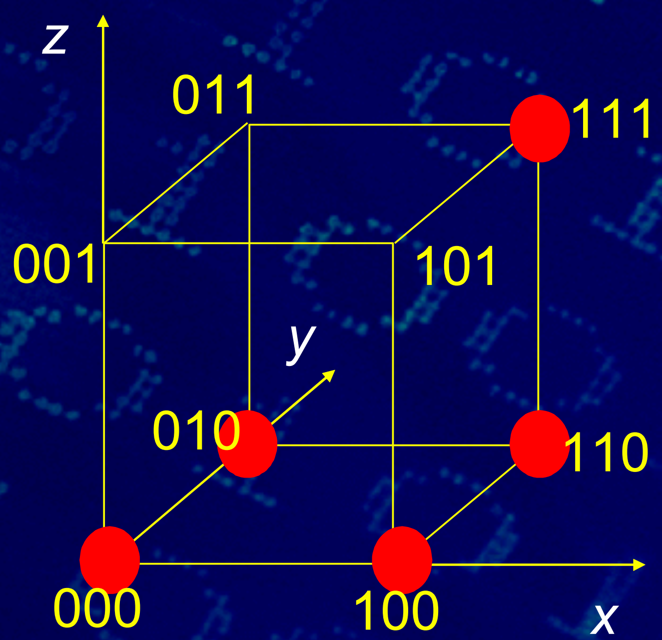


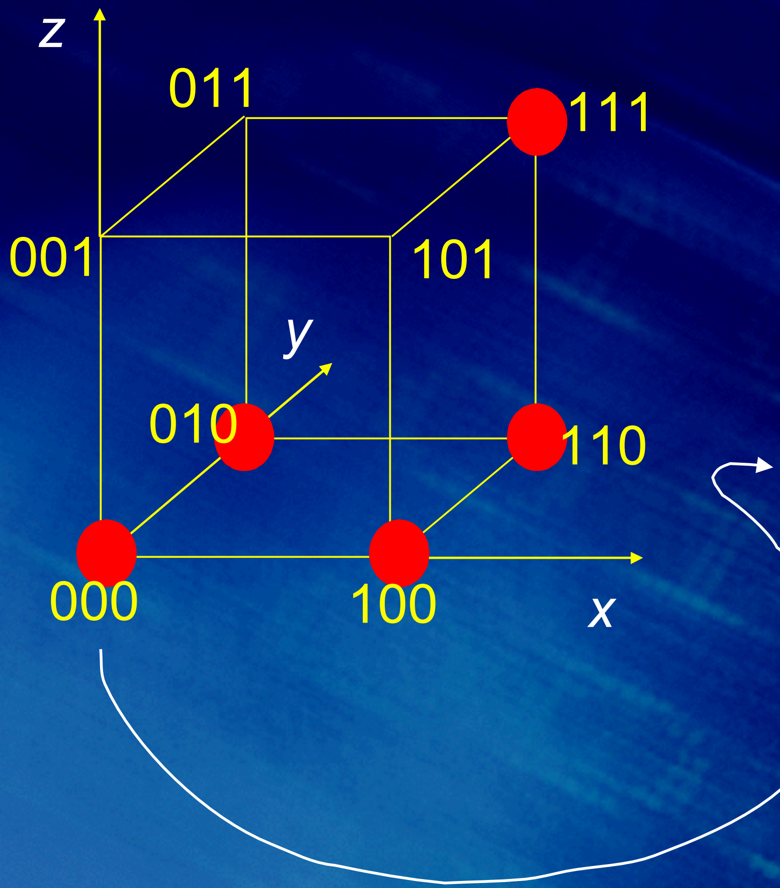
An intuitive approach

We could, for instance, deciding of expressing only the vertices in which the function gets the value 1:





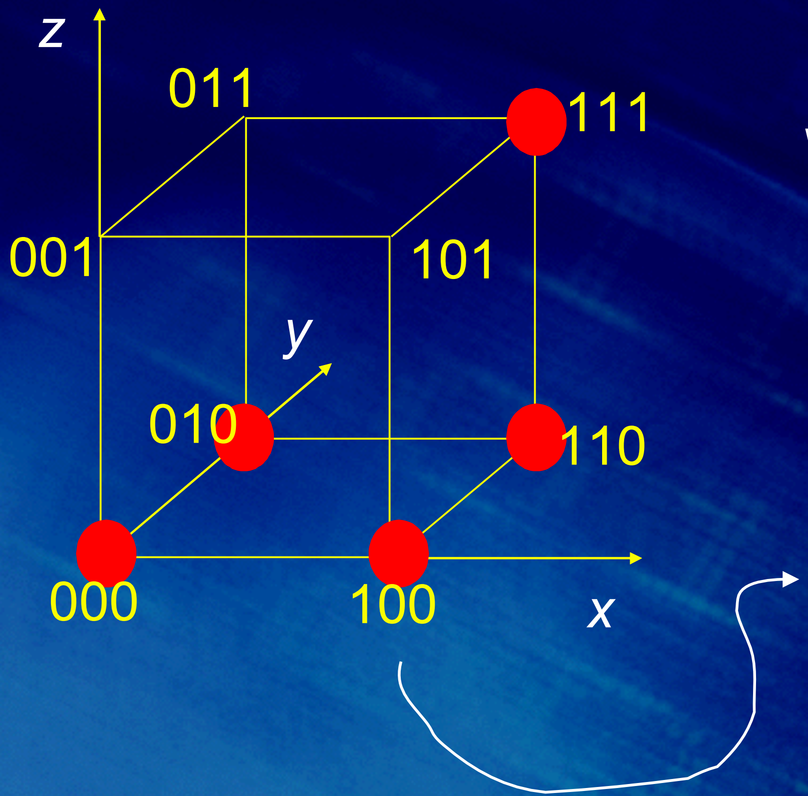




We could write:

F = 1 when

((x=0) and (y=0) and (z=0))



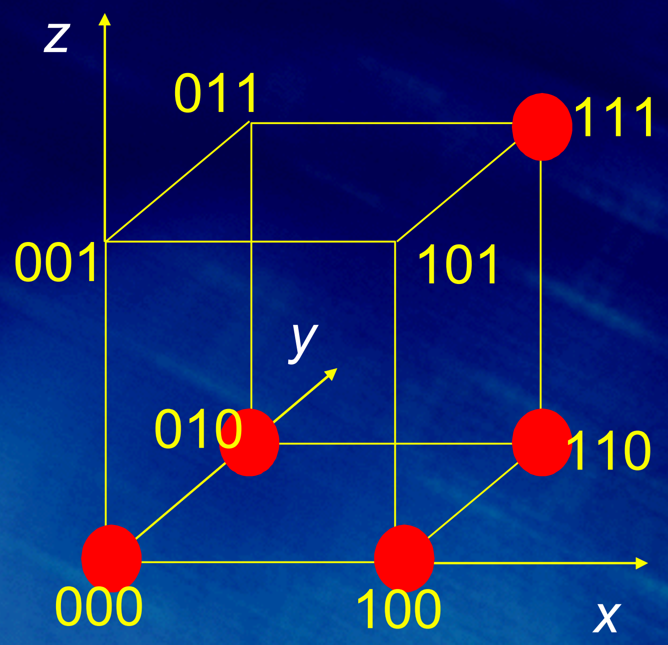
We could write:

F = 1 when

((x=0) and (y=0) and (z=0))

or

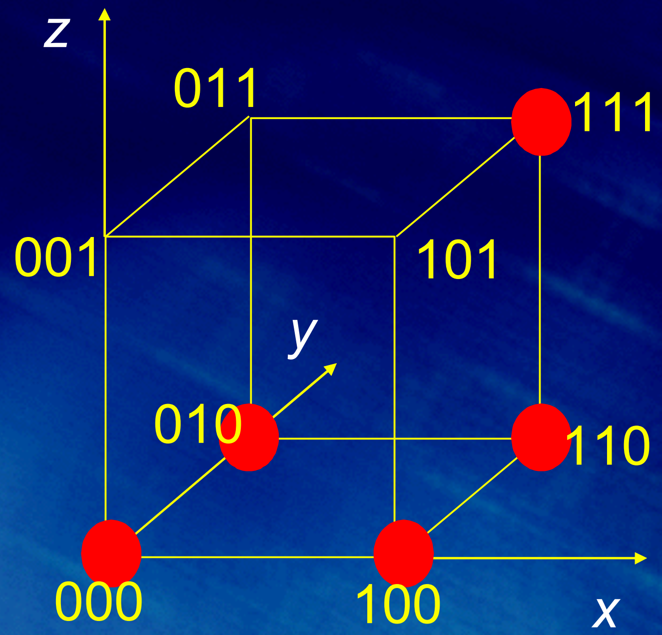
((x=1) and (y=0) and (z=0))

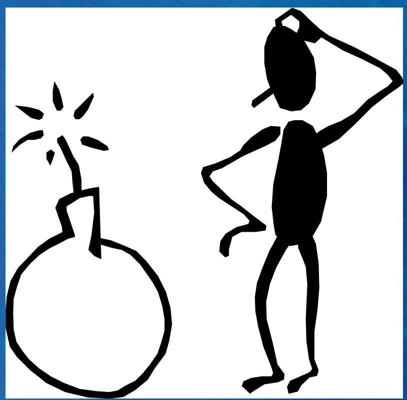


We could write:

F = 1 when

((x=0) and (y=0) and (z=0))
or
((x=1) and (y=0) and (z=0))
or
((x=1) and (y=1) and (z=0))
or
((x=0) and (y=1) and (z=0))
or
((x=1) and (y=1) and (z=1)).





We could write:

F = 1 when

((x=0) and (y=0) and (z=0))
or
((x=1) and (y=0) and (z=0))
or
((x=1) and (y=1) and (z=0))
or
((x=0) and (y=1) and (z=0))
or
((x=1) and (y=1) and (z=1)).

Let's try to simplify

(x=1)	→	x	
(x=0)	→	x'	
and	→	·	→
or	→	+	

Let's try to simplify

(x=1)	→	x	
(x=0)	→	x'	
and	→	·	→
or	→	+	

F = 1 when

((x=0) and (y=0) and (z=0)) or

((x=1) and (y=0) and (z=0)) or

((x=1) and (y=1) and (z=0)) or

((x=0) and (y=1) and (z=0)) or

((x=1) and (y=1) and (z=1))

Let's try to simplify

(x=1)	→	x	
(x=0)	→	x'	
and	→	·	→
or	→	+	

F = 1 when

((x=0) and (y=0) and (z=0)) or

((x=1) and (y=0) and (z=0)) or

((x=1) and (y=1) and (z=0)) or

((x=0) and (y=1) and (z=0)) or

((x=1) and (y=1) and (z=1))

→

F =

x'y'z' +

x y'z' +

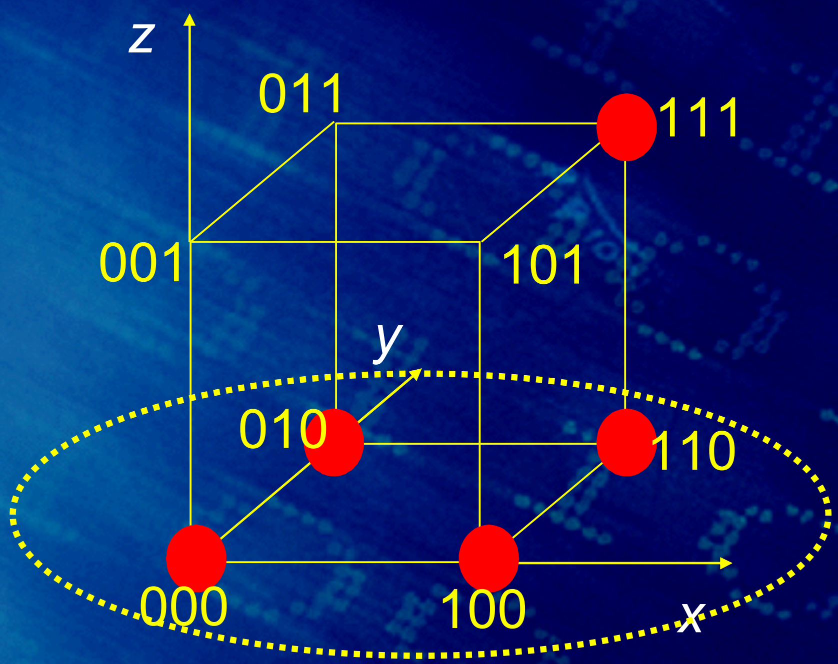
x y z' +

x'y z' +

x y z

A further step

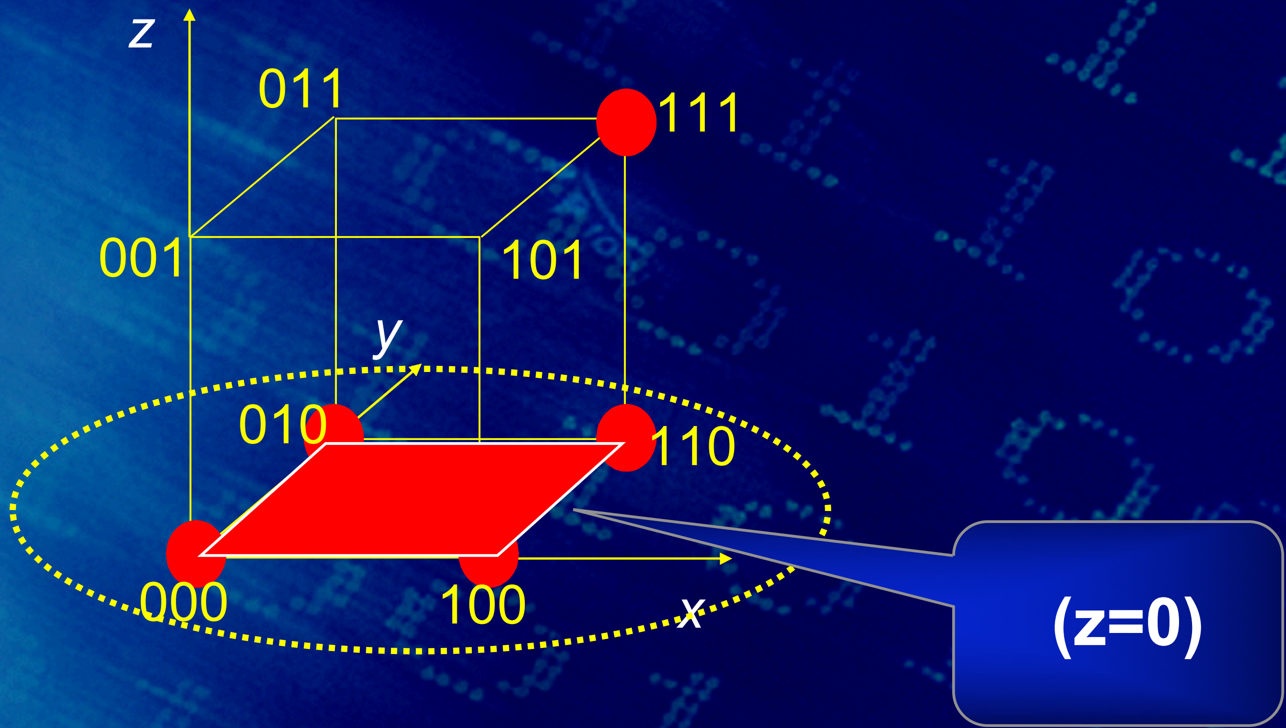
Instead of representing separately the 4 vertices



we could represent the whole "face"

A further step

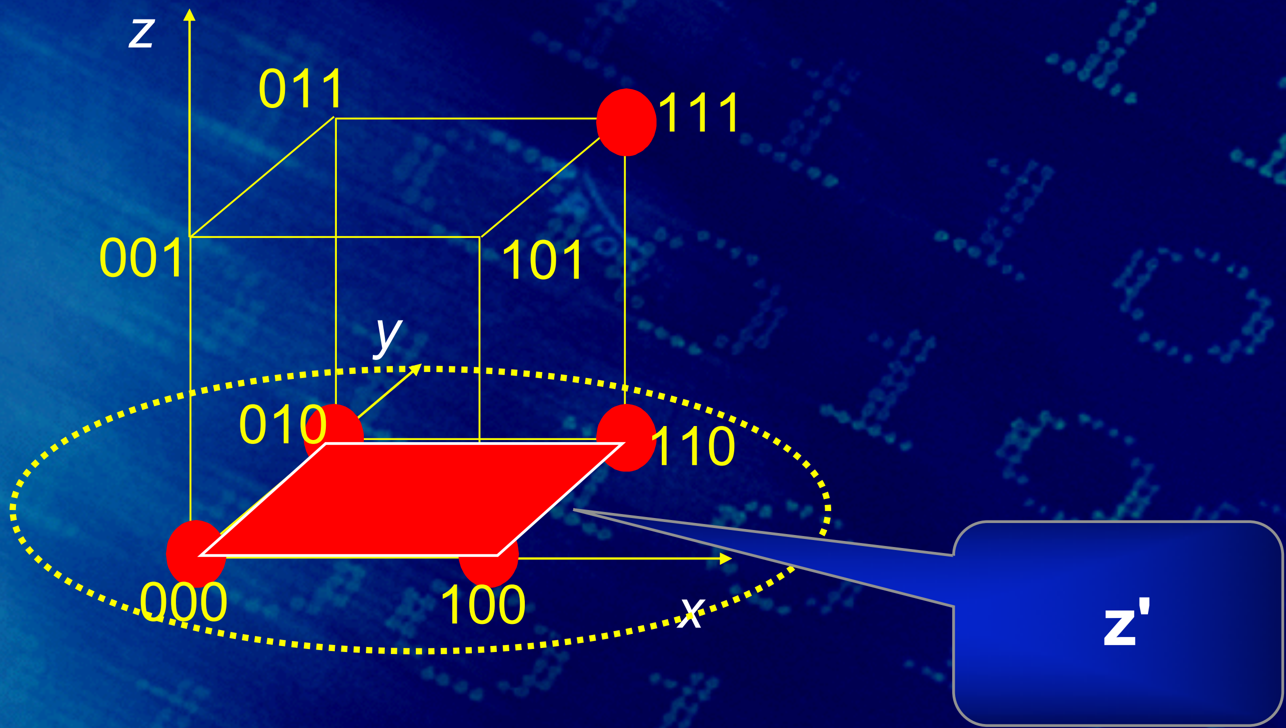
Instead of representing separately the 4 vertices



we could represent the whole "face"

A further step

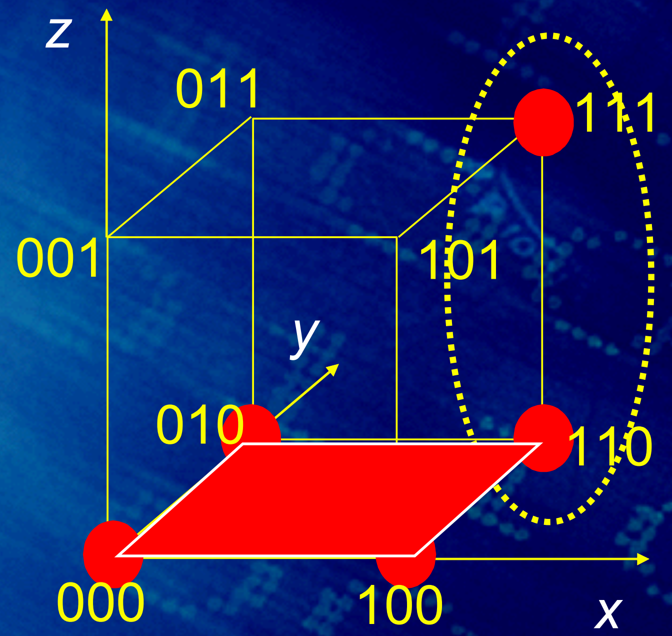
Instead of representing separately the 4 vertices



we could represent the whole "face"

A further step (3)

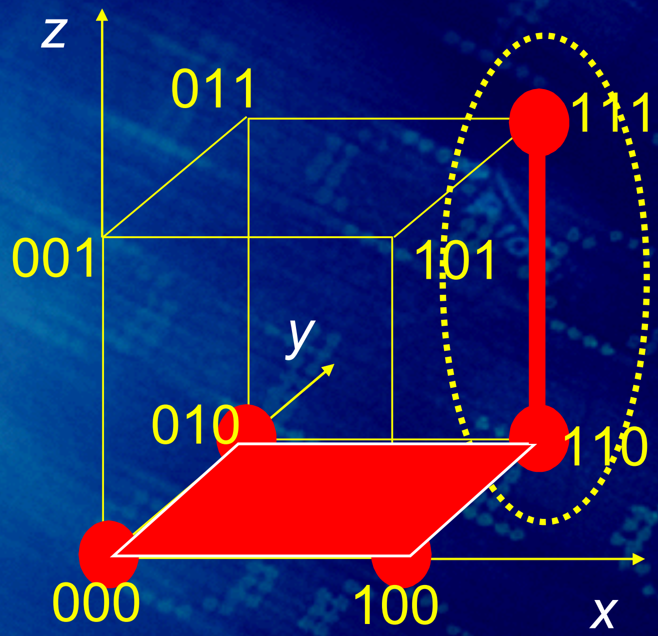
... and instead of representing separately the 2 vertices



we could represent the related "edge"

A further step *(3)*

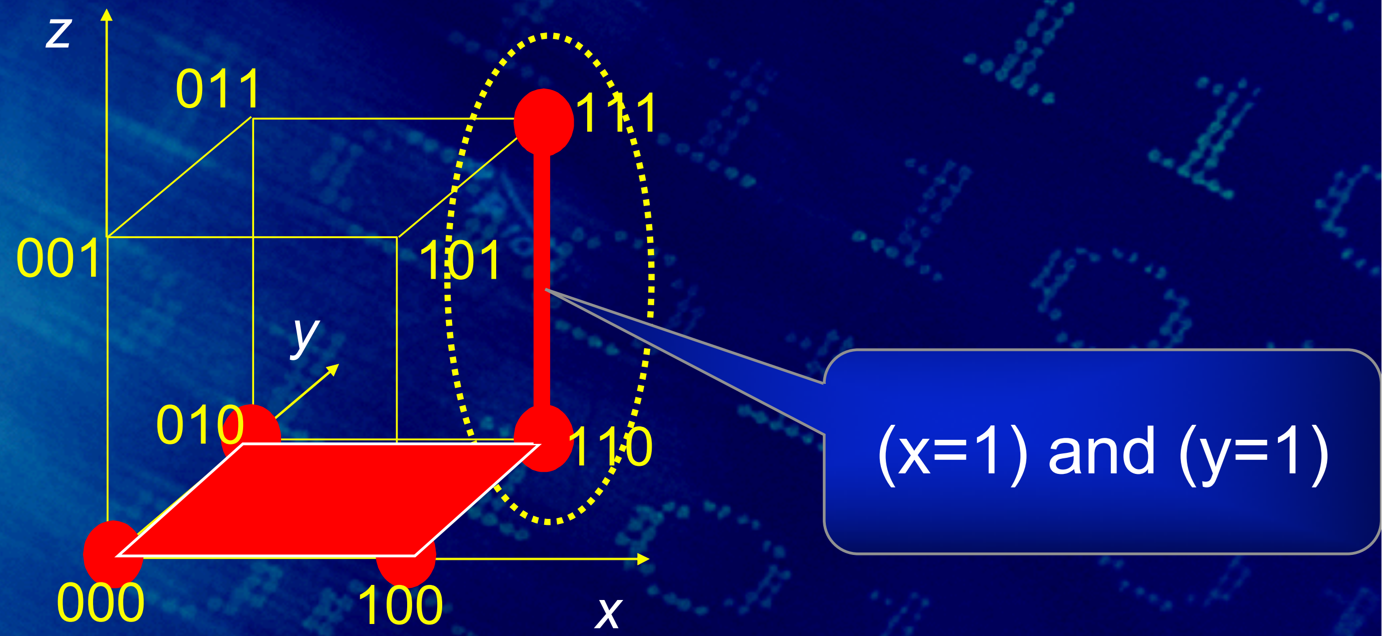
... and instead of representing separately the 2 vertices



we could represent the related “edge”

A further step (4)

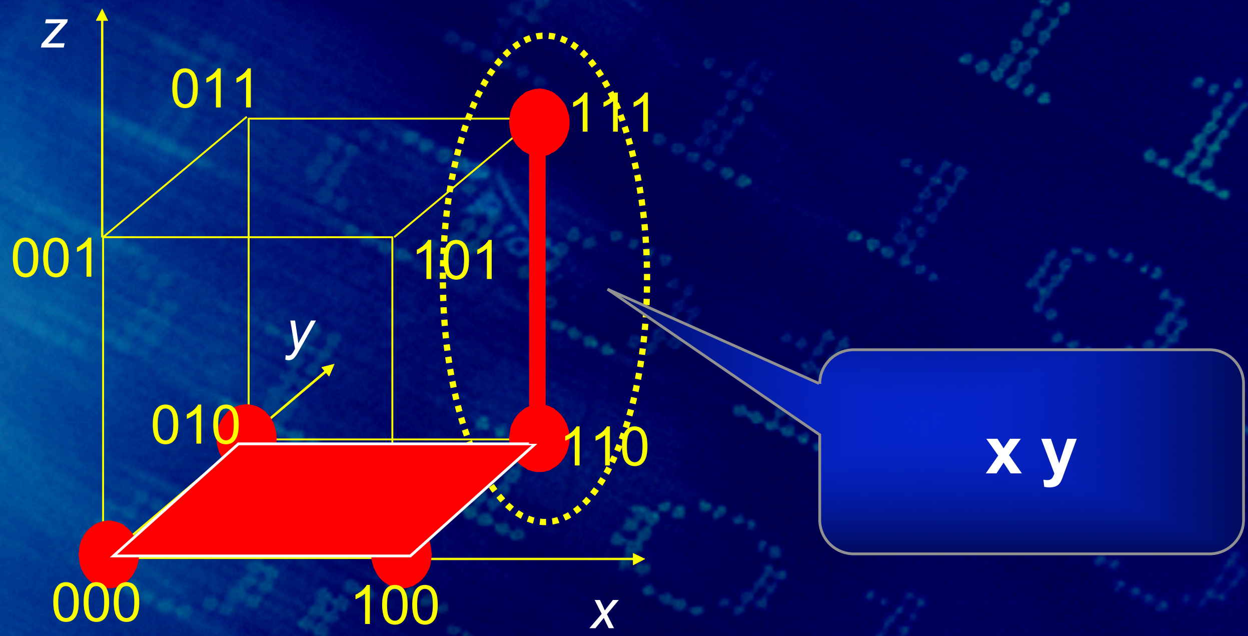
... and instead of representing separately the 2 vertices



we could represent the related "edge"

A further step (5)

... and instead of representing separately the 2 vertices



we could represent the related "edge"

Thus obtaining

F =

x'y'z' +

x y'z' +

x y z' +

x'y z' +

x y z

→ F = z' + x y

Let’s formalized the concept

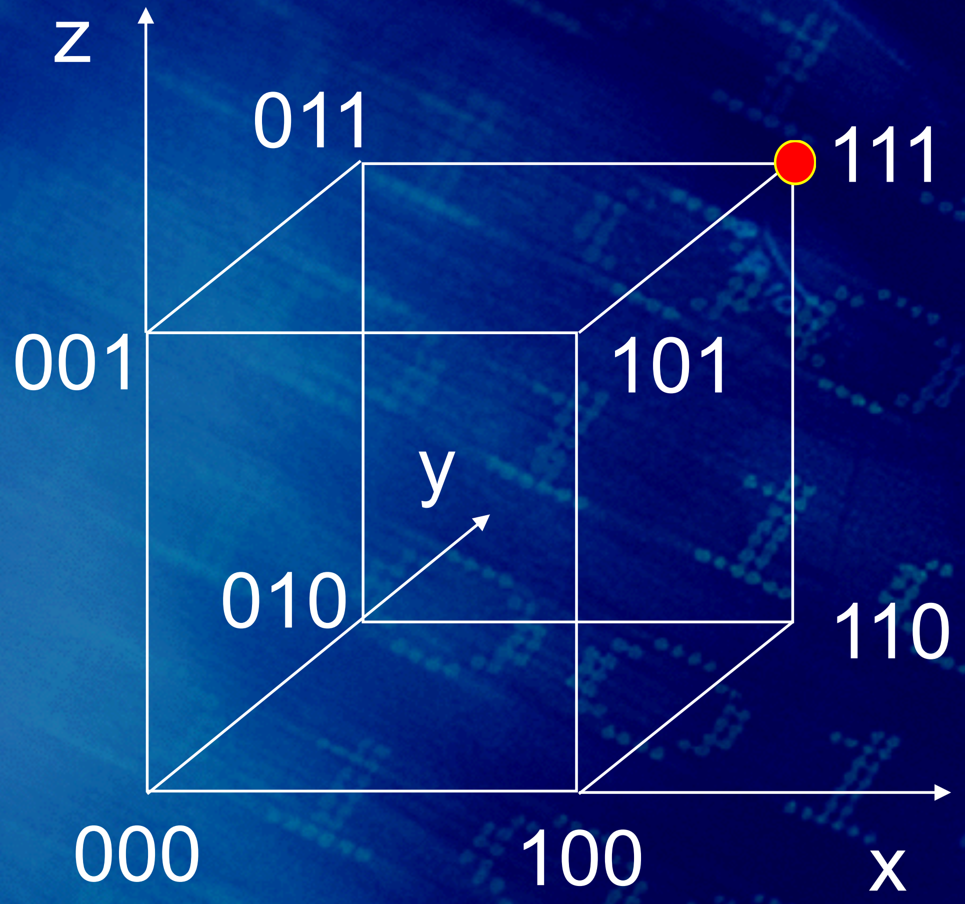
K-cube

A k-dimensional sub-space
(or k-cube)

$$S_k \subseteq B^n$$

is a set of 2^k vertices, in which n-k input variables get a same constant value

0-cube = vertex



***B* = { 0, 1 }**

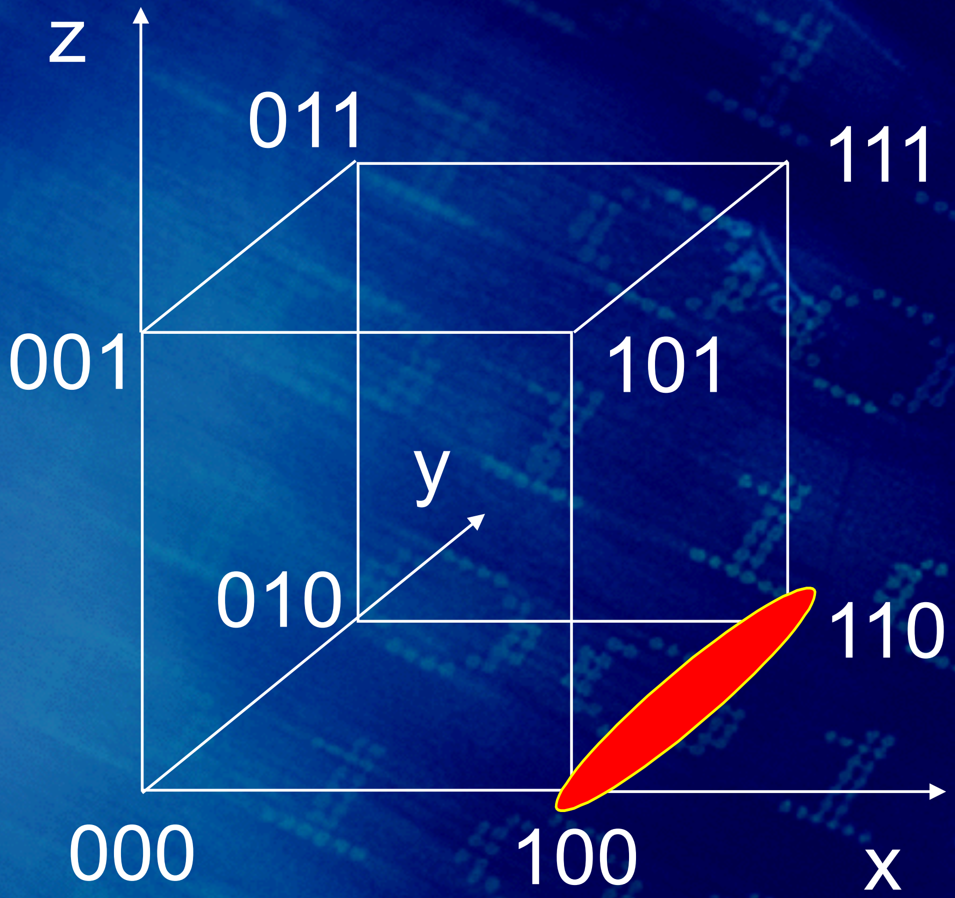
B^3

n = 3

k = 0

n-k = 3

1-cube = edge



B **= { 0, 1 }**

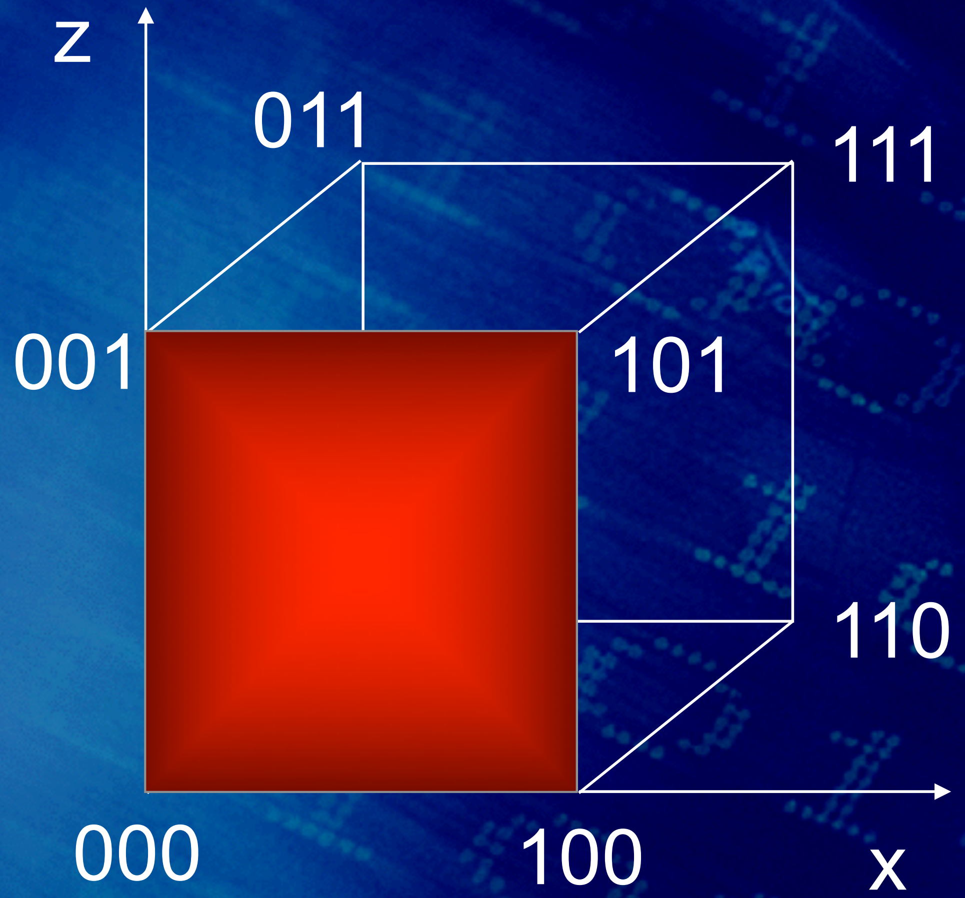
B^3

n = 3

k = 1

n-k = 2

2-cube = face



$\boldsymbol{B} = \{ 0, 1 \}$

$\boldsymbol{B}^3$

n = 3

k = 2

n-k = 1

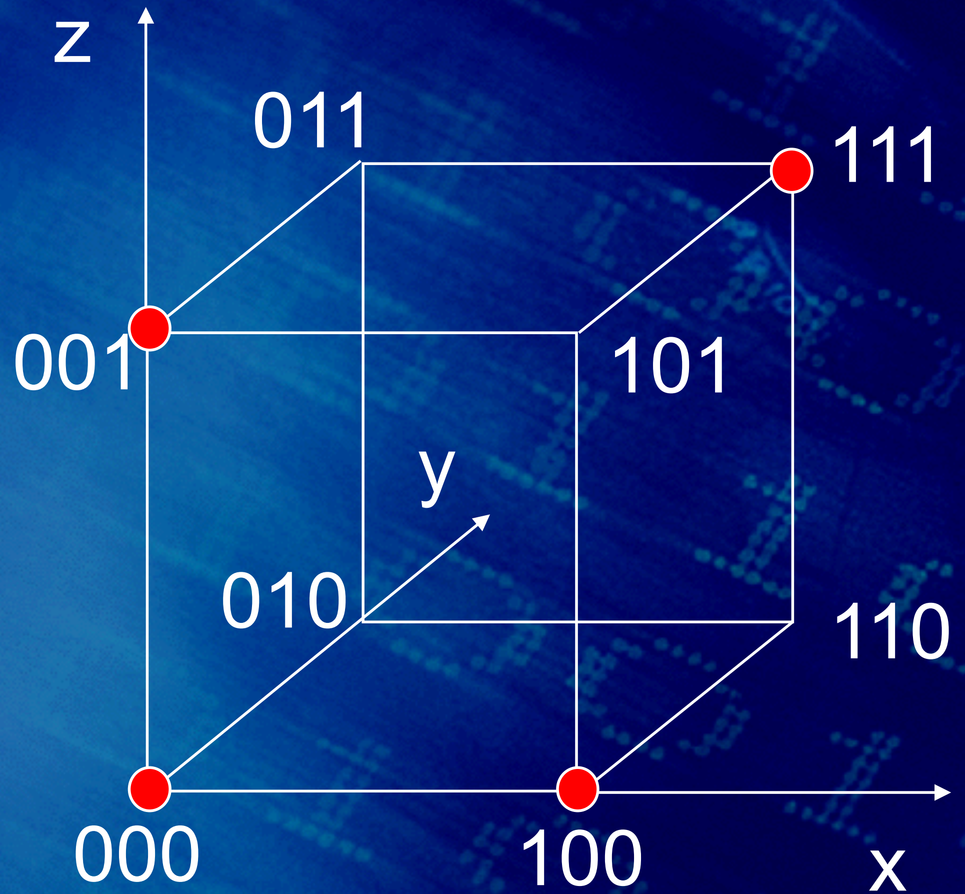
Remark #1

- Not all the sub-sets of B^n with cardinality 2^k are k-cubes.

An example of set of 2^2 vertices which is NOT a 2-cube

$B = \{ 0, 1 \}$

B^3



Remark #2

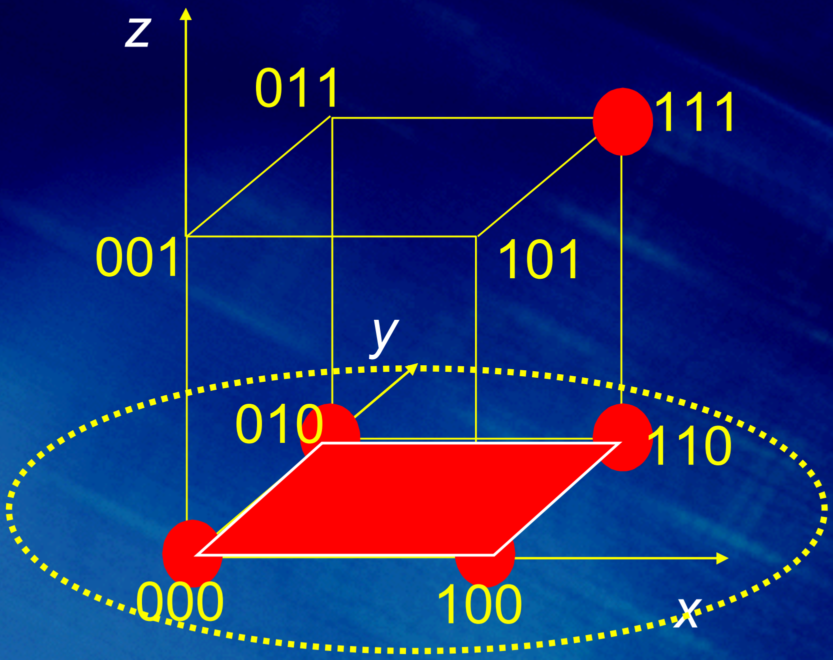
- K-cubes are easy to find when resorting to geometrical (spatial) representations, since we look for physically adjacent vertices…
- But not so easy when the function is represented in 2 dimensions, only

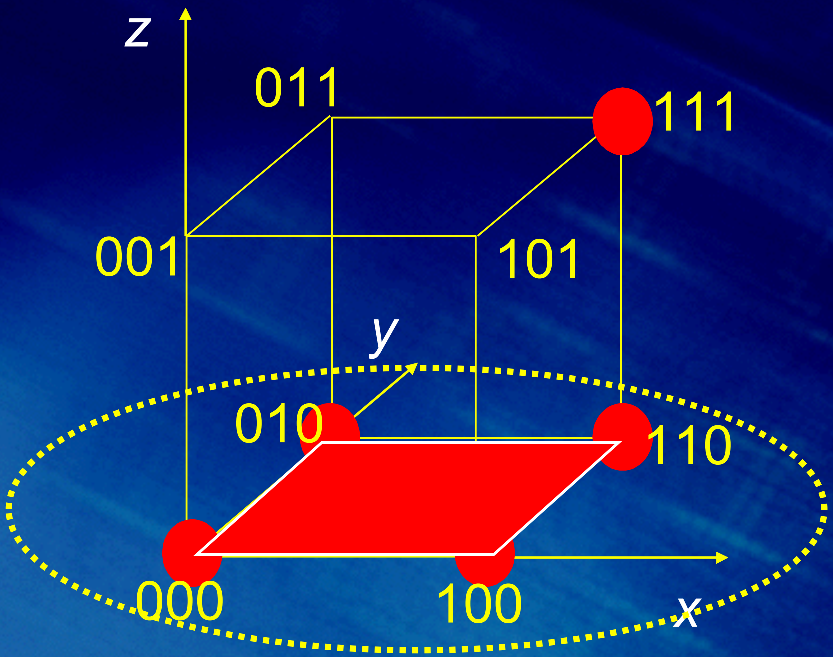
Karnaugh Maps

They try to make the task easier…

Karnaugh maps

- Values are assigned to rows and columns in such a way that cells that are physically adjacent in the geometric representation be physically adjacent in the planar representation, as well.
- Rows and columns are thus usually labeled to follow the reflected Gray code sequence:


z
011
111
001
101
y
010
110
000
100
x

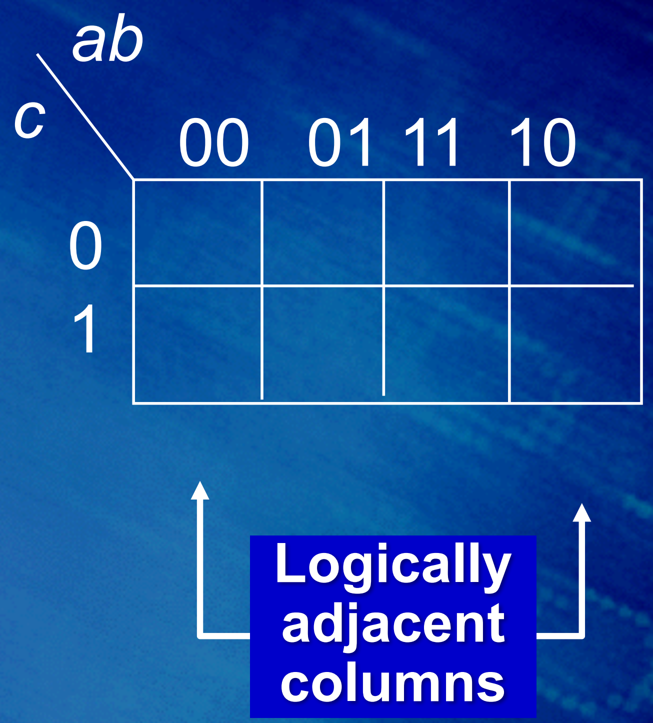


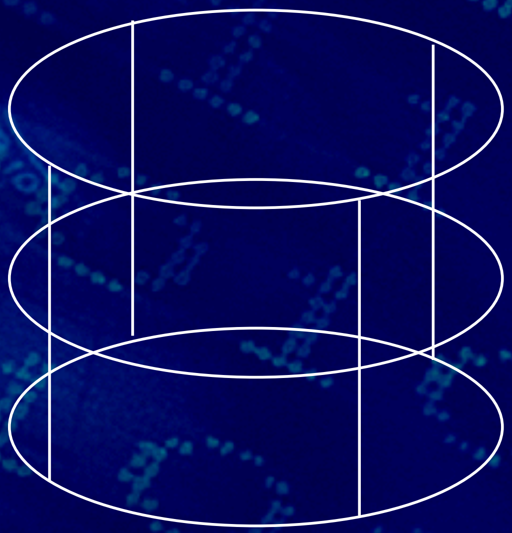
z \ x y	00	01	11	10
0	1	1	1	1
1	0	0	1	0

3 input maps

c \ ab	00	01	11	10
0				
1				

3 input maps

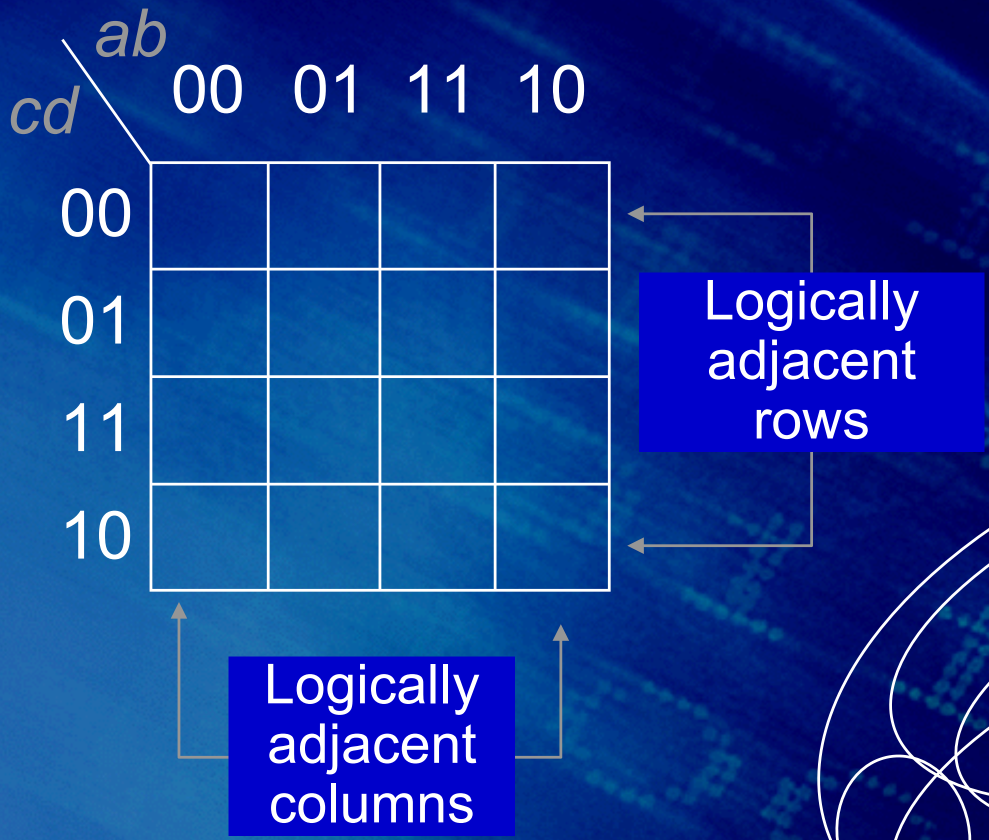




4 input maps

cd \ ab	00	01	11	10
00				
01				
11				
10				

4 input maps

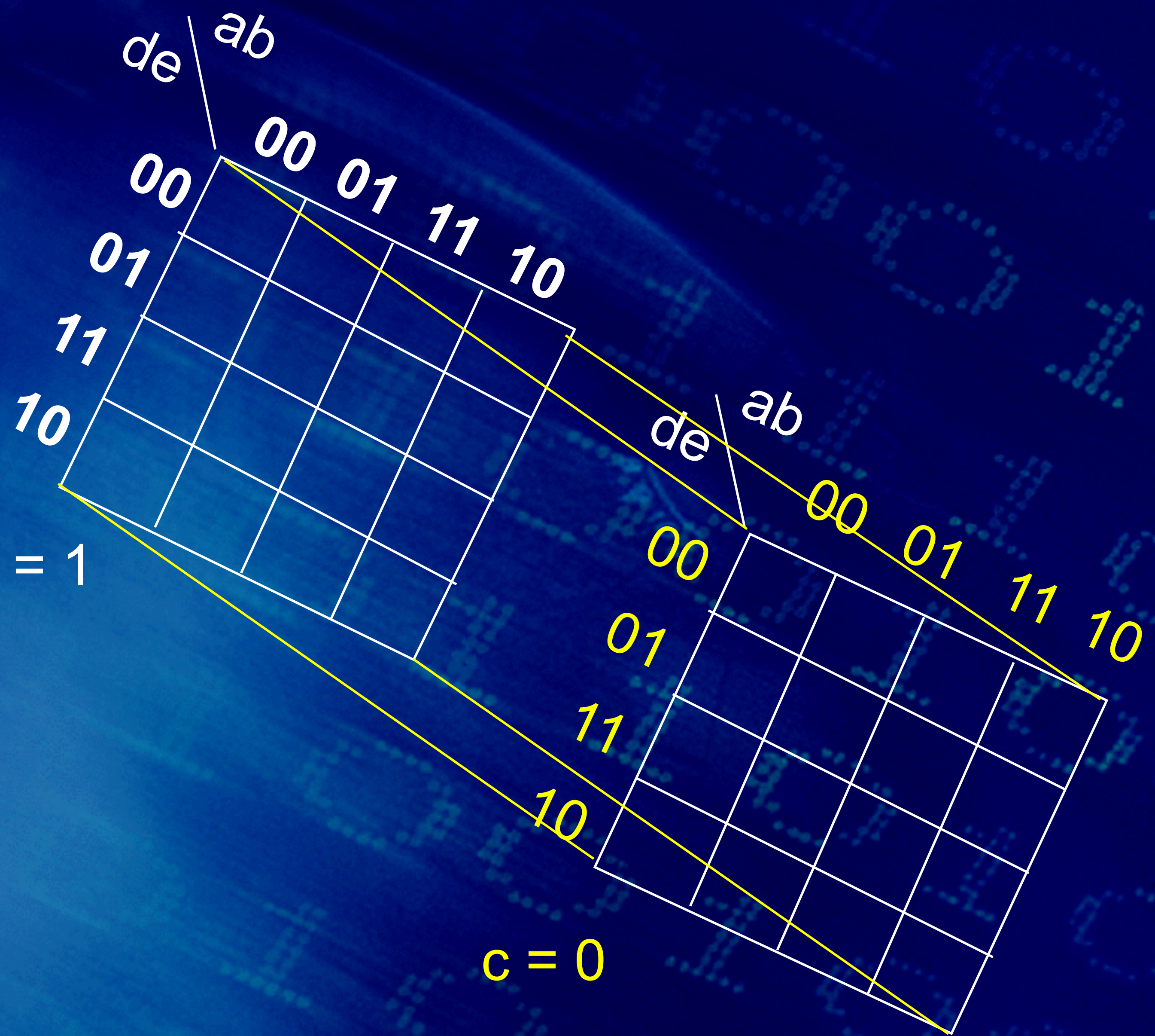


ab
cd
00 01 11 10
00
01
11
10
Logically adjacent rows
Logically adjacent columns

5 input maps

de \ ab (c)	0				1			
	00	01	11	10	00	01	11	10
00								
01								
11								
10								

Logically adjacent columns

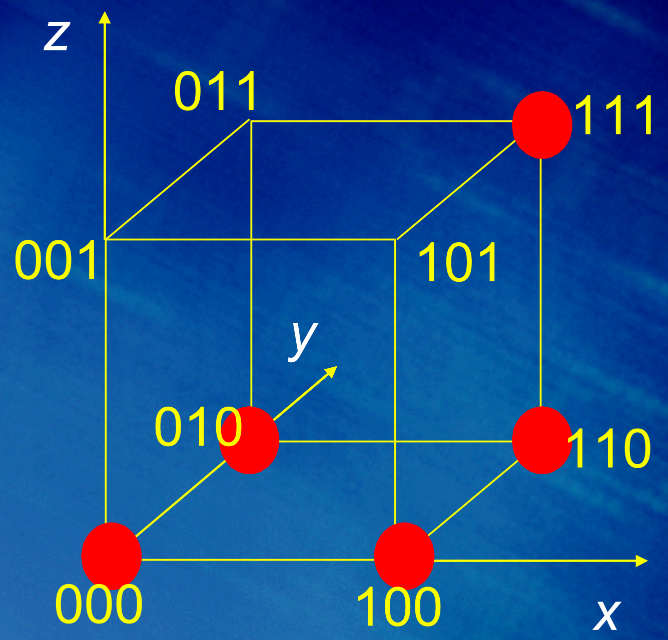


de
ab
00
01
11
10
00
01
11
10
c = 1
de
ab
00
01
11
10
00
01
11
10
c = 0

Remark #3

- The adopted solution is perfect for humans but not so efficient for machines…

Machine oriented representation

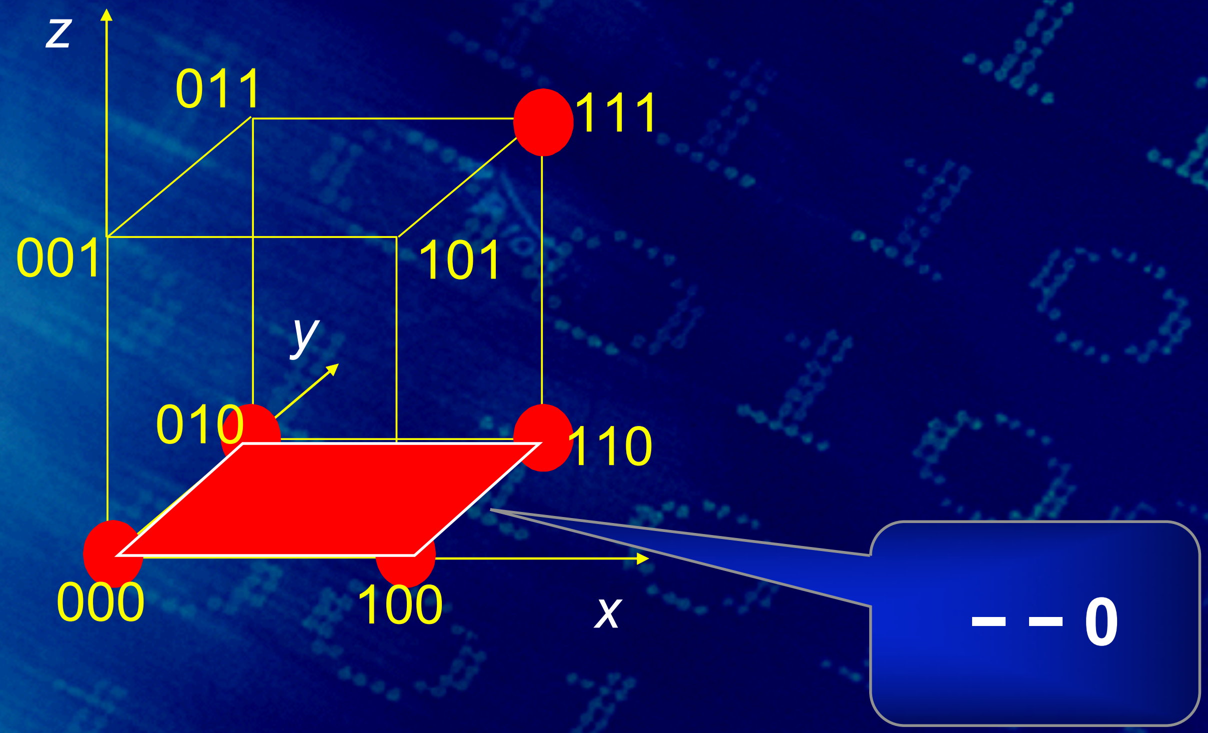


We could write:

F =
000 +
010 +
110 +
100 +
111

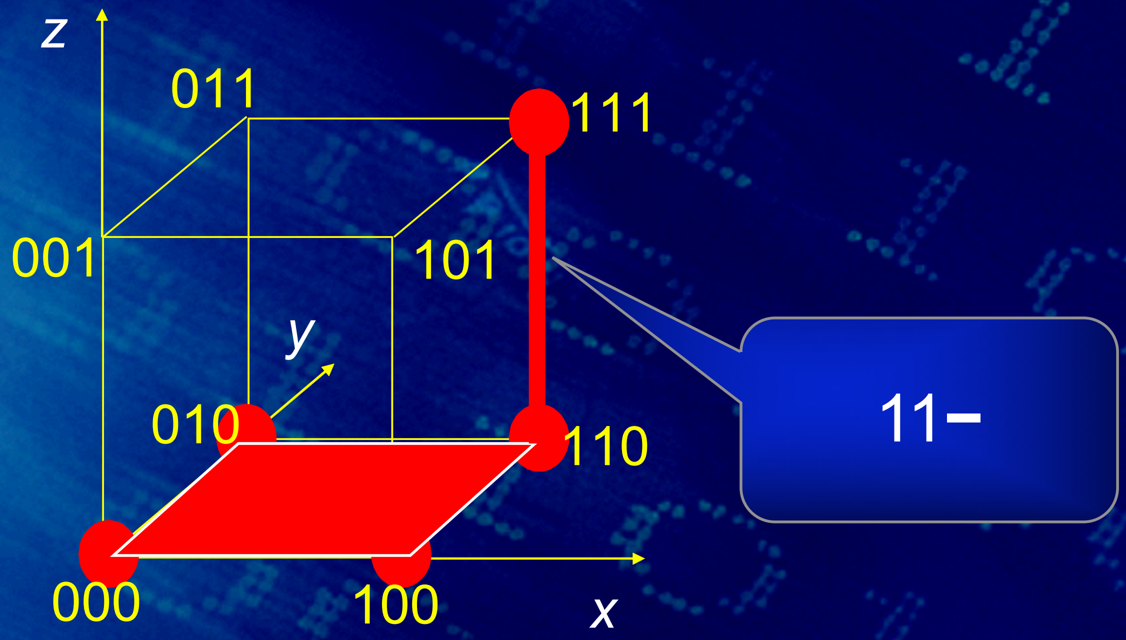
A further step

The whole "face" could be represented as



A further step *(2)*

... and the edge could be represented as



Thus obtaining

F =
000 +
010 +
110 +
100 +
111

→

F =
– – 0 +
1 1 –

Outline

- Boolean Algebras Definitions
- Examples of Boolean Algebras
- Geometric interpretation of Boolean Algebras
- Values taken by functions
- How representing functions?
- **Boolean Algebras properties**

Boolean Algebras properties

All Boolean Algebras satisfy interesting properties.

In the following we focus on some of them, particularly helpful on several applications.

The Stone Representation Theorem

"Every finite Boolean Algebra is isomorphic to the Boolean Algebra of subsets of some finite set"

[Stone, 1936]

Corollary

- In essence, the only relevant difference among the various Boolean Algebras is the cardinality of the carrier.
- Stone's theorem implies that the cardinality of the carrier of a Boolean Algebra must be a power of 2.

Consequence

- **Boolean Algebras can thus be represented resorting to the most appropriate and suitable formalisms.**
- **E.g., Venn diagrams can replace postulates.**

Duality

- Every identity is transformed into another identity by interchanging:
  - + and ·
  - ≤ and ≥
  - the identity elements 0 and 1

Examples

$a + 1 = 1$

$a \cdot 0 = 0$

$a + a' b = a + b$

$a (a' + b) = a b$

$a + (b + c) = (a + b) + c = a + b + c$

$a \cdot (b \cdot c) = (a \cdot b) \cdot c = a \cdot b \cdot c$

The inclusion relation

On any Boolean Algebra an inclusion relation ($\leq$) is defined as follows:

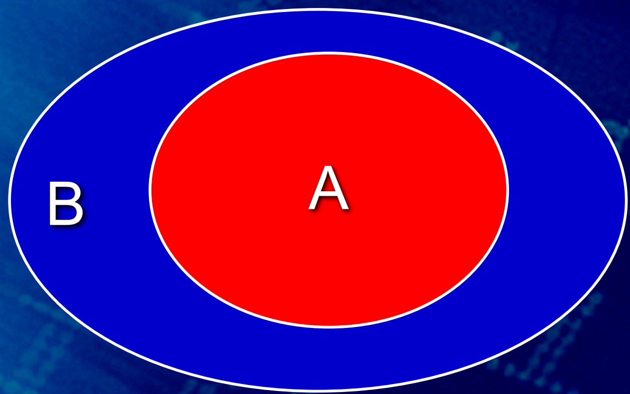
$$a \leq b \quad \text{iff} \quad a \cdot b' = 0.$$

Properties of the inclusion relation

- The inclusion relation is a *partial order* relation, i.e., it's:
  - *reflexive* : $a \leq a$
  - *antisimmetric* : $a \leq b$ e $b \leq a \Rightarrow a = b$
  - *transitive* : $a \leq b$ e $b \leq c \Rightarrow a \leq c$

The inclusion relation in the algebra of classes

- The relation gets its name from the fact that, in the *algebra of classes*, it is usually represented by the symbol $\subseteq$:
  - $A \subseteq B \quad \Leftrightarrow \quad A \cap B' = \varnothing$



 Rel. 22.12.2017

The inclusion relation in propositional calculus

- **In propositional calculus, inclusion relation corresponds to *logic implication*:**

$$a \leq b \quad \equiv \quad a \Rightarrow b$$

Note

The following expressions are all equivalent:

- $a \leq b$
- $a\,b' = 0$
- $a' + b = 1$
- $b' \leq a'$
- $a + b = b$
- $a\,b = a$.

Properties of inclusion

$a \leq a + b$

$a\,b \leq a$

Complement uniqueness

The complement of each element is unique.

Involution

$(a')' = a$

De Morgan's Laws

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

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

Generalized Absorbing

$a + a' b = a + b$

$a (a' + b) = a b$

Consensus Theorem

$$a\,b + a'\,c + b\,c = a\,b + a'\,c$$

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

Equality

$$a = b \quad \text{iff} \quad a'\,b + a\,b' = 0$$

__*Note*__

The formula

$$a'\,b + a\,b'$$

appears so often in expressions that it has been given a peculiar name: *exclusive-or* or *exor* or *modulo 2 sum*.

Implication

$$a \Rightarrow b \;=\; a' + b$$

Note

The formula

$$a \Rightarrow b$$

Is usually read as "*a implies b*" or "*if a then b*" and it is false just when a is true and b is false.

Boole's expansion theorem

Every Boolean function $f : B^n \rightarrow B$:

$$f(x_1, x_2, \ldots, x_n)$$

can be expressed as:

$$f(x_1, x_2, \ldots, x_n) =$$

$$= x_1' \cdot f(0, x_2, \ldots, x_n) + x_1 \cdot f(1, x_2, \ldots, x_n)$$

$$\forall (x_1, x_2, \ldots, x_n) \in B$$

Dual form

$$f(x_1, x_2, \ldots, x_n) =$$

$$= [\, x_1' + f(1, x_2, \ldots, x_n) \,] \cdot [x_1 + f(0, x_2, \ldots, x_n)\,]$$

$$\forall\ (x_1, x_2, \ldots, x_n) \in B$$

Remark

The expansion theorem, first proved by Boole in 1954, is mostly known as ***Shannon Expansion***.

Note

According to Stone's theorem, Boole's theorem holds independently from the cardinality of the *carrier* B.

Cancellation rule

The so called *cancellation rule*, valid in usual arithmetic algebras, cannot be applied to Boolean algebras.

This means, for instance, that from the expression:

$$x + y = x + z$$

you cannot deduce that

$$y = z$$

Proof

x	y	z	x+y	x+z	x+y = x+z	y=z
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Proof

x	y	z	x+y	x+z	x+y = x+z	y=z
0	0	0	0			
0	0	1	0			
0	1	0	1			
0	1	1	1			
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	1			

Proof

x	y	z	x+y	x+z	x+y = x+z	y=z
0	0	0	0	0		
0	0	1	0	1		
0	1	0	1	0		
0	1	1	1	1		
1	0	0	1	1		
1	0	1	1	1		
1	1	0	1	1		
1	1	1	1	1		

Proof

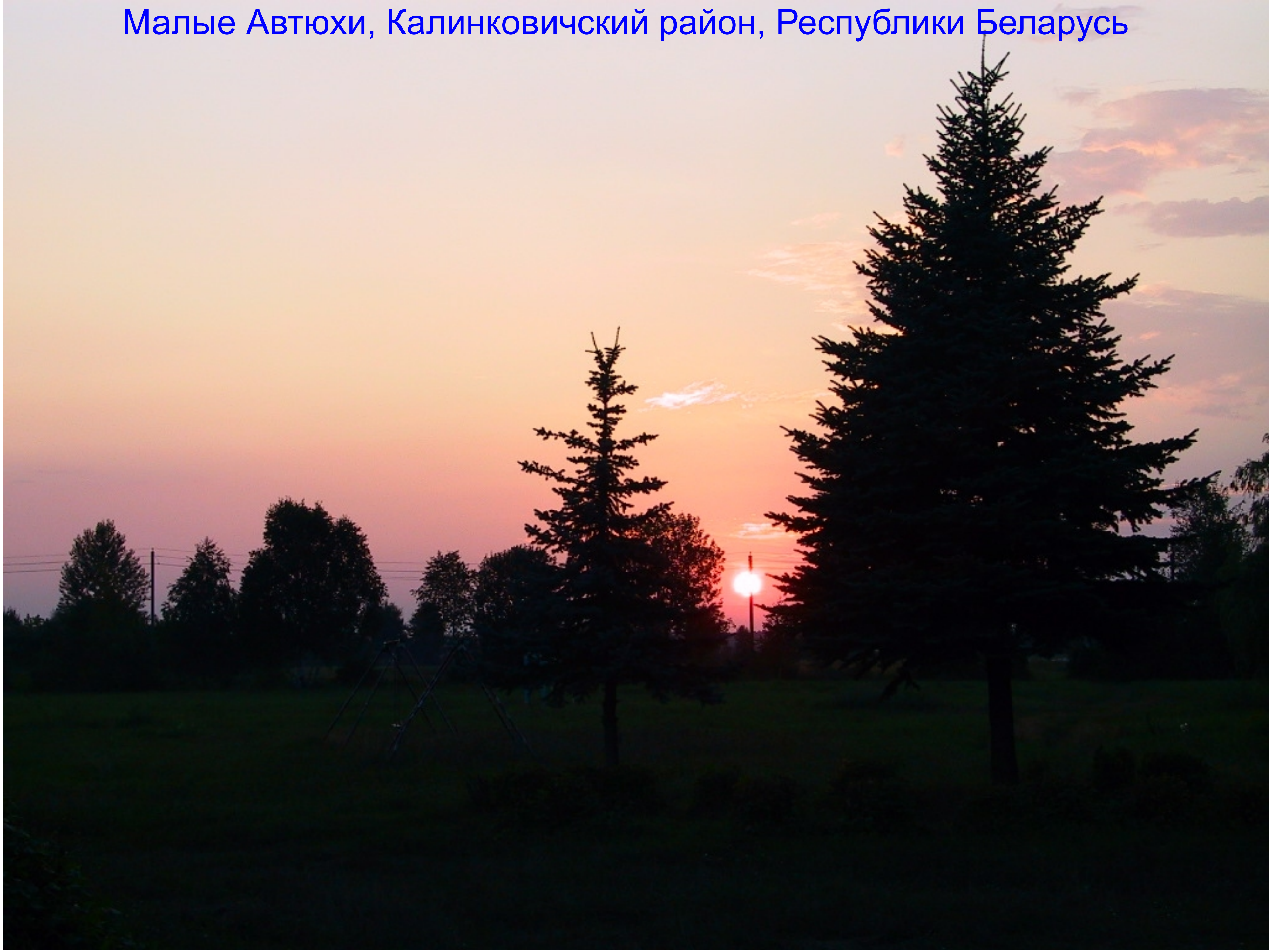
x	y	z	x+y	x+z	x+y = x+z	y=z
0	0	0	0	0	T	
0	0	1	0	1	F	
0	1	0	1	0	F	
0	1	1	1	1	T	
1	0	0	1	1	T	
1	0	1	1	1	T	
1	1	0	1	1	T	
1	1	1	1	1	T	

Proof

x	y	z	x+y	x+z	x+y = x+z	y=z
0	0	0	0	0	T	T
0	0	1	0	1	F	F
0	1	0	1	0	F	F
0	1	1	1	1	T	T
1	0	0	1	1	T	T
1	0	1	1	1	T	F
1	1	0	1	1	T	F
1	1	1	1	1	T	T

Proof

x	y	z	x+y	x+z	x+y = x+z	y=z
0	0	0	0	0	T	T
0	0	1	0	1	F	F
0	1	0	1	0	F	F
0	1	1	1	1	T	T
1	0	0	1	1	T	T
1	0	1	1	1	T	F
1	1	0	1	1	T	F
1	1	1	1	1	T	T



Малые Автюхи, Калинковичский район, Республики Беларусь