Boolean Algebra

"combinatorial logic"

A Boolean algebra consist of:
- Set of elements $\mathcal{B}$ (E.g. $\mathcal{B} = \{0,1\}$, but could be more general)
- binary operators $\vee, \wedge$
  - or: OR, AND
  - or: $+, \cdot$
- unary operator $'$ or: NOT or: "$-$"

Axioms:

B1 ("closure"): $a \vee b \in \mathcal{B}$ $\forall\, a,b \in \mathcal{B}$ ("$a \vee b$ is contained in $\mathcal{B}$ for all a, b in $\mathcal{B}$")
$a \wedge b \in \mathcal{B}$ $\forall\, a,b \in \mathcal{B}$

B2 ("commutative law"): $a \vee b = b \vee a$ $\forall\, a,b \in \mathcal{B}$
$a \wedge b = b \wedge a$ "

B3 ("associative law"): $a \vee (b \vee c) = (a \vee b) \vee c$ $\forall\, a,b,c \in \mathcal{B}$
$a \wedge (b \wedge c) = (a \wedge b) \wedge c$ "

B4 ("identity"): There exist $0 \in \mathcal{B}$ and $1 \in \mathcal{B}$ s.t.
$a \vee 0 = a$
$a \wedge 1 = a$

B5 ("distributivity"): $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

B6 ("complement"): $a \vee a' = 1$
$a \wedge a' = 0$

Note: The red axioms are different from standard algebra if $\vee$ is identified with $+$ and $\wedge$ is identified with $\cdot$.

Theorem 1: $a \vee a = a$

Proof: $a \overset{(B4)}{=} a \vee 0 \overset{(B6)}{=} a \vee (a \wedge a') \overset{(B5)}{=} (a \vee a) \wedge (a \vee a') \overset{(B6)}{=} (a \vee a) \wedge 1 = a \vee a$ □

Theorem 2: $a \vee 1 = 1$

Proof: $a \vee 1 \overset{(B6)}{=} a \vee (a \vee a') \overset{(B3)}{=} (a \vee a) \vee a' \overset{(T1)}{=} a \vee a' \overset{(B6)}{=} 1$ □

Theorem 3: $a \vee (a \wedge b) = a$ ("absorption")

Proof: $a \vee (a \wedge b) \overset{(B4)}{=} (a \wedge 1) \vee (a \wedge b) \overset{(B5)}{=} a \wedge (1 \vee b) \overset{(T2)}{=} a \wedge 1 \overset{(B4)}{=} a$ □

Theorem 4: $a = b \wedge a \iff a \vee b = b$

"iff"
"if and only if"

Proof: "$\Rightarrow$": Suppose $a = b \wedge a$

$\Rightarrow a \vee b \overset{\downarrow}{=} (b \wedge a) \vee b$

$\overset{(B2)}{=} b \vee (b \wedge a)$

$\overset{(T3)}{=} b$

"$\Leftarrow$": See argument regarding "dual" below.

Note that the axioms B1-B6 are completely symmetric under simultaneous exchange of

$$\wedge \leftrightarrow \vee$$

$$0 \leftrightarrow 1$$

This proves the following meta-theorem (theorem about theorems):

From each of the theorems about Boolean algebras you can generate a new theorem by exchanging $\vee \leftrightarrow \wedge$ and simultaneously $0 \leftrightarrow 1$.
The resulting theorem is called the corresponding <u>dual</u> theorem.

Example 1: This completes the proof of Theorem 4 as follows.

We have already proved that $a = b \wedge a \Rightarrow a \vee b = b$

Dual theorem: $a = b \vee a \Rightarrow a \wedge b = b$

Now exchange $a \leftrightarrow b$ (these are just names!) and re-arrange:

$$a \vee b = b \Rightarrow a = a \wedge b$$

□

Theorem ("De Morgan's Law"): $(a \vee b)' = a' \wedge b'$

Proof: To assert that $a' \wedge b'$ is the complement of $a \vee b$, we need to check the complement properties encoded in B6:

(i) $(a' \wedge b') \vee (a \vee b) \overset{(B5)}{=} (\underbrace{a' \vee a}_{=1} \vee b) \wedge (a \vee \underbrace{b \vee b'}_{=1}) = 1 \wedge 1 = 1$

(ii) $(a' \wedge b') \wedge (a \vee b) \overset{(B5)}{=} (\underbrace{a' \wedge b' \wedge a}_{=0}) \vee (a' \wedge \underbrace{b' \wedge b}_{=0}) = 0 \vee 0 = 0$ □

Example 2: "Dual De Morgan:" $(a \wedge b)' = a' \vee b'$

More complicated example:

"Consensus Theorem": $(a \wedge b) \vee (b \wedge c) \vee (a' \wedge c) = (a \wedge b) \vee (a' \wedge c)$

Proof:

$$(a \wedge b) \vee (b \wedge c) \vee (a' \wedge c) \overset{(B1)}{=} (a \wedge b) \vee (1 \wedge b \wedge c) \vee (a' \wedge c)$$

$$\overset{(B6)}{=} (a \wedge b) \vee ((a \vee a') \wedge b \wedge c) \vee (a' \wedge c)$$

$$\overset{(B5)}{=} (a \wedge b) \vee (a \wedge b \wedge c) \vee (a' \wedge b \wedge c) \vee (a' \wedge c)$$

$$\overset{(B5)}{=} a \wedge (b \vee (b \wedge c)) \vee a' \wedge ((b \wedge c) \vee c)$$

$$\overset{T3}{=} (a \wedge b) \vee (a' \wedge c)$$

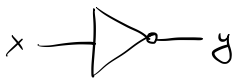
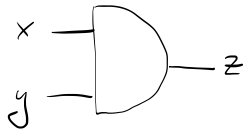
□

Binary logic

- Elementary building blocks of digital computers
- $\mathbb{B} = \{0,1\}$ represented by differing voltage levels in electronic circuitry
- Relatively robust to noise
- Scales to billions of elementary logic gates on a single chip
- Fast

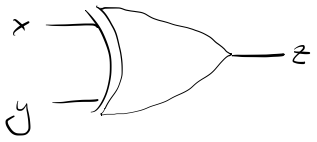
Def.: A **Boolean function** maps one or more inputs from $\mathbb{B} = \{0,1\}$ to a single output from $\{0,1\}$.

Logic gates:

		Truth table	Circuit symbol
NOT:	$y = x'$	$\begin{array}{c\|c} x & y \\ \hline 0 & 1 \\ 1 & 0 \end{array}$	
AND:	$z = x \wedge y$	$\begin{array}{cc\|c} x & y & z \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}$	
OR:	$z = x \vee y$	$\begin{array}{cc\|c} x & y & z \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array}$	

These are the elementary ones (as in the axioms of Boolean algebra), but there are other useful abbreviations: NAND ("NOT AND"), NOR ("NOT OR"), XOR ("eXclusive OR"):

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0



etc.

Note:

$$\begin{aligned} x \text{ XOR } y &= (x \vee y) \wedge (x \wedge y)' \\ &= x \wedge (x \wedge y)' \vee y \wedge (x \wedge y)' && \text{(distributivity)} \\ &= (x' \vee (x \wedge y))' \vee (y' \vee (x \wedge y))' && \text{(De Morgan)} \\ &= \Big(\underbrace{(x' \vee x)}_{=1} \wedge (x' \vee y)\Big)' \vee \Big((y' \vee x) \wedge \underbrace{(y' \vee y)}_{=1}\Big)' && \text{(distributivity)} \\ &= (x \wedge y') \vee (y \wedge x') && \text{(De Morgan)} \end{aligned}$$

Typical task in logic circuit design: Take truth table, write it as Boolean logic expression and simplify so that it can be implemented in the least number of elementary logic gates.

Example:

a	b	c	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$$z = a' \wedge b \wedge c \vee \underbrace{a \wedge b' \wedge c' \vee a \wedge b' \wedge c}_{= a \wedge b' \wedge (c \vee c') = 1} \vee \underbrace{a \wedge b \wedge c' \vee a \wedge b \wedge c}_{= a \wedge b \wedge (c \vee c') = 1}$$

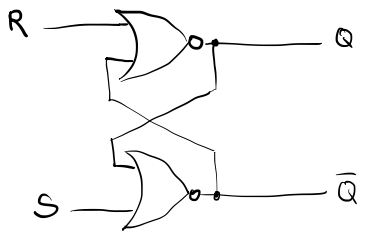
$$= a \wedge (b \vee b')$$
$$= a \wedge 1$$
$$= a$$

$$= a' \wedge b \wedge c \vee a$$

$$= b \wedge c \vee a \qquad \text{(why???)}$$

To build a computer, we need memory. The following is a simple model of memory:

Flip-flop:



S	R	Q
0	0	hold state
0	1	0
1	0	1
1	1	not allowed (breaks complementarity, race on switch to 0)

Note: Flip-flops can be used to build state machines (needs "clock")