

► **Agenda:** an introduction to

1. **propositional logic**,
2. **Boolean algebra**, and
3. application of, i.e., use-cases and rationale for the above within the context of COMS10015.

## Part 1: propositional logic (1)

- ▶ A **proposition** is basically a statement

the temperature is  $20^{\circ}\text{C}$

this statement is false

the temperature is too hot

## Part 1: propositional logic (1)

- ▶ A **proposition** is basically a statement

the temperature is  $20^{\circ}\text{C}$

~~this statement is false~~

the temperature is too hot

whose meaning

1. can be **evaluated** to yield a **truth value**, i.e., **false** or **true**.

## Part 1: propositional logic (1)

- ▶ A **proposition** is basically a statement

the temperature is 20°C

~~this statement is false~~

~~the temperature is too hot~~

1. can be **evaluated** to yield a **truth value**, i.e., **false** or **true**,
2. must be unambiguous.

## Part 1: propositional logic (1)

- ▶ A **proposition** is basically a statement

the temperature is  $20^{\circ}\text{C}$

the temperature is  $x^{\circ}\text{C}$

~~this statement is false~~

~~the temperature is too hot~~

1. can be **evaluated** to yield a **truth value**, i.e., **false** or **true**,
2. must be unambiguous,
3. can include free **variables**.

## Part 1: propositional logic (1)

- ▶ A **proposition** is basically a statement

$f$  = the temperature is  $20^{\circ}\text{C}$   
 $g(x)$  = the temperature is  $x^{\circ}\text{C}$   
~~this statement is false~~  
~~the temperature is too hot~~

1. can be **evaluated** to yield a **truth value**, i.e., **false** or **true**,
2. must be unambiguous,
3. can include free **variables**, and
4. can be represented using a short-hand variable or function, whereby free variables must be bound to concrete arguments before evaluation.

## Part 1: propositional logic (2)

- ▶ Single statements can be combined using various **connectives**, e.g.,

the temperature is not  $20^{\circ}\text{C}$

adding parentheses where needed to add clarity, so that

1. “not  $x$ ” is denoted  $\neg x$ ,



## Part 1: propositional logic (2)

- ▶ Single statements can be combined using various **connectives**, e.g.,

$\neg(\text{the temperature is } 20^{\circ}\text{C})$

adding parentheses where needed to add clarity, so that

1. “not  $x$ ” is denoted  $\neg x$ ,

## Part 1: propositional logic (2)

- ▶ Single statements can be combined using various **connectives**, e.g.,

the temperature is  $20^{\circ}\text{C}$  and it is sunny

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,

## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

(the temperature is  $20^{\circ}\text{C}$ )  $\wedge$  (it is sunny)

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,

## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

the temperature is  $20^{\circ}\text{C}$  or it is sunny

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,
3. " $x$  or  $y$ " is denoted  $x \vee y$ , and usually called inclusive-or,

## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

(the temperature is  $20^{\circ}\text{C}$ )  $\vee$  (it is sunny)

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,
3. " $x$  or  $y$ " is denoted  $x \vee y$ , and usually called inclusive-or,

## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

either the temperature is  $20^{\circ}\text{C}$  or it is sunny, but not both

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,
3. " $x$  or  $y$ " is denoted  $x \vee y$ , and usually called inclusive-or,
4. " $x$  or  $y$  but not  $x$  and  $y$ " is denoted  $x \oplus y$ , and usually called exclusive-or,

## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

(the temperature is 20°C)  $\oplus$  (it is sunny)

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,
3. " $x$  or  $y$ " is denoted  $x \vee y$ , and usually called inclusive-or,
4. " $x$  or  $y$  but not  $x$  and  $y$ " is denoted  $x \oplus y$ , and usually called exclusive-or,

## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

the temperature being  $20^{\circ}\text{C}$  implies that it is sunny

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,
3. " $x$  or  $y$ " is denoted  $x \vee y$ , and usually called inclusive-or,
4. " $x$  or  $y$  but not  $x$  and  $y$ " is denoted  $x \oplus y$ , and usually called exclusive-or,
5. " $x$  implies  $y$ " is denoted  $x \Rightarrow y$ , and sometimes written "if  $x$  then  $y$ ", and



## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

(the temperature is 20°C)  $\Rightarrow$  (it is sunny)

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,
3. " $x$  or  $y$ " is denoted  $x \vee y$ , and usually called inclusive-or,
4. " $x$  or  $y$  but not  $x$  and  $y$ " is denoted  $x \oplus y$ , and usually called exclusive-or,
5. " $x$  implies  $y$ " is denoted  $x \Rightarrow y$ , and sometimes written "if  $x$  then  $y$ ", and

## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

the temperature is  $20^{\circ}\text{C}$  is equivalent to it being sunny

adding parentheses where needed to add clarity, so that

1. “not  $x$ ” is denoted  $\neg x$ ,
2. “ $x$  and  $y$ ” is denoted  $x \wedge y$ ,
3. “ $x$  or  $y$ ” is denoted  $x \vee y$ , and usually called inclusive-or,
4. “ $x$  or  $y$  but not  $x$  and  $y$ ” is denoted  $x \oplus y$ , and usually called exclusive-or,
5. “ $x$  implies  $y$ ” is denoted  $x \Rightarrow y$ , and sometimes written “if  $x$  then  $y$ ”, and
6. “ $x$  is equivalent to  $y$ ” is denoted  $x \equiv y$ , and sometimes written “ $x$  if and only if  $y$ ” or “ $x$  iff.  $y$ ”.

## Part 1: propositional logic (2)

- Single statements can be combined using various **connectives**, e.g.,

(the temperature is  $20^{\circ}\text{C}$ )  $\equiv$  (it is sunny)

adding parentheses where needed to add clarity, so that

1. "not  $x$ " is denoted  $\neg x$ ,
2. " $x$  and  $y$ " is denoted  $x \wedge y$ ,
3. " $x$  or  $y$ " is denoted  $x \vee y$ , and usually called inclusive-or,
4. " $x$  or  $y$  but not  $x$  and  $y$ " is denoted  $x \oplus y$ , and usually called exclusive-or,
5. " $x$  implies  $y$ " is denoted  $x \Rightarrow y$ , and sometimes written "if  $x$  then  $y$ ", and
6. " $x$  is equivalent to  $y$ " is denoted  $x \equiv y$ , and sometimes written " $x$  if and only if  $y$ " or " $x$  iff.  $y$ ".

## Part 1: propositional logic (2)

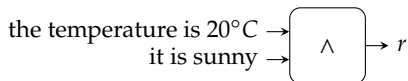
- ▶ You *might* see more formal terms or different notation for the *same* connectives:
  - ▶  $\neg$  is often termed logical **complement** (or **negation**),
  - ▶  $\wedge$  is often termed logical **conjunction**,
  - ▶  $\vee$  is often termed logical (inclusive) **disjunction**,
  - ▶  $\oplus$  is often termed logical (exclusive) **disjunction**,
  - ▶  $\Rightarrow$  is often termed logical **implication**, and
  - ▶  $\equiv$  is often termed logical **equivalence**.

## Part 1: propositional logic (3)

- You can think of the same thing diagrammatically, i.e.,

$$r = (\text{the temperature is } 20^{\circ}\text{C}) \wedge (\text{it is sunny})$$

$\equiv$



but either way, the question is how do we **evaluate** the (compound) proposition (or **expression**) to produce a truth value?

## Part 1: propositional logic (4)

- ▶ Since each statement can only evaluate to **true** or **false**, we can enumerate all possible outcomes in a **truth table**, e.g., if

$$\begin{aligned}x &= \text{the temperature is } 20^{\circ}\text{C} \\y &= \text{it is sunny} \\r &= (\text{the temperature is } 20^{\circ}\text{C}) \wedge (\text{it is sunny})\end{aligned}$$

then

inputs		output
$x$	$y$	$r$
false	false	false
false	true	false
true	false	false
true	true	true

- ▶ Note that
  1. each row details the output(s) associated with a given assignment to the inputs,
  2. if there are  $n$  inputs, the truth table will have  $2^n$  rows.

## Part 1: propositional logic (5)

### Definition

$x$	$y$	$\neg x$	$x \wedge y$	$x \vee y$	$x \oplus y$	$x \Rightarrow y$	$x \equiv y$
false	false	true	false	false	false	true	true
false	true	true	false	true	true	true	false
true	false	false	false	true	true	false	false
true	true	false	true	true	false	true	true

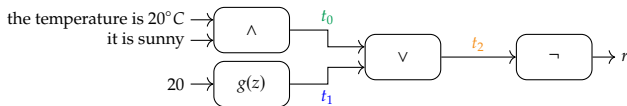
## Part 1: propositional logic (6)

### Example

Imagine that now

$x$  = the temperature is 20°C  
 $y$  = it is sunny  
 $g(z)$  = the temperature is z°C  
 $r$  =  $\neg(((\text{the temperature is 20°C}) \wedge (\text{it is sunny})) \vee (\text{the temperature is z°C}))$

which we translate into the diagrammatic form



An example evaluation might be as follows:

inputs		intermediates			output
$x$	$y$	$t_0$	$t_1$	$t_2$	$r$
false	false	false	false	false	true
false	true	false	false	false	true
true	false	false	true	true	false
true	true	true	true	true	false



## Part 2: Boolean algebra (1)

### ► Notice that

1. in **elementary algebra**, for some number  $x$  we have that

$$x + 0 = x$$

and

$$x \cdot 1 = x,$$

2. in **set theory**, for some set  $x$  we have that

$$x \cup \emptyset = x$$

and

$$x \cap \mathcal{U} = x,$$

plus we've now demonstrated that

3. in **propositional logic**, for some truth value  $x$  we have that

$$x \vee \mathbf{false} = x$$

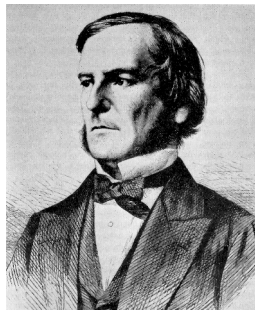
and

$$x \wedge \mathbf{true} = x.$$

## Part 2: Boolean algebra (2)

Thou must

1. work with the set  $\mathbb{B} = \{0, 1\}$  of **binary** digits, using 0 and 1 instead of **false** and **true**,
  2. shorten every statement into either a **variable** *or* **function**,
  3. use **unary operators**, e.g.,  $\neg$  (or NOT), and **binary operators**, e.g.,  $\wedge$  and  $\vee$  (or AND and OR), to form **expressions**,
  4. manipulate said expressions according to some axioms (or rules),
- then call the result **Boolean algebra**.



## Part 2: Boolean algebra (3)

► Put more concretely, we now have

1. a set of operators specified by

### Definition

$x$	$y$	$\neg x$	$x \wedge y$	$x \vee y$	$x \oplus y$	$x \Rightarrow y$	$x \equiv y$
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	0	0
1	1	0	1	1	0	1	1

## Part 2: Boolean algebra (3)

- Put more concretely, we now have

2. a set of axioms that allow manipulation of expressions comprised of said operators, i.e.,

### Definition

Name	Axiom(s)	Name	Axiom(s)
commutativity	$x \wedge y \equiv y \wedge x$	commutativity	$x \vee y \equiv y \vee x$
association	$(x \wedge y) \wedge z \equiv x \wedge (y \wedge z)$	association	$(x \vee y) \vee z \equiv x \vee (y \vee z)$
distribution	$x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$	distribution	$x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$

plus other rules such as **precedence** (to deal with ambiguity in the absence of parentheses).

## Part 2: Boolean algebra (3)

► Put more concretely, we now have

2. a set of axioms that allow manipulation of expressions comprised of said operators, i.e.,

### Definition

Name	Axiom(s)	Name	Axiom(s)
identity	$x \wedge 1 \equiv x$	identity	$x \vee 0 \equiv x$
null	$x \wedge 0 \equiv 0$	null	$x \vee 1 \equiv 1$
idempotency	$x \wedge x \equiv x$	idempotency	$x \vee x \equiv x$
inverse	$x \wedge \neg x \equiv 0$	inverse	$x \vee \neg x \equiv 1$

plus other rules such as **precedence** (to deal with ambiguity in the absence of parentheses).

## Part 2: Boolean algebra (3)

► Put more concretely, we now have

2. a set of axioms that allow manipulation of expressions comprised of said operators, i.e.,

### Definition

Name	Axiom(s)	Name	Axiom(s)
absorption de Morgan	$\begin{aligned}x \wedge (x \vee y) &\equiv x \\ \neg(x \wedge y) &\equiv \neg x \vee \neg y\end{aligned}$	absorption de Morgan	$\begin{aligned}x \vee (x \wedge y) &\equiv x \\ \neg(x \vee y) &\equiv \neg x \wedge \neg y\end{aligned}$

plus other rules such as **precedence** (to deal with ambiguity in the absence of parentheses).

## Part 2: Boolean algebra (3)

► Put more concretely, we now have

2. a set of axioms that allow manipulation of expressions comprised of said operators, i.e.,

### Definition

Name	Axiom(s)
equivalence	$x \equiv y \equiv (x \Rightarrow y) \wedge (y \Rightarrow x)$
implication	$x \Rightarrow y \equiv \neg x \vee y$
involution	$\neg \neg x \equiv x$

plus other rules such as **precedence** (to deal with ambiguity in the absence of parentheses).

## Part 2: Boolean algebra (4)

### Standard forms

#### Definition

The fact there are AND and OR forms of most axioms hints at a more general underlying principle. Consider a Boolean expression  $e$ : the **principle of duality** states that the **dual expression**  $e^D$  is formed by

1. leaving each variable as is,
2. swapping each  $\wedge$  with  $\vee$  and vice versa, and
3. swapping each 0 with 1 and vice versa.

Of course  $e$  and  $e^D$  are different expressions, and clearly not equivalent; if we start with some  $e \equiv f$  however, then we do still get  $e^D \equiv f^D$ .

#### Example

As an example, consider axioms for

1. distribution, e.g., if

$$e = x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$$

then

$$e^D = x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$$

and

2. identity, e.g., if

$$e = x \wedge 1 \equiv x$$

then

$$e^D = x \vee 0 \equiv x.$$



## Part 2: Boolean algebra (6)

### Standard forms

#### Definition

Consider a Boolean expression:

1. When the expression is written as a sum (i.e., OR) of terms which each comprise the product (i.e., AND) of variables, e.g.,

$$\underbrace{(a \wedge b \wedge c) \vee (d \wedge e \wedge f),}_{\text{minterm}}$$

it is said to be in **disjunctive normal form** or **Sum of Products (SoP)** form; the terms are called the **minterms**. Note that each variable can exist as-is *or* complemented using NOT, meaning

$$\underbrace{(\neg a \wedge b \wedge c) \vee (d \wedge \neg e \wedge f),}_{\text{minterm}}$$

is also a valid SoP expression.

2. When the expression is written as a product (i.e., AND) of terms which each comprise the sum (i.e., OR) of variables, e.g.,

$$\underbrace{(a \vee b \vee c) \wedge (d \vee e \vee f),}_{\text{maxterm}}$$

it is said to be in **conjunctive normal form** or **Product of Sums (PoS)** form; the terms are called the **maxterms**. As above each variable can exist as-is *or* complemented using NOT.

## Part 2: Boolean algebra (7)

### Derived operators

- **Concept:** we can define various **derived operators** in terms of NOT, AND, and OR.
- **Example:**
  - “exclusive-OR” or **XOR**, such that

$$x \oplus y \equiv (\neg x \wedge y) \vee (x \wedge \neg y)$$

so

$x$	$y$	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

## Part 2: Boolean algebra (7)

### Derived operators

► **Concept:** we can define various **derived operators** in terms of NOT, AND, and OR.

► **Example:**

► “NOT-AND” or **NAND**, such that

$$x \overline{\wedge} y \equiv \neg(x \wedge y)$$

so

$x$	$y$	$x \overline{\wedge} y$
0	0	1
0	1	1
1	0	1
1	1	0

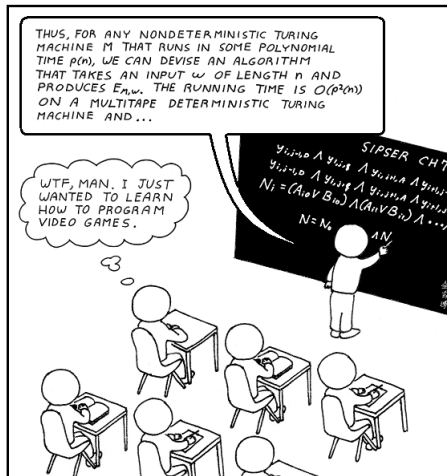
► “NOT-OR” or **NOR**, such that

$$x \overline{\vee} y \equiv \neg(x \vee y)$$

so

$x$	$y$	$x \overline{\vee} y$
0	0	1
0	1	0
1	0	0
1	1	0

## Part 3: application (1)



## Part 3: application (2)

► (Fairly) reasonable question(s):

1. "I thought this was CS, not Maths!", and
2. "why does *this* unit duplicate material in *other* units?".

## Part 3: application (2)

- ▶ (Fairly) reasonable **question(s)**:
  1. “I thought this was CS, not Maths!”, and
  2. “why does *this* unit duplicate material in *other* units?”.
- ▶ **Answer**: it isn’t, and it doesn’t (well, not *too* much) ... note that
  - ▶ theoretical concepts, e.g., often have significant practical motivations or implications, and
  - ▶ it’s perfectly reasonable to utilise **Electronic Design Automation (EDA)** [3] tools.

## Part 3: application (3)

Axiomatic manipulation  $\leadsto$  optimisation

- **Question:** simplify the Boolean expression

$$(\neg(a \vee b) \wedge \neg(c \vee d \vee e)) \vee \neg(a \vee b)$$

into a form that contains the fewest operators possible.

## Part 3: application (3)

Axiomatic manipulation  $\leadsto$  optimisation

- **Question:** simplify the Boolean expression

$$(\neg(a \vee b) \wedge \neg(c \vee d \vee e)) \vee \neg(a \vee b)$$

into a form that contains the fewest operators possible.

- **Solution #1:** less steps.

$$\begin{aligned} & (\neg(a \vee b) \wedge \neg(c \vee d \vee e)) \vee \neg(a \vee b) \\ = & \neg(a \vee b) \vee (\neg(a \vee b) \wedge \neg(c \vee d \vee e)) && \text{(commutativity)} \\ = & \neg(a \vee b) && \text{(absorption)} \end{aligned}$$



## Part 3: application (3)

Axiomatic manipulation  $\rightsquigarrow$  optimisation

- **Question:** simplify the Boolean expression

$$(\neg(a \vee b) \wedge \neg(c \vee d \vee e)) \vee \neg(a \vee b)$$

into a form that contains the fewest operators possible.

- **Solution #2:** more steps.

$$\begin{aligned} & (\neg(a \vee b) \wedge \neg(c \vee d \vee e)) \vee \neg(a \vee b) \\ = & ((\neg a \wedge \neg b) \wedge \neg(c \vee d \vee e)) \vee \neg(a \vee b) && \text{(de Morgan)} \\ = & ((\neg a \wedge \neg b) \wedge (\neg c \wedge \neg d \wedge \neg e)) \vee \neg(a \vee b) && \text{(de Morgan)} \\ = & ((\neg a \wedge \neg b) \wedge (\neg c \wedge \neg d \wedge \neg e)) \vee (\neg a \wedge \neg b) && \text{(de Morgan)} \\ = & (\neg a \wedge \neg b) \vee ((\neg a \wedge \neg b) \wedge (\neg c \wedge \neg d \wedge \neg e)) && \text{(commutativity)} \\ = & (\neg a \wedge \neg b) && \text{(absorption)} \\ = & \neg(a \vee b) && \text{(de Morgan)} \end{aligned}$$

## Part 3: application (4)

Axiomatic manipulation  $\leadsto$  optimisation

- **Question:** simplify the Boolean expression

$$(a \wedge b \wedge c) \vee (\neg a \wedge b) \vee (a \wedge b \wedge \neg c)$$

into a form that contains the fewest operators possible.

## Part 3: application (4)

Axiomatic manipulation  $\leadsto$  optimisation

- **Question:** simplify the Boolean expression

$$(a \wedge b \wedge c) \vee (\neg a \wedge b) \vee (a \wedge b \wedge \neg c)$$

into a form that contains the fewest operators possible.

- **Solution:**

$$\begin{aligned} & (a \wedge b \wedge c) \vee (\neg a \wedge b) \vee (a \wedge b \wedge \neg c) \\ = & (a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c) \vee (\neg a \wedge b) && \text{(commutativity)} \\ = & (a \wedge b) \wedge (c \vee \neg c) \vee (\neg a \wedge b) && \text{(distribution)} \\ = & (a \wedge b) \wedge 1 \vee (\neg a \wedge b) && \text{(inverse)} \\ = & (a \wedge b) \vee (\neg a \wedge b) && \text{(identity)} \\ = & b \wedge (a \vee \neg a) && \text{(distribution)} \\ = & b \wedge 1 && \text{(inverse)} \\ = & b && \text{(identity)} \end{aligned}$$

## Part 3: application (5)

Axiomatic manipulation  $\rightsquigarrow$  optimisation

### Quote

*If I designed a computer with 200 chips, I tried to design it with 150. And then I would try to design it with 100. I just tried to find every trick I could in life to design things real tiny.*

– Wozniak

### Quote

*So I took 20 chips off their board; I bypassed 20 of their chips.*

– Wozniak

## Part 3: application (6)

Axiomatic manipulation  $\leadsto$  optimisation



[https://en.wikipedia.org/wiki/File:Shugart\\_SA400.jpg](https://en.wikipedia.org/wiki/File:Shugart_SA400.jpg)

[https://en.wikipedia.org/wiki/File:Interface\\_Card\\_-\\_Disk\\_II\\_Interface\\_Apple2.jpg](https://en.wikipedia.org/wiki/File:Interface_Card_-_Disk_II_Interface_Apple2.jpg)

## Part 3: application (7)

Practical use-cases  $\leadsto$  richer specification

- **Concept:** truth tables can accommodate **don't care** entries, e.g.,

$x$	$y$	$r$
?	0	1
0	1	?
1	1	0

such that

- a ? (rather than 0 or 1) means we “don't care” ( $\neq$  “don't know”),
- on the LHS, for an *input*,
  - ? is a wildcard (or short-hand),
  - it means 0 *and* 1,
  - we've compressed two truth table rows into one.
- on the RHS, for an *output*,
  - ? is a choice,
  - it means 0 *or* 1,
  - we can select which one to, e.g., optimise the associated expression.

- **Fact:** NAND and NOR are **functionally complete** (or **universal**), e.g.,

$$\begin{aligned}\neg x &\equiv x \bar{\wedge} x \\ x \wedge y &\equiv (x \bar{\wedge} y) \bar{\wedge} (x \bar{\wedge} y) \\ x \vee y &\equiv \neg x \bar{\wedge} \neg y \equiv (x \bar{\wedge} x) \bar{\wedge} (y \bar{\wedge} y)\end{aligned}$$

which we can prove via

$x$	$y$	$x \bar{\wedge} y$	$x \bar{\wedge} x$	$y \bar{\wedge} y$	$(x \bar{\wedge} y) \bar{\wedge} (x \bar{\wedge} y)$	$(x \bar{\wedge} x) \bar{\wedge} (y \bar{\wedge} y)$
0	0	1	1	1	0	0
0	1	1	1	0	0	1
1	0	1	0	1	0	1
1	1	0	0	0	1	1

$\therefore$  any Boolean function can be expressed using a *single* operator.

► **Question:** translate

$$x \wedge (y \vee z)$$

into a version using NAND only.



- **Question:** translate

$$x \wedge (y \vee z)$$

into a version using NAND only.

- **Solution #1:** apply the identities *naively* to get

$$\begin{aligned} & x \wedge (y \vee z) \\ = & x \wedge ((y \overline{\wedge} y) \overline{\wedge} (z \overline{\wedge} z)) \\ = & (x \overline{\wedge} ((y \overline{\wedge} y) \overline{\wedge} (z \overline{\wedge} z))) \overline{\wedge} (x \overline{\wedge} ((y \overline{\wedge} y) \overline{\wedge} (z \overline{\wedge} z))) \end{aligned}$$

- **Question:** translate

$$x \wedge (y \vee z)$$

into a version using NAND only.

- **Solution #2:** apply the identities *intelligently* to get

$$\begin{aligned} & x \wedge (y \vee z) \\ = & x \wedge ((y \overline{\wedge} y) \overline{\wedge} (z \overline{\wedge} z)) \\ = & t \overline{\wedge} t \end{aligned}$$

where  $t = x \overline{\wedge} ((y \overline{\wedge} y) \overline{\wedge} (z \overline{\wedge} z))$  is a common sub-expression [2].

# Conclusions

## ► Take away points:

1. The design of computational devices, e.g., micro-processors, *isn't* ad hoc: Boolean algebra offers a theoretical basis for reasoning about computational devices (and computation) in practice.

## ► Take away points:

2. Boolean algebra is a (somewhat) cosmetic extension of what you already know.
3. Keep in mind that
  - *any* Boolean function  $f$  which can be expressed by a truth table can be computed using an associated Boolean expression,
  - a Boolean expression is composed of Boolean operators,
  - *if* we (physically) implement the Boolean operators, we can implement the Boolean expression and hence compute  $f$ .

## ► Take away points:

4. We'll focus on *application* (i.e., *use*) vs. *theory* (e.g., *study*) of Boolean algebra from here on.
5. Keep in mind that
  - “it works”  $\neq$  “it works *well*”,
  - using automation is fine *iff.* you know the underlying theory,
  - using brute-force is fine *iff.* you know the underlying theory,
  - Boolean algebra > Boolean axioms: concepts that *seem* of interest in theory alone, *can* be important if/when applied in practice.

# Additional Reading

- ▶ *Wikipedia: Boolean algebra*. URL: [https://en.wikipedia.org/wiki/Boolean\\_algebra](https://en.wikipedia.org/wiki/Boolean_algebra).
- ▶ D. Page. “Chapter 1: Mathematical preliminaries”. In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009.
- ▶ W. Stallings. “Chapter 11: Digital logic”. In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013.
- ▶ A.S. Tanenbaum and T. Austin. “Section 3.1: Gates and Boolean algebra”. In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012.

# References

- [1] *Wikipedia: Boolean algebra*. URL: [https://en.wikipedia.org/wiki/Boolean\\_algebra](https://en.wikipedia.org/wiki/Boolean_algebra) (see p. 54).
- [2] *Wikipedia: Common sub-expression elimination*. URL: [https://en.wikipedia.org/wiki/Common\\_subexpression\\_elimination](https://en.wikipedia.org/wiki/Common_subexpression_elimination) (see pp. 49, 50).
- [3] *Wikipedia: Electronic Design Automation (EDA)*. URL: [https://en.wikipedia.org/wiki/Electronic\\_design\\_automation](https://en.wikipedia.org/wiki/Electronic_design_automation) (see pp. 37, 38).
- [4] D. Page. “Chapter 1: Mathematical preliminaries”. In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009 (see p. 54).
- [5] W. Stallings. “Chapter 11: Digital logic”. In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013 (see p. 54).
- [6] A.S. Tanenbaum and T. Austin. “Section 3.1: Gates and Boolean algebra”. In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012 (see p. 54).