Intro. to Computer Architecture

Daniel Page

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

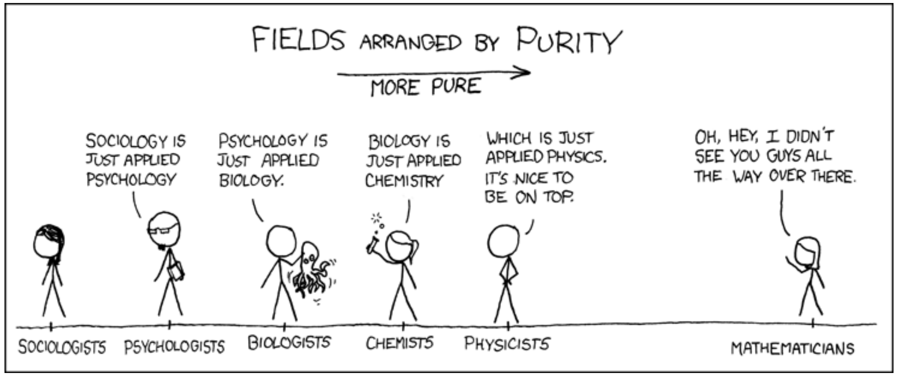
January 9, 2018

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:



http://xkcd.com/435/

Notes:

- 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 1840s Boole unified concepts in logic and set theory, predating what we now know as **abstract algebra**, which then
  2. enabled Shannon to design and analyse electrical circuits via logic gates in seminal 1937*s* work.

Notes:

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

Notes:

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**,

Notes:

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,

Notes:

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

Notes:

Propositional Logic (1)

- A **proposition** is basically a statement

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

  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.

Notes:

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:

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 $\neg x$,

Notes:

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 $x \wedge y$,

Notes:

Propositional Logic (2)

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

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

adding parentheses where needed to add clarity, so that

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

Notes:

Propositional Logic (2)

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

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

adding parentheses where needed to add clarity, so that

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

Notes:

Propositional Logic (2)

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

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

  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,

Notes:

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,

Notes:

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Intro. to Computer Architecture
git # 3b6f641 @ 2018-01-09
University of BRISTOL

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,

Notes:

Propositional Logic (2)

- 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:

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

Notes:

Propositional Logic (2)

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

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

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:

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".

Notes:

Propositional Logic (2)

- 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:

Propositional Logic (3)

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

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

$$\equiv$$

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

Notes:

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

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

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.

Notes:

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**

Notes:

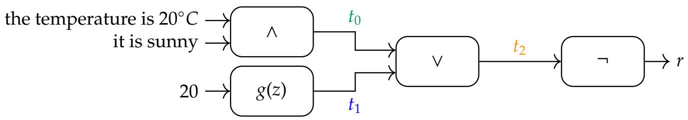
Propositional Logic (6)

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**

Notes:

Boolean Algebra (1)

- If you look closely, some commonalities between propositional logic and *other* concepts in Mathematics start to emerge:
  1. In **elementary algebra**, for some number x we have that
     $$x + 0 = x$$
     and
     $$x \cdot 1 = x.$$
  2. In **propositional logic**, for some truth value x we have that
     $$x \vee \textbf{false} = x$$
     and
     $$x \wedge \textbf{true} = x.$$
  3. In **set theory**, for some set x we have that
     $$x \cup \emptyset = x$$
     and
     $$x \cap \mathcal{U} = x.$$

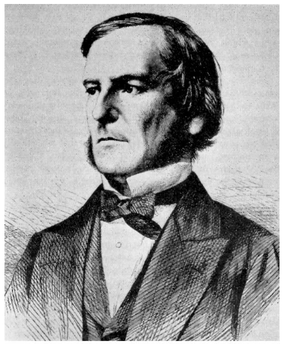
Notes:

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

Notes:

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

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 $\vee$).

Boolean Algebra (3)

- Put more concretely, we now have

2. a set of axoims 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 some others, such as **precedence** to deal with any 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 $\vee$).

Boolean Algebra (3)

- Put more concretely, we now have

2. a set of axoims 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 some others, such as **precedence** to deal with any 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 $\vee$).

Boolean Algebra (3)

- Put more concretely, we now have

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

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 some others, such as **precedence** to deal with any 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 $\vee$).

Boolean Algebra (3)

- Put more concretely, we now have

2. a set of axoims 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 some others, such as **precedence** to deal with any 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 $\vee$).

Boolean Algebra (4)

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

Notes:

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

Notes:

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.

Notes:

Conclusions

- Take away points:
  1. In essence, Boolean algebra is a (somewhat) cosmetic extension of what you already know.
  2. Keep in mind that
     - *any* Boolean function f which can be expressed by a truth table can be computed using an associated Boolean expression, so
     - if we can construct *physical* implementations of NOT, AND and OR we can build something to actually compute f,

     i.e., Boolean algebra is an important formal basis for reasoning about computation (and thus computers) work in practice.

Notes:

Additional Reading

- *Wikipedia: Boolean algebra*. URL: http://en.wikipedia.org/wiki/Boolean_algebra.
- D. Page. "Chapter 1: Mathematical preliminaries". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer-Verlag, 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: http://en.wikipedia.org/wiki/Boolean_algebra (see p. 75).

[2] D. Page. "Chapter 1: Mathematical preliminaries". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer-Verlag, 2009 (see p. 75).

[3] W. Stallings. "Chapter 11: Digital logic". In: *Computer Organisation and Architecture*. 9th ed. Prentice-Hall, 2013 (see p. 75).

[4] A.S. Tanenbaum and T. Austin. "Section 3.1: Gates and Boolean algebra". In: *Structured Computer Organisation*. 6th ed. Prentice-Hall, 2012 (see p. 75).

Notes: