Computer Architecture

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
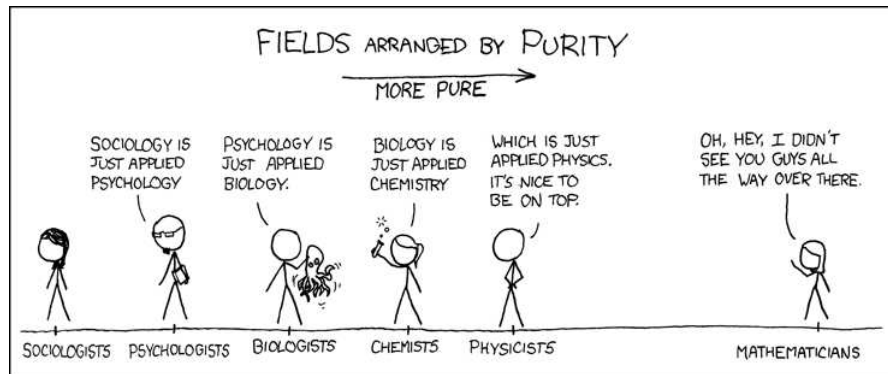⟨csdsp@bristol.ac.uk⟩

September 5, 2025

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and

2. a PDF of non-examinable, extra material:

   ▶ the associated notes page may be pre-populated with extra, written explaination of
     material covered in lecture(s), plus
   ▶ anything with a "grey'ed out" header/footer represents extra material which is
     useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

FIELDS ARRANGED BY PURITY

MORE PURE →

SOCIOLOGY IS JUST APPLIED PSYCHOLOGY

PSYCHOLOGY IS JUST APPLIED BIOLOGY.

BIOLOGY IS JUST APPLIED CHEMISTRY

WHICH IS JUST APPLIED PHYSICS. IT'S NICE TO BE ON TOP.

OH, HEY, I DIDN'T SEE YOU GUYS ALL THE WAY OVER THERE.

SOCIOLOGISTS   PSYCHOLOGISTS   BIOLOGISTS   CHEMISTS   PHYSICISTS   MATHEMATICIANS

https://xkcd.com/435

University of BRISTOL

Notes:

► 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.

University of BRISTOL

Notes:

▶ A **proposition** is basically a statement

the temperature is $20°C$

this statement is false
the temperature is too hot

▶ 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 yield a **truth value**, i.e., **false** or **true**.

Part 1: propositional logic (1)

▶ A **proposition** is basically a statement

<div align="center">

the temperature is $20°C$

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

</div>

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

<div align="center">

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

</div>

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

$$
\begin{array}{rcl}
f & = & \text{the temperature is } 20^\circ C \\
g(x) & = & \text{the temperature is } x^\circ C \\
& & \text{\sout{this statement is false}} \\
& & \text{\sout{the temperature is too hot}}
\end{array}
$$

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.

Notes:

Part 1: propositional logic (2)

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

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

adding parentheses where needed to add clarity, so that

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

Notes:

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

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

adding parentheses where needed to add clarity, so that

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

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

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

adding parentheses where needed to add clarity, so that

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

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

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

adding parentheses where needed to add clarity, so that

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

Notes:

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

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

adding parentheses where needed to add clarity, so that

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

Notes:

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

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

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,

Notes:

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

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

adding parentheses where needed to add clarity, so that

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

Notes:

▶ 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,

Notes:

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

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

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

Notes:

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

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

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

Notes:

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

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

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".

Notes:

▶ 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".

Notes:

▶ You *might* see more formal terms or different notation for the *same* 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
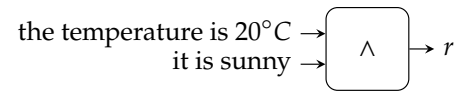▶ $\equiv$ is often termed logical **equivalence**.

Notes:

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

$$r = (\text{the temperature is } 20°C) \land (\text{it is sunny})$$

$$\equiv$$

the temperature is $20°C$ → [∧] → r
it is sunny →

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

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

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

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.

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

Notes:

Example

Imagine that now

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

which we translate into the diagrammatic form
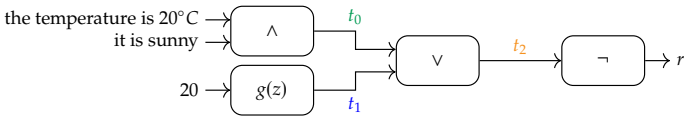
An example evaluation might be as follows:

	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

Notes:

► 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
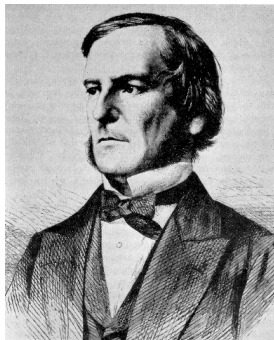
$$x \vee \textbf{false} = x$$

and

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

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., ¬ (or NOT), and **binary operators**, e.g., ∧ and ∨ (or AND and OR), to form **expressions**,

4. manipulate said expressions according to some axioms (or rules),

then call the result **Boolean algebra**.

- ► 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

University of BRISTOL

Notes:
- The precedence levels for our suite of Boolean operators is
  1. $\neg$,
  2. $\wedge$,
  3. $\vee$

  meaning, for example, that we resolve an $\wedge$ before and $\vee$ (and sometimes say $\wedge$ "binds more tightly" to operands than $\vee$).

- ► 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).

University of BRISTOL

Notes:
- The precedence levels for our suite of Boolean operators is
  1. $\neg$,
  2. $\wedge$,
  3. $\vee$

  meaning, for example, that we resolve an $\wedge$ before and $\vee$ (and sometimes say $\wedge$ "binds more tightly" to operands than $\vee$).

▶ 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).

Notes:
- The precedence levels for our suite of Boolean operators is
  1. $\neg$,
  2. $\wedge$,
  3. $\vee$

  meaning, for example, that we resolve an $\wedge$ before and $\vee$ (and sometimes say $\wedge$ "binds more tightly" to operands than $\vee$).

Definition

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

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

▶ 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	$x \equiv y \quad \equiv \quad (x \Rightarrow y) \land (y \Rightarrow x)$
implication	$x \Rightarrow y \quad \equiv \quad \neg x \lor y$
involution	$\neg\neg x \quad \equiv \quad x$

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

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 ∧ with ∨ 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

$$e = x \land (y \lor z) \equiv (x \land y) \lor (x \land z)$$

   then

$$e^D = x \lor (y \land z) \equiv (x \lor y) \land (x \lor z)$$

   and

2. identity, e.g., if

$$e = x \land 1 \equiv x$$

   then

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

Definition

The de Morgan axiom can be turned into a more general principle. Consider a Boolean expression e: the **principle of complements** states that 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

As an 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

University of BRISTOL
git # b282dbb9 @ 2025-09-03

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.,

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

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.

University of BRISTOL
git # b282dbb9 @ 2025-09-03

Notes:

Notes:

▶ 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

▶ 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

Notes:

Notes:

▶ (Fairly) reasonable question(s):
  1. "I thought this was CS, not Maths!", and
  2. "why does *this* unit duplicate material in *other* units?".

▶ (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.

Notes:

Part 3: application (3)
Axiomatic manipulation ⤳ 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.

Notes:

▶ 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.

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

Notes:

▶ 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.

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

Notes:

▶ Question: simplify the Boolean expression

$$(a \land b \land c) \lor (\neg a \land b) \lor (a \land b \land \neg c)$$

into a form that contains the fewest operators possible.

▶ Question: simplify the Boolean expression

$$(a \land b \land c) \lor (\neg a \land b) \lor (a \land b \land \neg c)$$

into a form that contains the fewest operators possible.

▶ Solution:

$$
\begin{array}{llllll}
 & (a \land b \land c) & \lor & (\neg a \land b) & \lor & (a \land b \land \neg c) \\
= & (a \land b \land c) & \lor & (a \land b \land \neg c) & \lor & (\neg a \land b) & \text{(commutativity)} \\
= & (a \land b) & \land & (c \lor \neg c) & \lor & (\neg a \land b) & \text{(distribution)} \\
= & (a \land b) & \land & 1 & \lor & (\neg a \land b) & \text{(inverse)} \\
= & (a \land b) & & & \lor & (\neg a \land b) & \text{(identity)} \\
= & b & \land & (a \lor \neg a) & & & \text{(distribution)} \\
= & b & \land & 1 & & & \text{(inverse)} \\
= & b & & & & & \text{(identity)}
\end{array}
$$

> ## 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*

Notes:

- The quotes relate to design and implementation of a (floppy) disk controller for the Apple II computer (circa 1977); there is an obvious focus on efficiency, which is credited as allowing the controller to be commercially viable. A detailed overview of the overarching anecdote is available via

  https://en.wikipedia.org/wiki/Disk_II

  or

  https://apple2history.org/history/ah05/

  The moral is that, in reality, "it works", while important, may not be good enough: meeting various other (market-driven) quality metrics (e.g., efficiency, physical size, power consumption, etc.) is often vital rather than simply attractive.

Shugart SA400 minifloppy Disk Drive

https://en.wikipedia.org/wiki/File:Shugart_SA400.jpg
https://en.wikipedia.org/wiki/File:Interface_Card_-_Disk_II_Interface_Apple2.jpg

Notes:

▶ 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" (≠ "don't know"),
▶ on the LHS, for an *in*put,
  ▶ **?** 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 *out*put,
  ▶ **?** is a choice,
  ▶ it means 0 *or* 1,
  ▶ we can select which one to, e.g., optimise the associated expression.

Notes:

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

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

which we can prove via

x	y	$x \mathbin{\overline{\wedge}} y$	$x \mathbin{\overline{\wedge}} x$	$y \mathbin{\overline{\wedge}} y$	$(x \mathbin{\overline{\wedge}} y) \mathbin{\overline{\wedge}} (x \mathbin{\overline{\wedge}} y)$	$(x \mathbin{\overline{\wedge}} x) \mathbin{\overline{\wedge}} (y \mathbin{\overline{\wedge}} y)$
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

∴ *any* Boolean function can be expressed using a *single* operator.

Notes:

▶ Question: translate

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

into a version using NAND only.

Notes:

▶ Question: translate

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

into a version using NAND only.

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

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

Notes:

▶ 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.

Notes:

Notes:

Conclusions

- 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.

Notes:

Conclusions

- 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" ≠ "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.

Notes:

Additional Reading

- *Wikipedia: Boolean algebra*. URL: `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.

Notes:

References

[1] *Wikipedia: Boolean algebra*. URL: `https://en.wikipedia.org/wiki/Boolean_algebra` (see p. 113).

[2] *Wikipedia: Common sub-expression elimination*. URL: `https://en.wikipedia.org/wiki/Common_subexpression_elimination` (see pp. 103, 105).

[3] *Wikipedia: Electronic Design Automation (EDA)*. URL: `https://en.wikipedia.org/wiki/Electronic_design_automation` (see pp. 79, 81).

[4] D. Page. "Chapter 1: Mathematical preliminaries". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009 (see p. 113).

[5] W. Stallings. "Chapter 11: Digital logic". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013 (see p. 113).

[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. 113).

Notes: