Fachgebiet Rechnersysteme

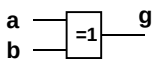
1. Boolean Algebra

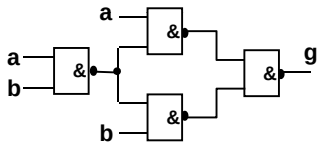
Verification Technology

Content

1.1 Boolean algebra basics (recap)
1.2 Reasoning about Boolean expressions

- **The problem of logic verification:** Show that two circuits implement the same boolean function, or: Show that a circuit correctly implements a specification



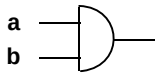


1.1 Boolean Algebra Basics

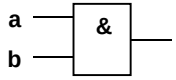
- **AND-operation (conjunction)**

a	b	a·b
0	0	0
0	1	0
1	0	0
1	1	1

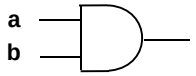
Gate representation



German (old)



IEEE-standard



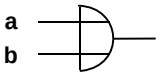
US-standard (old)

✉ 0-dominance

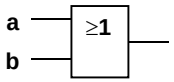
- **OR-operation (disjunction)**

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

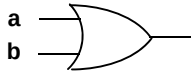
Gate representation



German (old)



IEEE-standard



US-standard (old)

✉ 1-dominance

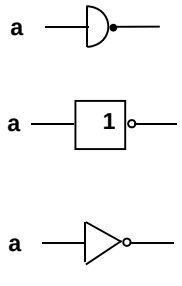
1. Boolean Algebra — 5

1.1 Boolean algebra basics

- **NOT-operation (negation, complement)**

a	$\overline{a}$
0	1
1	0

Gate representation



German (old)

IEEE-Standard

US-standard (old)

1. Boolean Algebra — 6

1.1 Boolean algebra basics

- **More notations ...**

$a \bullet b$	$a \wedge b$	a & b
$a + b$	$a \vee b$	a | b
$\overline{a}$	$\neg a$	a'

Propositional Calculus

1. Boolean Algebra — 7

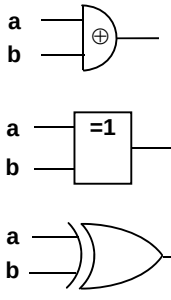
1.1 Boolean algebra basics

- **More operations:**
  - **(E)XOR $\oplus$ (exclusive-or, unequal, addition modulo 2)**

Definition: $a \oplus b = a \cdot \overline{b} + \overline{a} \cdot b$

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Gate representation



German (old)

IEEE-standard

US-standard (old)

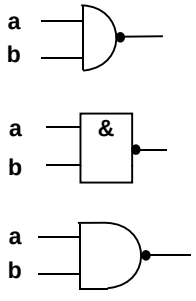
1. Boolean Algebra — 8

1.1 Boolean algebra basics

- **NAND**

a	b	$\overline{(a \cdot b)}$
0	0	1
0	1	1
1	0	1
1	1	0

Gate representation



German (old)

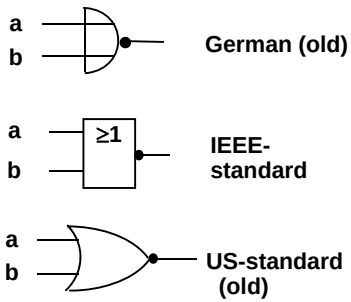
IEEE-standard

US-standard (old)

↗ **NOR**

a	b	$\overline{(a+b)}$
0	0	1
0	1	0
1	0	0
1	1	0

Gate representation



↗ **Implikation →:**

$$a \rightarrow b = \bar{a} + b$$

↗ **Equivalence ≡ (equality):**

$$a \equiv b = a \cdot b + \bar{a} \cdot \bar{b}$$

— The equivalence-function equals one iff the arguments have equal values

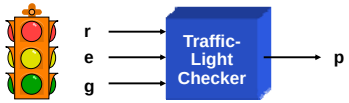
— The exor-function equals one iff the arguments have unequal values

— Generally: $a \equiv b = \overline{(a \oplus b)}$

- **Boolean functions**
  — Traffic-light checker: inputs: r red, g green, e yellow

$2^3 = 8$ cases

r	e	g	p?
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



- r, e, g are Boolean variables
- B = {0, 1} is the set of Boolean values
- A Boolean function in n variables is a mapping

$$F: B^n \rightarrow B$$

✉ Such a function models a circuit with n inputs and one output, in the example we have $B^3 \rightarrow B$

$2^3 2^2 2^1 2^0$

	x_3	x_2	x_1	x_0	f
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Representation of Boolean functions by means of function tables and Veitch-(Karnaugh-) maps

x_3

0_0	1_4	1_{12}	0_8
1_1	1_5	1_{13}	1_9
0_3	1_7	1_{15}	0_{11}
0_2	0_6	0_{14}	0_{10}

x_0 x_1 x_2

- Boolean terms are textual representations of Boolean functions, e.g.,

$$a \cdot b \cdot c + \bar{c} \cdot \bar{a} \cdot (e + b)$$

- The syntax of Boolean terms:
  - The constants 0 and 1 are Boolean terms
  - Literals (variables and complemented variables) are Boolean terms, for instance, a and $\bar{a}$
  - If a and b are Boolean terms then so are

$$(a \cdot b), \quad (a + b), \quad \bar{a}$$

- Product-terms (products, cubes) are conjunctions of literals
  - Each variable occurs only once
    - Example: $x \bullet \bar{y} \bullet z$ or $x\bar{y}z$ for short
  - Special case: Minterm (product-term in all variables)
- Sum-terms (clauses) are disjunctions of literals
  - Special case: Maxterm (sum-term in all variables)

- A Sum-of-Products (sop) (also called a disjunctive normal form, dnf) is a disjunction of products
  - Example: $cyz + d + a\bar{x}$
- A Product-of-Sums (pos) (also called a conjunctive normal form, cnf) is a conjunction of sums
  - Example: $(\bar{c} + x + b)(s + \bar{x})(a + x)$

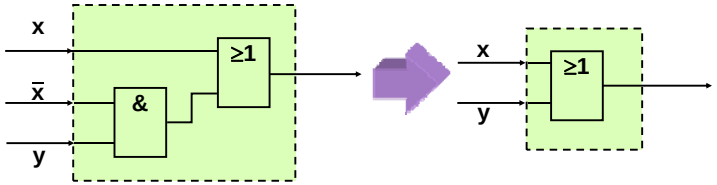
- **The Rules of Boolean Algebra:**

(T1)	$x + 0 = x$	(T1')	$x \cdot 1 = x$	Identity
(T2)	$x + 1 = 1$	(T2')	$x \cdot 0 = 0$	0/1-Element
(T3)	$x + x = x$	(T3')	$x \cdot x = x$	Idempotence
(T4)	$\bar{\bar{x}} = x$	(T4')	$\bar{1} = 0$	Involution
(T5)	$x + \bar{x} = 1$	(T5')	$x \cdot \bar{x} = 0$	Complement

(T6)	$x + y = y + x$	(T6')	$x \cdot y = y \cdot x$	Commutativity
(T7)	$(x + y) + z = x + (y + z) = x + y + z$			Associativity
(T7')	$(x \cdot y) \cdot z = x \cdot (y \cdot z) = x \cdot y \cdot z$			
(T8)	$x \cdot y + x \cdot z = x \cdot (y + z)$			Distributivity
(T8')	$(x + y) \cdot (x + z) = x + y \cdot z$			
(T9)	$\overline{(x + y)} = \bar{x} \cdot \bar{y}$	(T9')	$\overline{(x \cdot y)} = \bar{x} + \bar{y}$	De Morgan's Law
(T10)	$\overline{(x_1 + \ldots + x_n)} = \bar{x}_1 \cdot \ldots \cdot \bar{x}_n$			Generalized De Morgan's Law
(T10')	$\overline{(x_1 \cdot \ldots \cdot x_n)} = \bar{x}_1 + \ldots + \bar{x}_n$			

(T11)	$x + \bar{x} \cdot y = x + y$	(T11') $x \cdot (\bar{x} + y) = x \cdot y$	Absorption
(T12)	$x + x \cdot y = x$	(T12') $x \cdot (x + y) = x$	

✉ Very important for the simplification of terms and circuits! Example:



- **Boole's Expansion Theorem** (1849)
  ✉ also attributed to Shannon

1. Boolean Algebra — 21

1.1 Boolean algebra basics

	x_1	x_2	x_3	x_4	f
0	0	0	0		0
0	0	0	1		1
0	0	1	0		0
0	0	1	1		0
0	1	0	0		1
0	1	0	1		1
0	1	1	0		0
0	1	1	1		1
1	0	0	0		0
1	0	0	1		1
1	0	1	0		0
1	0	1	1		0
1	1	0	0		1
1	1	0	1		1
1	1	1	0		0
1	1	1	1		1

$f(0,x_2,x_3,x_4)$

$f(1,x_2,x_3,x_4)$

- Idea: Decompose a function into two sub-functions which do not depend on some variable, e.g., x_1

1. Boolean Algebra — 22

1.1 Boolean algebra basics

- **Expansion Theorem**: a Boolean Function f can be decomposed according to a variable x as follows

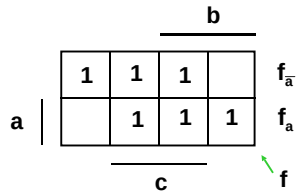
$$f = x \cdot f_x + \overline{x} \cdot f_{\overline{x}}$$

  - f_x, $f_{\overline{x}}$ are called the **Positive (1-)** and **Negative (0-) Cofactors** of f with respect to x
  - The cofactors can be calculated by replacing the variable x by the constants 0 and 1, respectively

1. Boolean Algebra — 23

1.1 Boolean algebra basics

— Illustration in the Veitch-diagram:

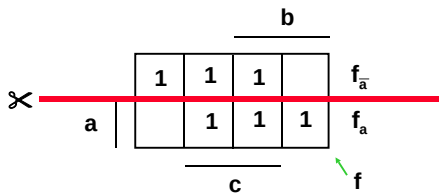


$$f = c + \overline{a}\,\overline{b} + ab$$

1. Boolean Algebra — 24

1.1 Boolean algebra basics

— Illustration in the Veitch-diagram:

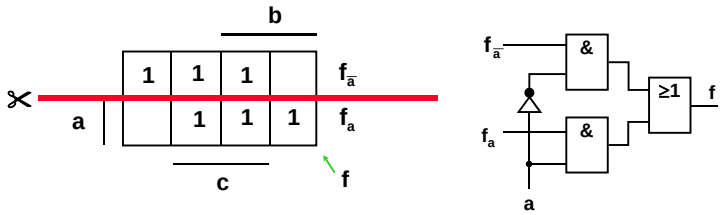


$$f = c + \overline{a}\,\overline{b} + ab$$
$$f = a \cdot f_a + \overline{a} \cdot f_{\overline{a}}$$

1. Boolean Algebra — 25

1.1 Boolean algebra basics

— Circuit realization:



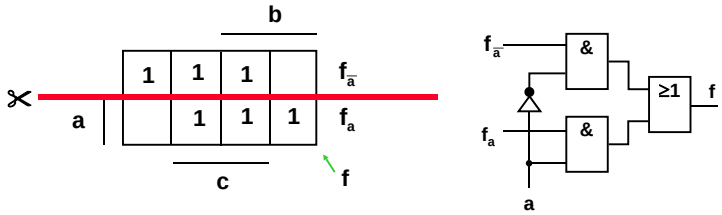
$f = c + \bar a \bar b + ab$

$f = a \cdot f_a + \bar a \cdot f_{\bar a}$

1. Boolean Algebra — 26

1.1 Boolean algebra basics

↗ The cofactors are calculated by replacing a by 0 and 1, respectively:



$f = c + \bar a \bar b + ab,$

$f_a = c + \bar 1 \bar b + 1b = c + b,$

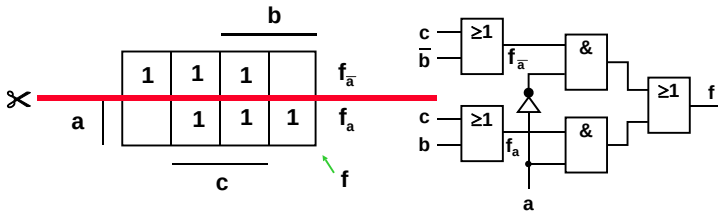
$f_{\bar a} = c + \bar 0 \bar b + 0b = c + \bar b,$

$f = a \cdot (c + b) + \bar a \cdot (c + \bar b)$

1. Boolean Algebra — 27

1.1 Boolean algebra basics

↗ The cofactors are calculated by replacing a by 0 and 1, respectively plus simplifications:



$f = c + \bar a \bar b + ab,$

$f_a = c + \bar 1 \bar b + 1b = c + b,$

$f_{\bar a} = c + \bar 0 \bar b + 0b = c + \bar b,$

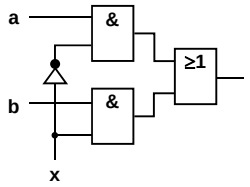
$f = a \cdot (c + b) + \bar a \cdot (c + \bar b)$

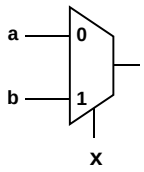
1. Boolean Algebra — 28

1.1 Boolean algebra basics

- The circuit used above is called a 2:1-Multiplexor

Symbol:

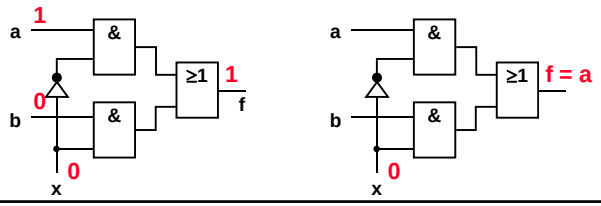




✉ Depending on x, the output is either equal to a or equal to b

1. Boolean Algebra — 29
1.1 Boolean algebra basics

- **Remark: the calculation of a cofactor can be viewed as a step of symbolic simulation:**
  - **Traditional simulation determines the value at the output of a gate-network for given values at the inputs**
  - **The calculation of a cofactor determines the function at the output if some inputs assume a constant value (0 or 1)**



1. Boolean Algebra — 30
1.1 Boolean algebra basics

- **3 operations between cofactors:**
  - **The Boolean Difference (Exor)**
  - **The Universal Quantification (And)**
  - **The Existential Quantification (Or)**

1. Boolean Algebra — 31
1.1 Boolean algebra basics

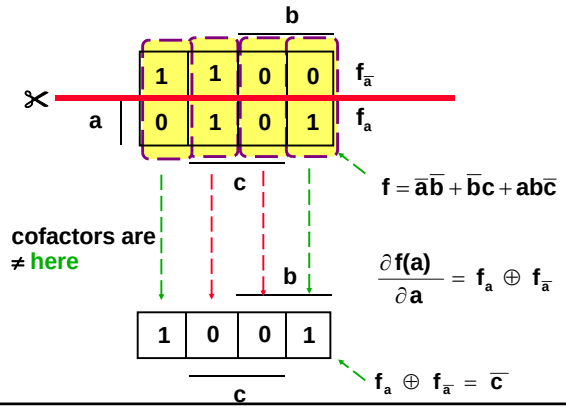
- **The Boolean Difference**

$$\frac{\partial f(x)}{\partial x} = f_x \oplus f_{\bar{x}}$$

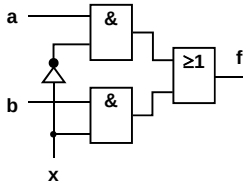
  - **XOR of both cofactors**
  - **Characterizes all situations for which a change of x implies a change of f**

1. Boolean Algebra — 32
1.1 Boolean algebra basics

— **Example:**



— **Example: 2:1-multiplexor**



$$f = x \cdot b + \bar{x} \cdot a$$

$$\frac{\partial f(a)}{\partial a} = f_a \oplus f_{\bar{a}}$$

$$= (b + \bar{x}) \oplus (x \cdot b)$$

$$= (b + \bar{x}) \cdot (\bar{x} + \bar{b}) +$$

$$(\bar{b} \cdot x) \cdot (x \cdot b) = \bar{x}$$

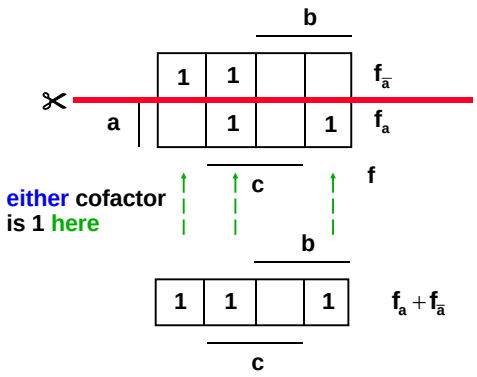
Of course! for x = 0 we have f = a, so a≠ implies f≠ !

- **The Boolean Existential Quantifer**

$$(\exists\, x : f(x)) = f_x + f_{\bar{x}}$$

↗ **The disjunction of both cofactors**

$$(\exists\, a : f(a)) = f_a + f_{\bar{a}}$$



- **The calculation of the Boolean existential quantifier is particularly easy for a Boolean expression in dnf:**

↗ $f = \underbrace{\ldots \cdot x + \ldots}_{\text{products with } x} + \underbrace{\ldots \cdot \bar{x} + \ldots}_{\text{products with } \bar{x}} + \underbrace{\ldots}_{\text{products without } x, \bar{x}}$

$(\exists x: f(x)) = f_x + f_{\bar{x}}$

$$(\exists\, x: f(x)) = \ldots \cdot 1 + \ldots + \ldots \cdot 0 + \ldots + \ldots + \quad \}\, f_x$$
$$\ldots \cdot 0 + \ldots + \ldots \cdot 1 + \ldots + \ldots \quad \}\, f_{\bar{x}}$$
$$= \underbrace{\ldots \quad + \ldots + \ldots \quad + \ldots + \ldots}_{\text{original expression where all occurences of } x, \bar{x} \text{ are eliminated}}$$

— **Example:** $f = \bar{a}\bar{b} + \bar{b}c + ab\bar{c}$

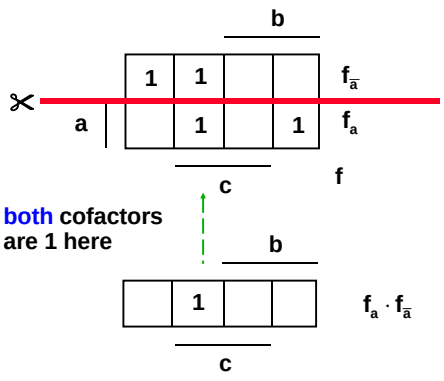
$\exists a: f(a) = \bar{b} + \bar{b}c + b\bar{c} \quad (= \bar{b} + \bar{c})$

- The Boolean Universal Quantor

$$(\forall x : f(x)) = f_x \cdot f_{\bar{x}}$$

↗ The conjunction of both cofactors

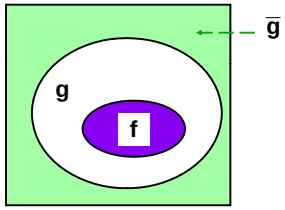
$$(\forall a : f(a)) = f_a \cdot f_{\bar{a}}$$



- The calculation of the Boolean universal quantifier is particularly easy for a Boolean expression in cnf:
  - ↗ Rule: Eliminate all occurences of $x, \bar{x}$
    - — Example: $f = (\bar{b} + \bar{c})(a + \bar{b})(\bar{a} + b + c)$
      $\forall a: f(a) = (\bar{b} + \bar{c})b(b + c) \quad (= \bar{b}c)$

- Inclusion $\leq$
  - ↗ A function f is included in (is covered by) function g,
    $f \leq g$
    iff (if and only if) $f \cdot \bar{g} = 0$ or equivalently $f \rightarrow g = 1$
  - ↗ Analogy (set theory):

$$F \subseteq G \Leftrightarrow F \cap \bar{G} = \emptyset$$



- Example of $f \leq g$ using Karnaugh-maps:

1			
		1	
	1	1	

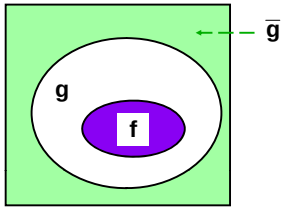
f

1			1
		1	
	1	1	1

g

- The following statements are equivalent:

$$f \leq g$$
$$f \cdot \overline{g} = 0$$
$$(f \rightarrow g = 1) \quad \overline{f} + g = 1$$
$$\overline{g} \leq \overline{f}$$
$$f + g = g$$
$$f \cdot g = f$$



- Assume (i) a function f in dnf and assume (ii) that g is the function f where some of the products of f are eliminated.
  Then $g \leq f$
  - Proof: f can be written in the form $f = a + g$.
    $g \leq f$ is equivalent to $g \rightarrow f = 1$ or $\overline{g} + f = 1$.
    $\overline{g} + (a + g) = \overline{g} + a + g = 1$.

products not in g

We have always: $a \leq a + g$.

- Assume (i) a function f in cnf and assume (ii) that g is the function f where some of the clauses are eliminated.
  Then $f \leq g$
  - Proof: f can be written in the form $f = a \cdot g$.
    $f \leq g$ is equivalent to $f \rightarrow g = 1$ or $\overline{f} + g = 1$.
    $\overline{(a \cdot g)} + g = \overline{a} + \overline{g} + g = 1$.

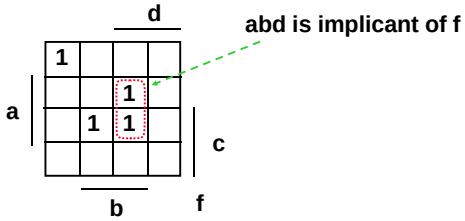
We have always: $a \cdot g \leq g$.

1. Boolean Algebra — 45

1.1 Boolean algebra basics

- **Implicant of a Boolean function**
  - A **product-term** p is an implicant of a Boolean function f iff

$$p \leq f$$

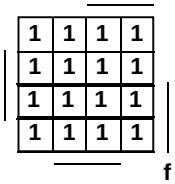


Concept of prime-implicants in two-level logic synthesis

1. Boolean Algebra — 46

1.1 Boolean algebra basics

- **Tautology:**
  - A Boolean function f is called a **Tautology** iff f = 1



— Is this a reasonable concept at all ???

— Examples:

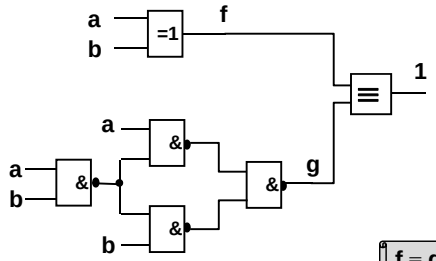
$$a + \bar{a} = 1$$

$$a + b + \bar{a} \cdot \bar{b} = 1$$

1. Boolean Algebra — 47

1.1 Boolean algebra basics

- **Logicverification**: show that two circuits with outputs f and g implement the same boolean function!
  - Show that $f \equiv g$ is a tautology!



$$f \equiv g \;=\; f*g + \bar{f}*\bar{g}$$

1. Boolean Algebra — 48

1.1 Boolean algebra basics

- Boole's expansion theorem is the basis for a "Divide-and-conquer" approach to tautology checking
  - — f = 1 iff both cofactors are equal to one

$$f = 1 \;\Rightarrow\; x \cdot f_x + \bar{x} \cdot f_{\bar{x}} = 1$$

$$\Rightarrow\; f_x = 1$$

$$\Rightarrow\; f_{\bar{x}} = 1$$

2 problems in n-1 variables

— Example: $f = a + b + \bar{a} \cdot \bar{b} = 1$?

$$f = a + b + \bar{a} \cdot \bar{b} = 1 \quad \begin{cases} f_a = 1 + b + \bar{1} \cdot \bar{b} = 1 \\ f_{\bar{a}} = 0 + b + \bar{0} \cdot \bar{b} = b + \bar{b} = 1 \end{cases}$$

Tautology checking may induce an exponential number of cases !

- Satisfiability: a Boolean function f is satisfiable, if there is at least one combination of variable values for which f = 1
  - A boolean function $f \neq 0$ is satisfiable!
  - f is a tautology iff $\bar{f}$ is unsatisfiable
    - — Example:

$$f = a + b + \bar{a} \cdot \bar{b} = 1 \quad ?$$

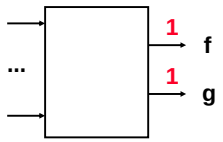
$$\bar{f} = \overline{(a + b + \bar{a} \cdot \bar{b})} =$$

$$= \bar{a} \cdot \bar{b} \cdot (a + b)$$

is not satisfiable

May also need an exponential # steps

- Example of application: assume a network of gates with two outputs f and g. If we want to know if both outputs can be 1 for the same combination of input values, then we can check if f·g is satisfiable.



- If f is a tautology and if $f \leq g$ then g is also a tautology
- If f is unsatifiable and if $g \leq f$ then g is also unsatisfiable

- German/Engl Glossary

$a, \bar{a}$	Literal	literal
$a + \bar{b} + \bar{c}$	Summenterm/Disjunktion/ Oder-Klausel	sum/disjunction/ clause
$ab\bar{c}$	Produktterm/Konjunktion/ Und-Klausel	product/conjunction/ cube
$ab + \bar{b}cd$	disjunktive Normalform (DNF)	disjunctive normal form (dnf)/ sum-of-products (sop)
$(a + b)(\bar{b} + c)$	konjunktive Normalform (KNF)	conjunctive normal form (cnf)/ product-of-sums
$f_{\bar{x}}$	0-/negativer Kofaktor	0-/negative cofactor
f_x	1-/positiver Kofaktor	1-/positive cofactor
$f = \bar{x}f_{\bar{x}} + xf_x$	Boolescher Entwicklungssatz	Boole's expansion theorem
$f = 1$	Tautologie	tautology
$f \neq 0$	Erfüllbarkeit	satisfiability

1.2 Reasoning About Boolean Expressions

- The verification of a circuit involves some procedures by which we give mathematical evidence that a statement about the circuit is true
  - A verification problem is not a circuit or a Booelan expression
  - A verification problem involves statements about a circuit or a Boolean expression
    - A Boolean expression is 0 or 1, a statement is true or false
    - "mathematical evidence" means that we are able to give something like a mathematical proof of the truth of some statement

- Typical statements involved in verification problems are:
  - "The outputs of two circuits are equal"
  - "It is not possible that the outputs x and y of a circuit both assume the value 1"
  - Implications: "If the inputs are ... then the outputs are ..." or: "If the outputs are ... then the inputs must be ..."
  - ... and many other types of statements involving, for instance, temporal qualifiers like "in the next clock-cycle", "never", "always", "sometimes", etc.

- Boolean expressions assume the Boolean values 0 or 1
- Statements (about a circuit) are either true or false, i.e., they hold or do not hold (for a circuit)
  - "it rains", "the colour of my car is red", "a+b=b+a", …
- A proof of a statement establishes the truth of the statement
- A proof-procedure is a method by which we can mechanically derive the truth of statements
- We are interested in the development of proof-procedures to verify statements about circuits

1. Boolean Algebra 57

1.2 Reasoning about Boolean expressions

- We employ common mathematical notations using symbols like "=", etc. to make statements
  - Example: "The outputs of two circuits are equal for all possible input combinations", formally: f = g (as in $\sin^2(x) + \cos^2(x) = 1$)
- We also used mathematical notations to introduce some basic concepts like "≤" in the former section
- The following paragraphs attempt to be a little bit more precise about symbols like "=" in order avoid confusion with the symbols of Boolean algebra
  - ✉ = and ≡ (or ⇒ and →) are so similar, what is the difference?
  - ✉ In fact, there IS a very close relationship between propositional logic and Boolean algebra

1. Boolean Algebra 58

1.2 Reasoning about Boolean expressions

- We employ two different types of notation with different symbols:
  - The language of Boolean expressions (or Boolean terms) with the Boolean operators ·, +, →, ≡, etc.
    - Boolean expressions assume the values 0 or 1.
  - A (mathematical) meta-language to make statements about Boolean expressions using symbols like =, ≤, ⇒, etc. which are well-known from other mathematical disciplines.
    - Statements are true or false.
    - Examples: a+b = b+a, a·g ≤ g, etc.

1. Boolean Algebra 59

1.2 Reasoning about Boolean expressions

- Again:
  - A Boolean expression may assume the values 0 or 1.
  - A statement may be true of false.
- The meta-language for statements has two different types of symbols:
  - Predicate-symbols like = and ≤. Applying a predicate to expressions results in a statement which is true or false.
    - Example: $\underbrace{\underbrace{x}_{\text{expression}} \leq \underbrace{x+1}_{\text{expression}}}_{\text{statement}}$
    - ✉ While functions and expressions return a value, predicates may be viewed as special functions returning the values true or false

1. Boolean Algebra 60

1.2 Reasoning about Boolean expressions

- Logical (propositional) connectives like ∧, ∨, ¬, ⇒ and ⇔. Logical connectives combine statements to statements.
  - Example: $\underbrace{\underbrace{x \leq x+1}_{\text{statement}} \wedge \underbrace{x = x+1}_{\text{statement}}}_{\text{statement}}$

$$\underbrace{\underbrace{x \leq y}_{\text{st.}} \wedge \underbrace{y \leq z}_{\text{st.}} \Rightarrow \underbrace{x \leq z}_{\text{st.}}}_{\text{statement}}$$

1. Boolean Algebra — 61

1.2 Reasoning about Boolean expressions

- The resulting truth-values of the logical connectives are defined analogously to the Boolean operators and are typically given by truth-tables.
  - Example: The truth-table of the propositional and "$\wedge$". Let A and B be two statements. Then the truth of $A \wedge B$ is defined as follows:

A \ B	false	true
false	false	false
true	false	true

$A \wedge B$

- We try to avoid an "over-formalization" of verification problems and will use logical connectives only rarely.

1. Boolean Algebra — 62

1.2 Reasoning about Boolean expressions

- Due to the close relationship between the Boolean operators and the operators of our mathematical meta-language, we can often transform a verification problem (phrased as a statement) into a different (often more managable) form.
  - Example: Rather than to prove f = g we show that $f \equiv g = 1$, i.e., that $f \equiv g$ is a tautology.

1. Boolean Algebra — 63

1.2 Reasoning about Boolean expressions

- Generally, we transform statements F and G involving Boolean expressions into equivalent Boolean expressions Be(F) and Be(G) as follows (f,g Boolean expressions):

Statement	Boolean expression
$f = 1$	f
$f = 0$	$\overline{f}$
$f = g$	$f \equiv g$
$f \leq g$	$f \rightarrow g$
$F \wedge G$	$Be(F) \cdot Be(g)$ etc.
$\neg F$	$\overline{Be(F)}$
$F \Rightarrow G$	$Be(F) \rightarrow Be(G)$

- The proof of a statement is equivalent to showing the tautology of the transformed Boolean expression, e.g., to prove f = g we show that $f \equiv g = 1$

1. Boolean Algebra — 64

1.2 Reasoning about Boolean expressions

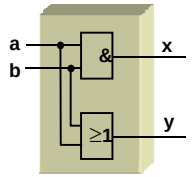
- Example: The statement "If x is 1 then y is 1 in the following circuit" is formalized as the statement

$$x = 1 \Rightarrow y = 1$$

The equivalent Boolean expression is

$$x \rightarrow y$$

The proof of the original statement is transformed into the proof of the tautology of $x \rightarrow y$, i.e., we have to prove the statement $x \rightarrow y = 1$



One possible proof:
$x \rightarrow y = \overline{x}+y = \overline{a \cdot b} + a+b$
$= \overline{a} + \overline{b} + a + b = 1$ (q.e.d.)

1. Boolean Algebra — 65

1.2 Reasoning about Boolean expressions

- A note on **implications**:
- In many situation, we have to prove the truth of an **if-then statement** in natural language: "if A then C" where the **commitment** C follows from another statement, the **assumption** A.
- Formally, this is denoted as an **implication** $A \Rightarrow C$.
- The definition of the logical implication $A \Rightarrow C$ in terms of a truth-table:

A \ C	false	true
false	true	true
true	false	true

$A \Rightarrow C$

$A \Rightarrow C = \neg A \vee C$
$a \rightarrow c = \bar{a} + c$

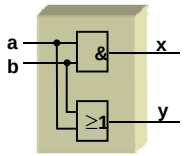
- If we want to prove $A \Rightarrow C$ then we see from the truth-table that for the case where A is false we do not have to prove anything - $A \Rightarrow C$ is true independently of C.

1. Boolean Algebra — 66

1.2 Reasoning about Boolean expressions

- All what we have to do is to prove that C is true for the case that A is true.
- We see this also if we transform the implication into the equivalent Boolean expression. Assume again the statement: $x = 1 \Rightarrow y = 1$ and the equivalent Boolean expression $x \rightarrow y$. If we want to show the tautology, i.e., $x \rightarrow y = 1$ and create the function table of $x \rightarrow y$:

x \ y	0	1
0	1	1
1	0	1



then we have to show only that y=1 follows from x=1.

- For the circuit, our argument would be: If x=1 then a=1 and b=1 since x=a·b. Thus, y=1 since y=a+b. (q.e.d.)

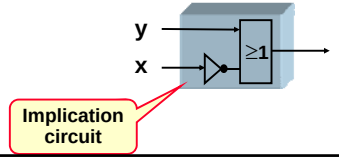
1. Boolean Algebra — 67

1.2 Reasoning about Boolean expressions

- There are many different ways ("decision procedures") to establish the truth of statements, the transformation into an equivalent Boolean expression is only one possibility.
- A very common technique is to create a circuit for the logical structure of the statement and combine this circuit with the original circuit.

  Since we are able to transform statements into Boolean expressions we can also transform statements into circuits!

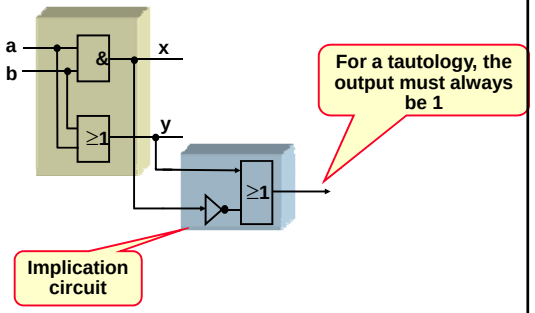
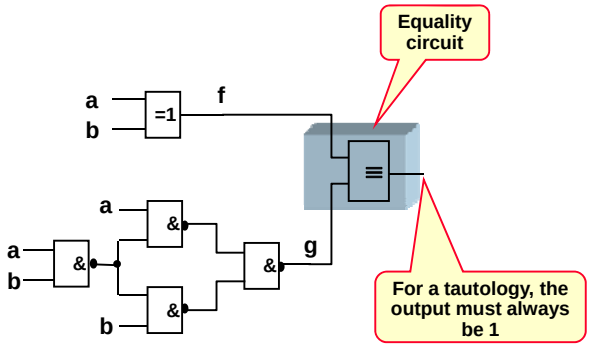
— Example: The implication circuit for $x \rightarrow y$



1. Boolean Algebra — 68

1.2 Reasoning about Boolean expressions

— Statement and original circuit in combination:



— Example we discussed before: Prove f = g