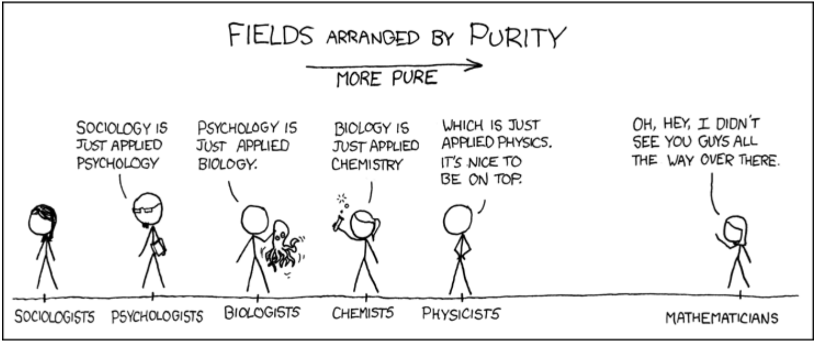


---

http://xkcd.com/435/

- Modern computing devices *aren't* ad hoc constructions; a rich theory underpins their design and operation.
- Focusing on computer architecture specifically, **Boolean algebra** is central to more or less *everything*:
  1. in 1840*s* Boole unified concepts in logic and set theory, predating what we now know as **abstract algebra**, which then
  2. enabled Shannon to design analyse and design electrical circuits via logic gates in seminal 1937*s* work.

Propositional Logic (1)

- A **proposition** is basically a statement

  the temperature is 20°C

  this statement is false
  the temperature is too hot

  whose meaning

Propositional Logic (1)

- A **proposition** is basically a statement

  the temperature is $20°C$

  ~~this statement is false~~
  the temperature is too hot

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

Propositional Logic (1)

- A **proposition** is basically a statement

  the temperature is $20°C$

  ~~this statement is false~~
  ~~the temperature is too hot~~

  whose meaning

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

Propositional Logic (1)

- A **proposition** is basically a statement

  the temperature is $20°C$
  the temperature is $x°C$
  ~~this statement is false~~
  ~~the temperature is too hot~~

  whose meaning

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

Propositional Logic (1)

- A **proposition** is basically a statement

$$
\begin{array}{lcl}
f & = & \text{the temperature is } 20°C \\
g(x) & = & \text{the temperature is } x°C \\
 & & \cancel{\text{this statement is false}} \\
 & & \cancel{\text{the temperature is too hot}}
\end{array}
$$

  whose meaning

  1. can be **evaluated** to give 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.

Propositional Logic (2)

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

  the temperature is not $20°C$

  adding parentheses where needed to add clarity, so that

  1. "not x" is denoted $\neg x$,

Propositional Logic (2)

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

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

  adding parentheses where needed to add clarity, so that
  1. "not x" is denoted $\neg x$,

Propositional Logic (2)

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

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

  adding parentheses where needed to add clarity, so that

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

Propositional Logic (2)

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

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

adding parentheses where needed to add clarity, so that

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

Propositional Logic (2)

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

  the temperature is $20°C$ 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,

Propositional Logic (2)

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

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

adding parentheses where needed to add clarity, so that

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

Propositional Logic (2)

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

  either the temperature is $20°C$ 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,

Propositional Logic (2)

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

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

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,

Propositional Logic (2)

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

  the temperature being $20°C$ 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

Propositional Logic (2)

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

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

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

Propositional Logic (2)

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

  the temperature is $20°C$ 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".

Propositional Logic (2)

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

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

  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".

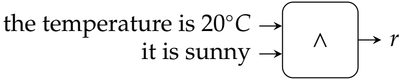
- You might also hear more formal terms for these connectives:
  - $\neg$ is often termed logical **compliment** (or **negation**),
  - $\wedge$ is often termed logical **conjunction**,
  - $\vee$ is often termed logical (inclusive) **disjunction**,
  - $\oplus$ is often termed logical (exclusive) **disjunction**,
  - $\Rightarrow$ is often termed logical **implication**, and
  - $\equiv$ is often termed logical **equivalence**.

Propositional Logic (3)

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

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

$$\equiv$$



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

Propositional Logic (4)

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

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

then

inputs		output
x	y	r
false	**false**	**false**
false	**true**	**false**
true	**false**	**false**
true	**true**	**true**

- With n inputs, the truth table will have 2^n rows: each row details the output(s) associated with a given assignment to the inputs.

Propositional Logic (5)

Definition (**logical connectives**)

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**

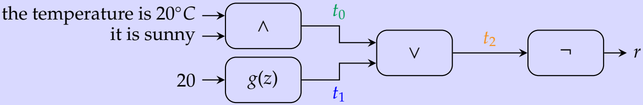
Propositional Logic (6)

Example

Imagine that now

$$
\begin{array}{lcl}
x & = & \text{the temperature is } 20^{\circ}C \\
y & = & \text{it is sunny} \\
g(z) & = & \text{the temperature is } z^{\circ}C \\
r & = & \neg(((\text{the temperature is } 20^{\circ}C) \wedge (\text{it is sunny})) \vee (\text{the temperature is } z^{\circ}C))
\end{array}
$$

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**

Boolean Algebra (1)

- If you look closely, some commonalities between propositional logic and *other* concepts in Mathematics start to emerge ...
- ... among many possibilities, consider *one* example:

1. In **propositional logic**, for some truth value x we have that

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

   and

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

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

$$x \cup \emptyset = x$$

   and

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

3. In **elementary algebra**, for some number x we have that

$$x + 0 = x$$

   and

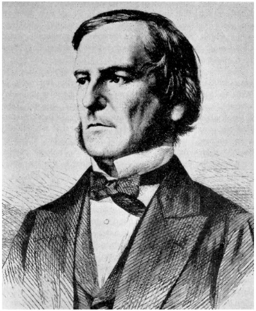
$$x \cdot 1 = x.$$

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 the unary **operator** $\neg$ (or NOT) and the binary **operators** $\wedge$, $\vee$ and $\oplus$ (or AND, OR and XOR) to form **expressions**,
4. manipulate said expressions according to some axioms (or rules)

then call the result **Boolean algebra**.



http://en.wikipedia.org/wiki/File:George_Boole.jpg

Boolean Algebra (3)

- One benefit of Boolean axiomatisation is that we can now manipulate expressions *without* evaluating them, e.g., via

Definition

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

plus some others, such as **precedence** to deal with any ambiguity in the absence of parentheses.

- Why?!
  1. to prove equivalence (without brute-force enumeration), or
  2. to perform simplification (e.g., less operators).

Boolean Algebra (3)

- One benefit of Boolean axiomatisation is that we can now manipulate expressions *without* evaluating them, e.g., via

Definition

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

plus some others, such as **precedence** to deal with any ambiguity in the absence of parentheses.

- Why?!
  1. to prove equivalence (without brute-force enumeration), or
  2. to perform simplification (e.g., less operators).

Boolean Algebra (3)

- One benefit of Boolean axiomatisation is that we can now manipulate expressions *without* evaluating them, e.g., via

Definition

Name	*Axiom(s)*					
absorption	$x \wedge (x \vee y)$	$\equiv$	x	$x \vee (x \wedge y)$	$\equiv$	x
de Morgan	$\neg(x \wedge y)$	$\equiv$	$\neg x \vee \neg y$	$\neg(x \vee y)$	$\equiv$	$\neg x \wedge \neg y$

plus some others, such as **precedence** to deal with any ambiguity in the absence of parentheses.

- Why?!
  1. to prove equivalence (without brute-force enumeration), or
  2. to perform simplification (e.g., less operators).

Boolean Algebra (3)

- One benefit of Boolean axiomatisation is that we can now manipulate expressions *without* evaluating them, e.g., via

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 some others, such as **precedence** to deal with any ambiguity in the absence of parentheses.
- Why?!
  1. to prove equivalence (without brute-force enumeration), or
  2. to perform simplification (e.g., less operators).

Boolean Algebra (4)

Definition (**principle of duality**)

The fact there are AND and OR forms of most axioms hints at a more general underlying principle. Consider a Boolean expression e: 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

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

Boolean Algebra (5)

Definition (**principle of complements**)

The de Morgan axiom can be turned into a more general principle. Consider a Boolean expression e: the **complement** expression $\neg e$ is formed by

1. swapping each variable x with the complement $\neg x$,
2. swapping each $\wedge$ with $\vee$ and vice versa, and
3. swapping each 0 with 1 and vice versa.

Example

Consider that if

$$e = x \wedge y \wedge z,$$

then by the above we should find

$$f = \neg e = (\neg x) \vee (\neg y) \vee (\neg z).$$

Proof:

x	y	z	$\neg x$	$\neg y$	$\neg z$	e	f
0	0	0	1	1	1	0	1
0	0	1	1	1	0	0	1
0	1	0	1	0	1	0	1
0	1	1	1	0	0	0	1
1	0	0	0	1	1	0	1
1	0	1	0	1	0	0	1
1	1	0	0	0	1	0	1
1	1	1	0	0	0	1	0

Boolean Algebra (6)

Definition (**standard forms**)

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.,

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

   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

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

   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.,

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

   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.

Conclusions

- Take away points:
  1. In essence, Boolean algebra is a (somewhat) cosmetic extension of what you already know: however, keep in mind that
     - *any* Boolean function f which can be expressed by a truth table can be computed using a Boolean expression, so
     - if we can construct *physical* implementations of NOT, AND and OR we can build something to actually compute f.
  2. So the pay-off for understanding Boolean algebra is that *it* explains how computation (and hence computers) work in practice.
  3. The point is that from here on is that you just learn
     - the truth tables for each Boolean operator, and
     - the set of axioms for Boolean algebra,

     and focus on *applying* the theory to practical challenges and tasks, rather than the theory itself.

References and Further Reading

[1] Wikipedia: Boolean algebra.
http://en.wikipedia.org/wiki/Boolean_algebra.

[2] Wikipedia: Claude Shannon.
http://en.wikipedia.org/wiki/Claude_Shannon.

[3] Wikipedia: George Boole.
http://en.wikipedia.org/wiki/George_Boole.

[4] D. Page.
Chapter 1: Mathematical preliminaries.
In *A Practical Introduction to Computer Architecture*. Springer-Verlag, 1st edition, 2009.

[5] W. Stallings.
Chapter 11: Digital logic.
In *Computer Organisation and Architecture*. Prentice-Hall, 9th edition, 2013.

[6] A.S. Tanenbaum.
Section 3.1: Gates and Boolean algebra.
In *Structured Computer Organisation*. Prentice-Hall, 6th edition, 2012.