Lecture
0_2.2

Boolean Primitives and Logic Gates

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

This work is licensed under the Creative Commons BY-NC License

To view a copy of the license, visit:
http://creativecommons.org/licenses/by-nc/3.0/legalcode

Rel. 14.06.2018

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 the so called ***Basic Boolean Functions,*** often referred to as ***Boolean Primitives,*** and their corresponding ***Logic Gates***.

Rel. 14.06.2018

Prerequisites

- **Lecture 0_2.1**

Homework

- **None**

Further readings

- **No peculiar hint**

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Rel. 14.06.2018

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Rel. 14.06.2018

Basic Boolean Functions

– **Several "Basic" Boolean Functions, often referred to as Boolean Primitives, have been defined, each identified by a unique name, become a world-wide *de-facto* standard.**

They include:

- ***not***
- ***and***
- ***or***
- ***nand***
- ***nor***
- ***exor***
- ***exnor***

Why are they so significant?

- Electronic devices (often referred to as ***Logic Gates***) are available to implement each of these basic boolean functions
- Each device is usually given the same name of the corresponding boolean functions, e.g.:

AND function ⇔ AND gate

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Rel. 14.06.2018

"Logic Complement" or "NOT" function

Informal definition

It's a unary function, providing the *logic complement* of its input variable.

Representations

x' $\overline{x}$ not (x)

"not" (cont'd)

Axiomatic definition

0' = 1 not (0) = 1

1' = 0 not (1) = 0

Truth table

x	not(x)
0	1
1	0

"not" (cont'd)

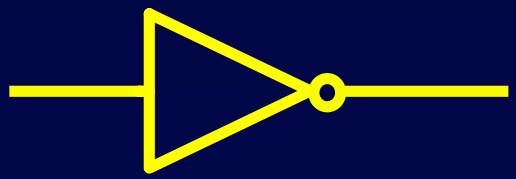
Functional definition

```
if x = 1 then not(x) = 0
         else not(x) = 1
```

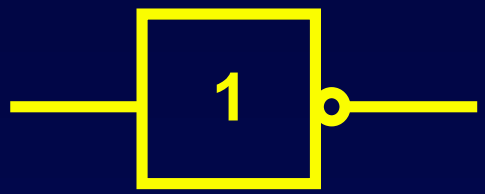
Theorems

(x')' = x **not (not (x)) = x**

NOT gate, or inverter



traditional symbol



IEEE symbol

[ANSI/IEEE standard STD 91-1984
"Graphics symbols for logic functions"]

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Rel. 14.06.2018

"Logic product" or "AND" function

Informal definition

The AND function on 2 (or more) input variables provides the value 1 iff *all* its input variables get the value 1.

Representations

x · y **x y** **and (x, y)**

"and" (cont'd)

Axiomatic definition

$0 \cdot 0 = 0$ and(0,0) = 0

$0 \cdot 1 = 0$ and(0,1) = 0

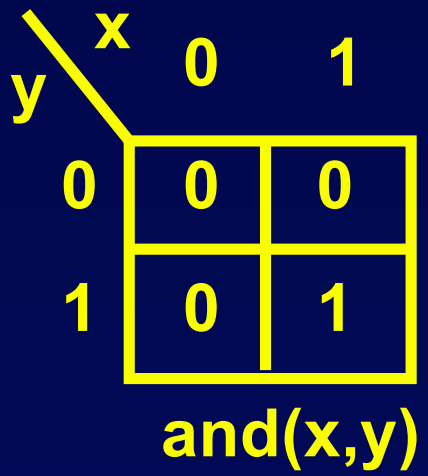
$1 \cdot 0 = 0$ and(1,0) = 0

$1 \cdot 1 = 1$ and(1,1) = 1

Truth table

x	y	and(x,y)
0	0	0
0	1	0
1	0	0
1	1	1

Karnaugh map



"and" (cont'd)

Functional representation

```
if x = 1 then and(x,y) = y
         else and(x,y) = 0
```

"and" (cont'd)

Theorems

$x \cdot 0 = 0$

$x \cdot 1 = x$

$x \cdot x = x$

$x \cdot x' = 0$

$x \cdot y = y \cdot x$

$x \cdot y \cdot z = (x \cdot y) \cdot z = x \cdot (y \cdot z)$

AND gate

traditional symbol *IEEE symbol*

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Rel. 14.06.2018

"Logic sum" or "OR" function

Informal definition

The OR function on 2 (or more) input variables provides the value 1 iff *at least one* of its input variables get the value 1.

Representations

x + y or (x, y)

8. A logician's wife is having a baby. The doctor immediately hands the newborn to the dad.

His wife asks impatiently: "So, is it a boy or a girl" ?

The logician replies: "yes".

"or" (cont'd)

Functional representation

```
if x = 1 then or(x,y) = 1
         else or(x,y) = y
```


Rel. 14.06.2018

"or" (cont'd)

Axiomatic definition

0 + 0 = 0 or(0,0) = 0

0 + 1 = 1 or(0,1) = 1

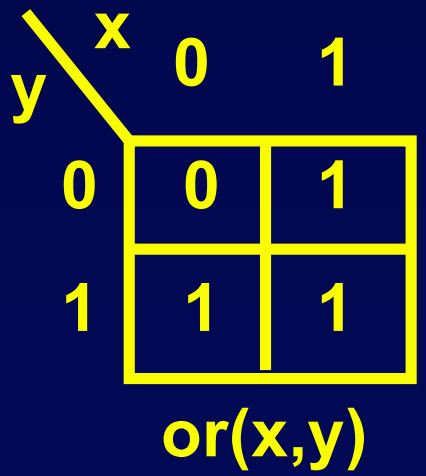
1 + 0 = 1 or(1,0) = 1

1 + 1 = 1 or(1,1) = 1

Truth table

x	y	or(x,y)
0	0	0
0	1	1
1	0	1
1	1	1

Karnaugh map



or(x,y)

"or" (cont'd)

Theorems

x + 0 = x

x + 1 = 1

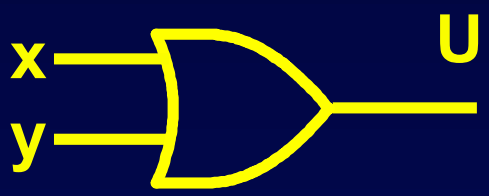
x + x = x

x + x' = 1

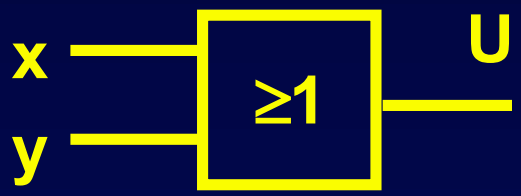
x + y = y + x

x + y + z = (x + y) + z = x + (y + z)

OR gate



traditional symbol



IEEE symbol

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Rel. 14.06.2018

"NAND" function

Informal definition

The NAND function on 2 (or more) input variables provides the value 1 iff *at least one* of its input variables get the value 0.

Representations

nand (x, y)

"nand" (cont'd)

Axiomatic definition

nand(0,0) = 1

nand(0,1) = 1

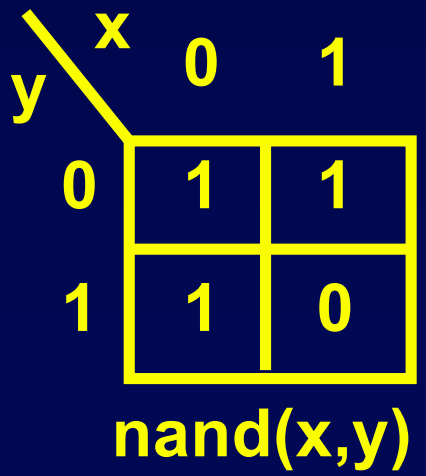
nand(1,0) = 1

nand(1,1) = 0

Truth table

x	y	nand(x,y)
0	0	1
0	1	1
1	0	1
1	1	0

Karnaugh map



nand(x,y)

"nand" (cont'd)

Functional representation

```
if x = 1 then nand(x,y) = not(y)
         else nand(x,y) = 1
```

"nand" (cont'd)

Theorems

nand (x, 0) = 1

nand (x, 1) = x'

nand (x, x) = x'

nand (x, x') = 1

nand (x, y) = nand (y, x)

nand (x, y) = not (and (x, y))

nand (x, y, z) = nand (x, and (y, z)) =

= nand (and (x, y), z) = nand (x, and (y,z))

NAND gate

traditional symbol *IEEE symbol*

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Rel. 14.06.2018

"NOR" function

Informal definition

The NOR function on 2 (or more) input variables provides the value 1 iff *none* of its input variables get the value 1.

Representations

nor (x, y)

"nor" (cont'd)

Axiomatic definition

nor(0,0) = 1

nor(0,1) = 0

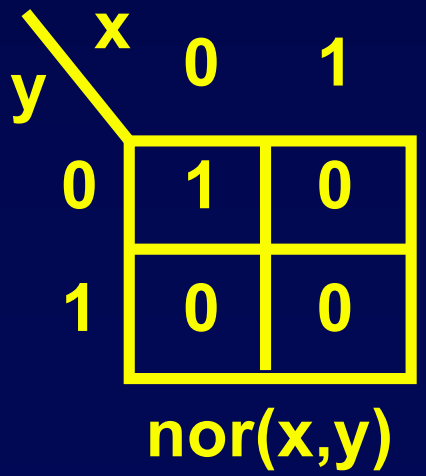
nor(1,0) = 0

nor(1,1) = 0

Truth table

x	y	nor(x,y)
0	0	1
0	1	0
1	0	0
1	1	0

Karnaugh map



"nor" (cont'd)

Functional representation

```
if x = 1 then nor(x,y) = 0
         else nor(x,y) = not(y)
```

"nor" (cont'd)

Theorems

nor (x, 0) = x'

nor (x, 1) = 0

nor (x, x) = x'

nor (x, x') = 0

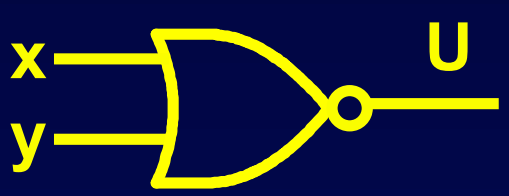
nor (x, y) = nor (y, x)

nor (x, y) = not (or (x, y))

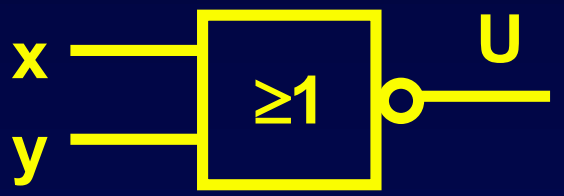
nor (x, y, z) = nor (x, or (y, z)) =

= nor (or (x, y), z) = nor (x, or (y,z))

NOR gate



traditional symbol



IEEE symbol

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Rel. 14.06.2018

"Exclusive OR" (or "EXOR" function)

Informal definition

The EXOR function on 2 (or more) input variables provides the value 1 iff *an odd* # of its input variables get the value 1.

Representations

x ⊕ y exor (x, y) xor (x, y)

"exor" (cont'd)

Axiomatic definition

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

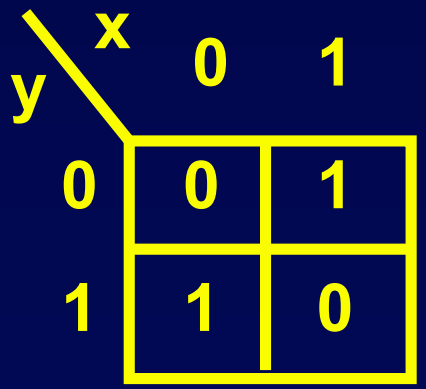
$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

Truth table

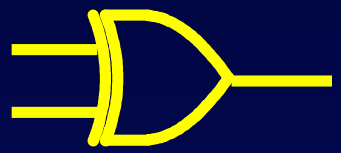
x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

Karnaugh map

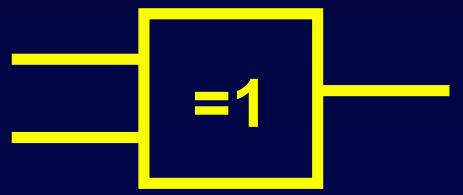


exor(x,y)

EXOR gate



traditional symbol



IEEE symbol

"exor" (cont'd)

1st functional representation

```
if x = 1 then exor(x,y) = not(y)
         else exor(x,y) = y
```

The exor function can be seen as a *controlled inverter :*

x	y	exor(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

Rel. 14.06.2018

"exor" (cont'd)

2nd functional representation

```
if x = y then exor(x,y) = 0
         else exor(x,y) = 1
```

The exor function can be seen as a *comparator :*

x	y	exor(x,y)
0	0	0
1	1	0
1	0	1
0	1	1

Rel. 14.06.2018

"exor" (cont'd)

3rd functional representation

```
if (x+y)mod 2 = 0 then exor(x,y) = 0
                  else exor(x,y) = 1
```

The exor function can be seen as a *modulo 2 adder*

x	y	(x+y)mod 2	exor(x,y)
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Rel. 14.06.2018

Exor - Theorems

exor (x, 0) = x

exor (x, 1) = x'

exor (x, x) = 0

exor (x, x') = 1

exor (x, y) = exor (y, x)

exor (x, y) = exor (x' , y')

exor (x, y') = exor (x', y) = not (exor(x,y))

exor (x, y) = x y' + x' y = (x y)' (x + y)

Exor - Composition

exor (x, y, z) = exor (x, exor (y, z))
= exor (exor (x,y), z)

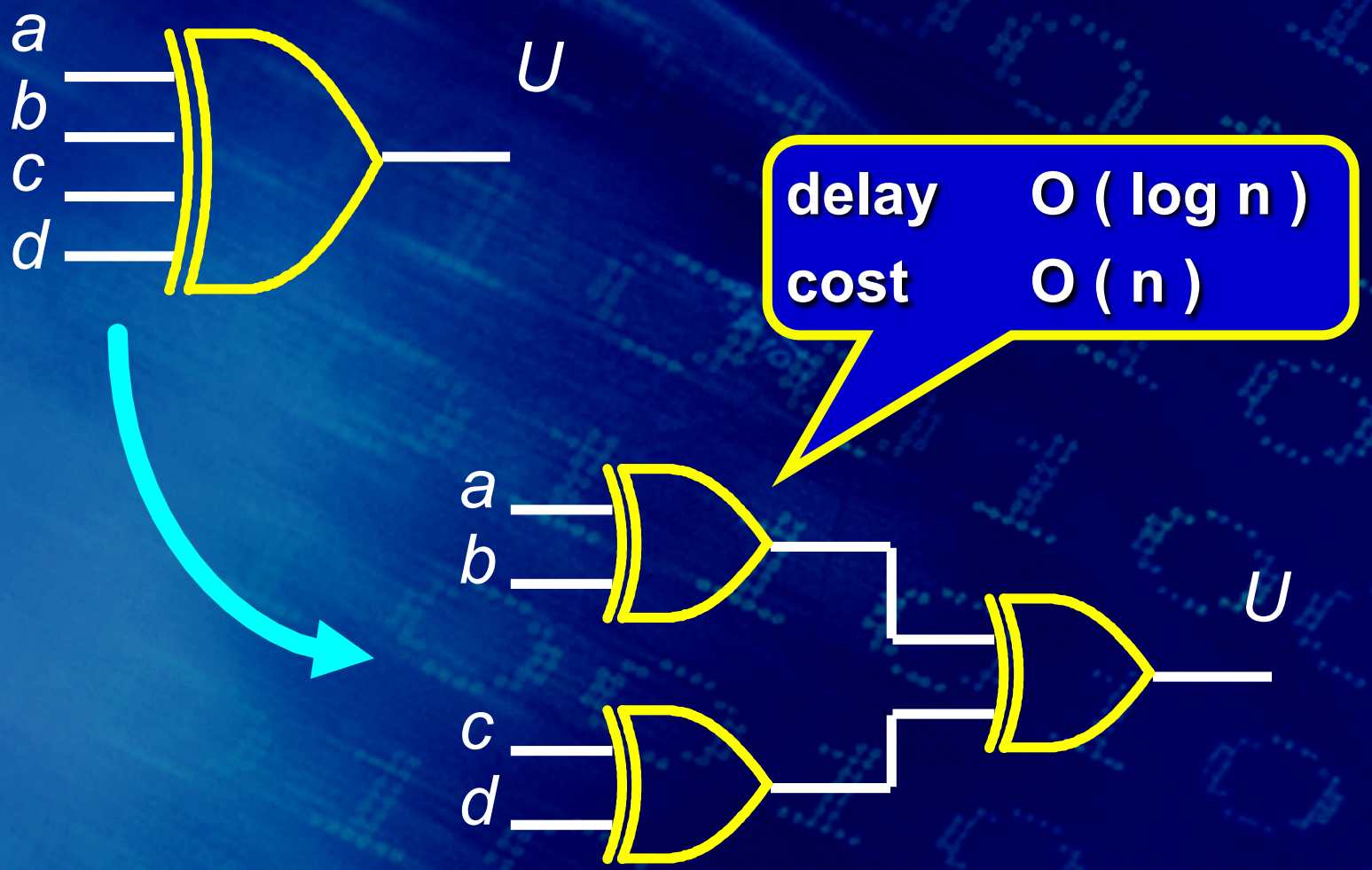
Exor - Composition

exor (x, y, z) = exor (x, exor (y, z))
= exor (exor (x,y), z)

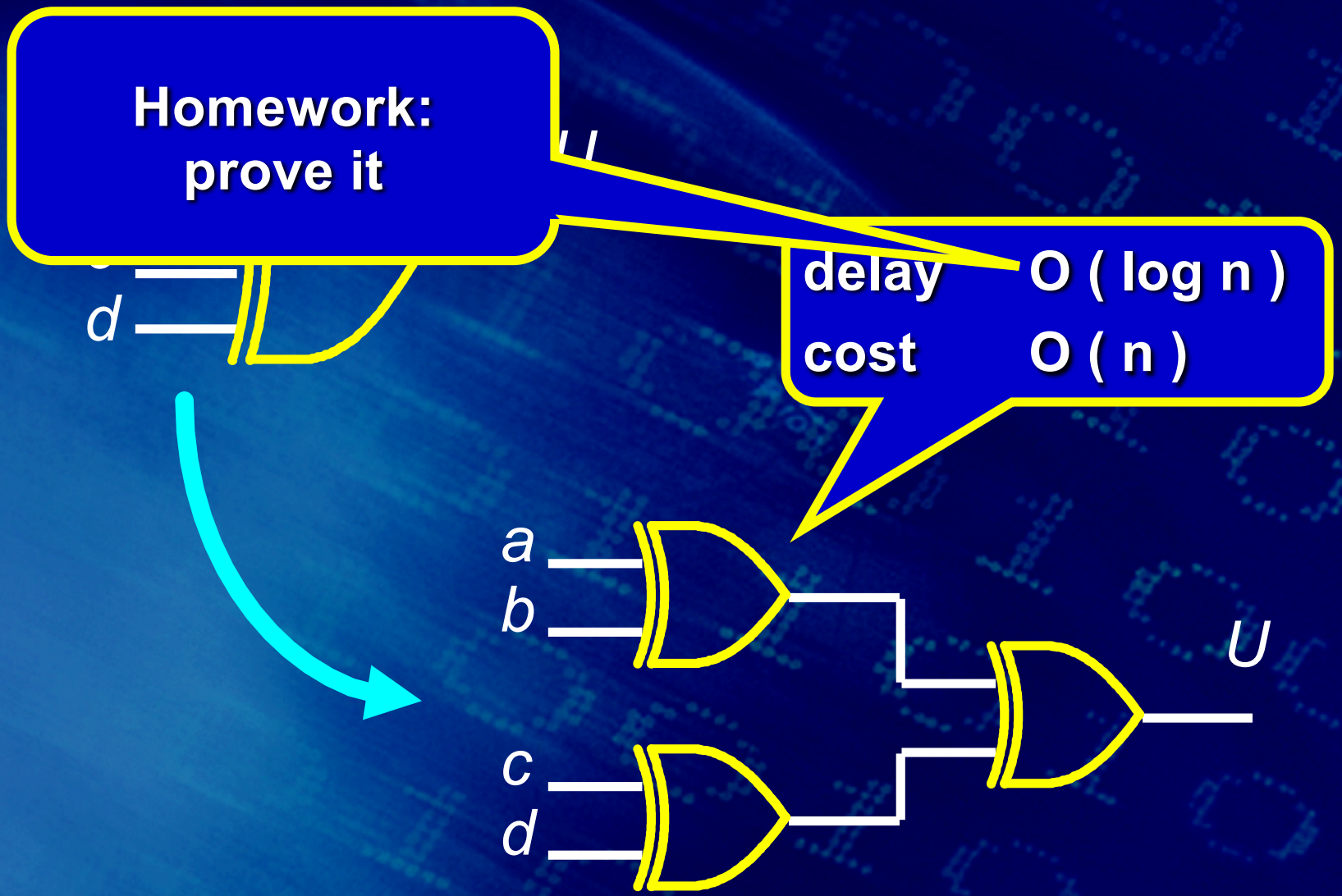
Exor - Composition - Remark

When using 2-input exor gates to implement N-input exor functions, 2 solutions are possible:

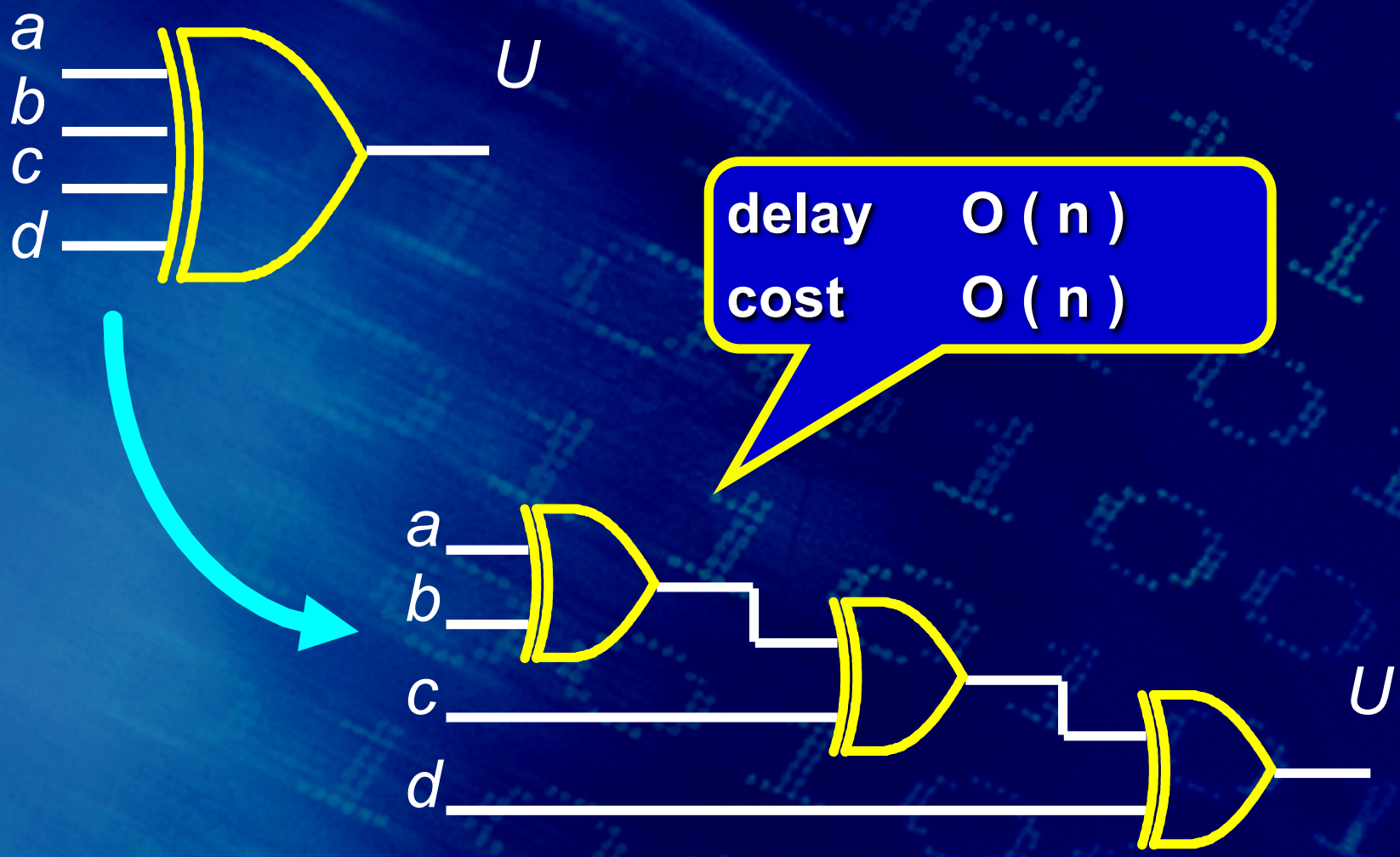
Exor - Composition – 1st solution



Exor - Composition – 1st solution



Exor - Composition – 2nd solution



Karnaugh map for a 4-inputs exor

exor (a,b,c,d)

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

"EXNOR" function

Informal definition

The EXNOR function on 2 (or more) input variables provides the value 1 iff *an even* # of its input variables get the value 1.

Representations

x ⊙ y **exnor (x, y)**

"exnor" (cont'd)

Axiomatic definition

exnor(0,0) = 1

exnor(0,1) = 0

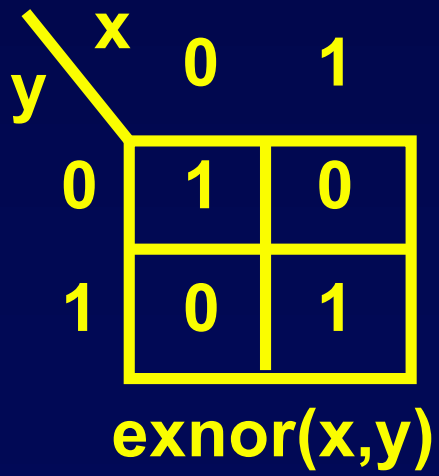
exnor(1,0) = 0

exnor(1,1) = 1

Truth table

x	y	exnor(x,y)
0	0	1
0	1	0
1	0	0
1	1	1

Karnaugh map



exnor(x,y)

"exnor" (cont'd)

Functional representations

```
if x = 1 then exor(x,y) = y
         else exor(x,y) = not(y)

if x = y then exor(x,y) = 1
         else exor(x,y) = 0
```

Theorems

exnor (x, y) = exor (x, y)' = exor (x', y) = exor (x, y')

exnor (x, y) = x y + x' y'

Outline

- Basic Boolean Functions:
  - not
  - and
  - or
  - nand
  - nor
  - exor
  - exnor
- Complete sets of logic primitives

Complete sets of logic primitives

- **Question:**
  - **Is it possible to represent any Boolean Function resorting to a whatever subset of the basic functions (or primitives) seen so far?**

Complete sets of logic primitives

- Question:
  - Is it possible to represent any Boolean Function resorting to a whatever subset of the basic functions (or primitives) seen so far?
- Answer:
  - No: you have to carefully select the subset!

Strongly complete set

- **A set of logic primitives is said to be *strongly complete* if any arbitrary logic function f (x_1, x_2, …, x_n) can be implemented by connecting a finite number of them, assuming the only asserted input variables x_1, x_2, …, $x_n$be available.**

Strongly complete set (2)

The definition implies that, in particular, the constant functions:

- $f(x_1, x_2, \ldots, x_n) = 1$
- $f(x_1, x_2, \ldots, x_n) = 0$

must also be producible.

Examples of Strongly complete sets

- and, or, not
- and, not
- or, not
- nand
- nor
- …

Problem

- **How determining whether a set of primitives is strongly complete?**

Intuitive approach

- **Check whether the set one can generate the 3 functions and, or and not.**

Example

The set { **nand** } is strongly complete since:

- $x' = (x \cdot x)'$
- $x \cdot y = ((x \cdot y)')'$
- $x + y = ((x + y)')' = (x' \cdot y')'$
- $1 = (x \cdot x')' = (x \cdot (x \cdot x)')'$
- $0 = ((x \cdot (x \cdot x)') \cdot (x \cdot (x \cdot x)')')'$

Systematic approach

- **It resorts to the Theorem of Post.**

Function properties

- **Some properties need to be preliminarily defined.**

Property 1 : Zero Preservation

A boolean function $f(x_1, x_2, \ldots, x_n)$ is said to be a *preserving zero function* iff

$$f(0, 0, \ldots, 0) = 0$$

Property 2 : One Preservation

A boolean function $f(x_1, x_2, \ldots, x_n)$ is said to be a *preserving one function* iff

$$f(1, 1, \ldots, 1) = 1$$

Property 3 : Self-Duality

A boolean function $f(x_1, x_2, \ldots, x_n)$ is said to *Self-dual* iff

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

Property 4 : Monotonicity

A boolean function $f(x_1, x_2, \ldots, x_n)$ is said to *monotonic* iff

$$f(a) \geq f(b) \qquad \forall\, a,b \mid a \geq b$$

Property 5 : Linearity

A boolean function $f(x_1, x_2, \ldots, x_n)$ is said to *linear* iff it can be expressed as

$$f(x_1, x_2, \ldots, x_n) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus \ldots \oplus a_nx_n$$

where

$a_i \in \{ 0,1 \}$ for $i = 0, 1, 2, \ldots, n$

Theorem

Be S = { f_1, f_2, …, f_n } a set of functions, all having one of the 5 previous properties.

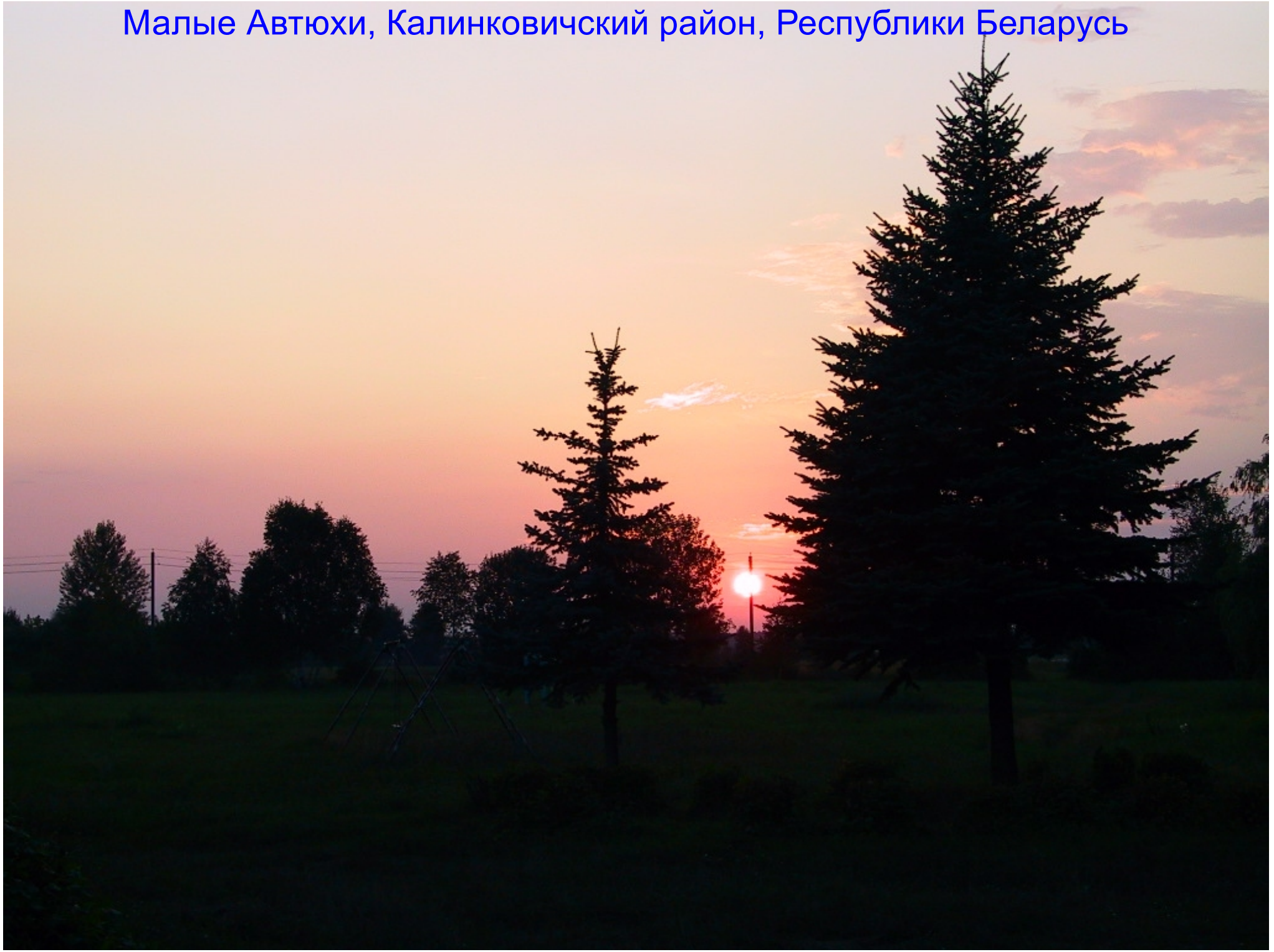
Any boolean function obtained combining the functions of S has the same property, i.e., :

- **a linear function of linear functions is a linear function**
- **a monotonic function of monotonic functions is a monotonic function**
- **…**

Theorem of Post

A set of functions is strongly complete iff it contains:

- ***at least one function not zero preserving***
- ***at least one function not one preserving***
- ***at least one non-self-dual function***
- ***at least one non-linear function***
- ***at least one non-monotonic function.***


Малые Автюхи, Калинковичский район, Республики Беларусь