Excursions in Computing Science: Week 10 The laws of thought

T. H. Merrett[*]
McGill University, Montreal, Canada

March 26, 2007

1. Let's take a look at a strange new algebra. We'll give it first in terms of the formal rules, or *axioms*, it obeys.

- operators +, . are
  - (a) commutative
  - (b) associative
- ! (c) identities: 0 for +; 1 for .; and

$$\begin{aligned} X + 1 &= 1 \\ X.0 &= 0 \end{aligned}$$

- ! (d) *mutually* distributive:

$$\begin{aligned} X.(Y+Z) &= X.Y + X.Z \\ X + Y.Z &= (X+Y).(X+Z) \end{aligned}$$

- ! (e) absorptive:

$$\begin{aligned} X + X.Y &= X \\ X.(X+Y) &= X \end{aligned}$$

- ! (f) unary operator ′ ("complement")

$$\begin{aligned} X + X' &= 1 \\ X.X' &= 0 \end{aligned}$$

From this (treating the rules formally as if for a game such as chess)

[*] The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

(1)

$$\begin{aligned} X+0 &= X+X.X' = X \\ X.1 &= X.(X+X') = X \end{aligned}$$

(2) if $X+Y=1$ and $X.Y=0$ then $Y=X'$:

$$\begin{aligned} Y &= Y.1 = Y.(X+X') = Y.X + Y.X' \\ &= 0 + Y.X' = X.X' + Y.X' \\ &= (X+Y).X' = X' \end{aligned}$$

(2a) $Y = Y''$:
combine (2) and (f) (set $X = Y'$).

(2b)

$$\begin{aligned} (X+Y)' &= X'.Y' \\ (X.Y)' &= X'+Y' \end{aligned}$$

De Morgan's laws. These follow from (2), since:

$$\begin{aligned} (X+Y)+X'.Y' &\overset{(d)}{=} ((X+Y)+X').((X+Y)+Y') \\ &\overset{(b)}{=} (Y+X+X').(X+Y+Y') \\ &\overset{(f)}{=} (Y+1).(X+1) \\ &\overset{(c)}{=} 1.1 \\ &= 1 \\ \text{and} \\ (X+Y).(X'.Y') &\overset{(d)}{=} X.X'.Y' + Y.X'.Y' \\ &\overset{(f),(c)}{=} 0+0 \\ &= 0 \end{aligned}$$

2. The algebra obeying these rules is "boolean algebra" [Boo54].

The simplest example has only two elements, 0 and 1.

$$\begin{array}{c|cc} 1 & 1 & 1 \\ 0 & 0 & 1 \\ \cline{2-3} + & 0 & 1 \end{array} \quad \begin{array}{c|cc} 1 & 0 & 1 \\ 0 & 0 & 0 \\ \cline{2-3} . & 0 & 1 \end{array} \quad \begin{array}{c|c} 1 & 0 \\ 0 & 1 \\ \cline{2-2} ' & \end{array}$$

(The existence of this particular interpretation (mathematicians call it a "model") of boolean algebra proves that the axioms (a)..(f) are *consistent*. It does not show that the axioms are *complete* or *independent*, which we do not attempt here.)

From now on, I'm going to stop using '.' as an operator and just write the operands adjacent to each other, as we normally do with multiplication in algebra. Thus $X.Y$ will henceforth be XY.

For this particular, two-valued model, we can use "truth tables" to check that the axioms apply. For instance, the second distributive law:

X	Y	Z	$X+YZ$	$(X+Y)(X+Z)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

We will now interpret +, ., and ', respectively as **or**, **and** and **not**, and we will interpret 0 and 1 as **false** and **true**, respectively.

Boolean algebra now formalizes *propositional logic* (hence Boole's motivation for titling his book *The Laws of Thought*).

Here are some example propositions, each abbreviated by a suitable letter.

L "lightspeed is the same for all observers"

R "ontogeny recapitulates phylogeny"

W "water is composed of oxygen and hydrogen"

Here are the values of each of these.

$$\begin{aligned} L &= 1\ \textbf{true} \\ R &= 0\ \textbf{false} \quad \text{“Haeckel's Lie”} \\ W &= 1\ \textbf{true} \end{aligned}$$

Here are some combinations.

$$\begin{aligned} L\ \textbf{or}\ R &= L+R:\ \textbf{true} \\ L\ \textbf{and}\ R &= LR:\ \textbf{false} \\ L\ \textbf{and}\ W &= LW:\ \textbf{true} \\ \textbf{not}\ R &= R':\ \textbf{true} \end{aligned}$$

3. A useful but misunderstood logical operator is *implication*.

$$\begin{array}{c|cc} 1 & 0 & 1 \\ 0 & 1 & 1 \\ \cline{2-3} \to & 0 & 1 \end{array} \qquad \begin{array}{c|cc} 1 & 0 & 1 \\ 0 & 1 & 1 \\ \cline{2-3} \leq & 0 & 1 \end{array}$$

Note that **false** implies anything.

Using two of the above propositions,

$$\begin{aligned} \textbf{if}\ R\ \textbf{then}\ L &= R \to L:\ \textbf{true} \\ \textbf{if}\ L\ \textbf{then}\ R &= L \to R:\ \textbf{false} \end{aligned}$$

→ is not commutative. (It is *transitive*:

$$X \to Y\ \textbf{and}\ Y \to Z \quad \text{gives} \quad X \to Z)$$

For example,

N "x is a minimum of $f()$"

S "the slope of $f()$ at x is zero"

$$\begin{aligned}\textbf{if } N \textbf{ then } S &= N \rightarrow S: \ \textbf{true}\\ \textbf{if } S \textbf{ then } N &= S \rightarrow N: \ \textbf{false}\end{aligned}$$

(x might be the maximum)

X "x is a maximum of $f()$"

$$\begin{aligned}\textbf{if } X \textbf{ then } S &= X \rightarrow S: \ \textbf{true}\\ \textbf{if } S \textbf{ then } (S \textbf{ or } N) &= X \rightarrow (S+N): \ \textbf{false}\end{aligned}$$

Can we get → from +, ., ' (**or**, **and**, **not**)?

X	Y	$X \rightarrow Y$
0	0	1
0	1	1
1	0	0
1	1	1

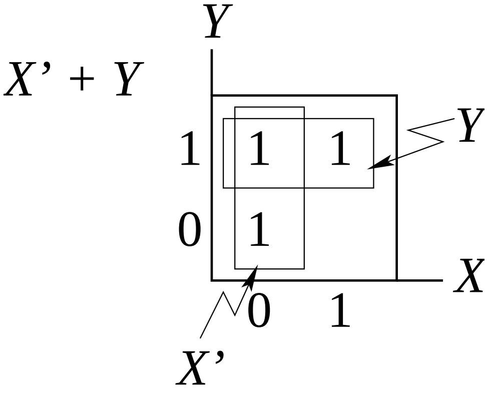
We can: for each row containing a 1 under $X \rightarrow Y$, write down the following
(if $X = 1$ then X else X')(if $Y = 1$ then Y else Y')
then take the (boolean) sum of all of these for which there was a 1 under $X \rightarrow Y$.

This gives $X \rightarrow Y$ is $X'Y' + X'Y + XY$.

We can *simplify* this using the axioms of boolean algebra.

$$\begin{aligned}X'Y' + X'Y + XY &= X'Y' + X'Y + X'Y + XY\\ &= X'(Y' + Y) + (X' + X)Y\\ &= X' + Y\end{aligned}$$

We can also simplify it visually, using a "2-cube" (a two-dimensional cube, i.e., a square).



X	Y	$X \rightarrow Y$	$X' + Y$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	1	1

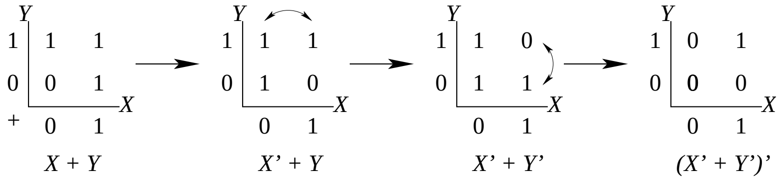
4. What is beyond →? We can find a grand total of sixteen binary operators.

false	$\begin{array}{\|cc} 0 & 0 \\ 0 & 0 \\ \hline \end{array}$	**and**	$\begin{array}{\|cc} 0 & 1 \\ 0 & 0 \\ \hline \end{array}$
$\not\leftarrow \quad \not\geq$	$\begin{array}{\|cc} 1 & 0 \\ 0 & 0 \\ \hline \end{array}$	**right**	$\begin{array}{\|cc} 1 & 1 \\ 0 & 0 \\ \hline \end{array}$
$\not\rightarrow \quad \not\leq$	$\begin{array}{\|cc} 0 & 0 \\ 0 & 1 \\ \hline \end{array}$	**left**	$\begin{array}{\|cc} 0 & 1 \\ 0 & 1 \\ \hline \end{array}$
xor, $\times$, $\neq$	$\begin{array}{\|cc} 1 & 0 \\ 0 & 1 \\ \hline \end{array}$	**or**, $+$	$\begin{array}{\|cc} 1 & 1 \\ 0 & 1 \\ \hline \end{array}$
nor	$\begin{array}{\|cc} 0 & 0 \\ 1 & 0 \\ \hline \end{array}$	**nxor**, $=$	$\begin{array}{\|cc} 0 & 1 \\ 1 & 0 \\ \hline \end{array}$
nleft	$\begin{array}{\|cc} 1 & 0 \\ 1 & 0 \\ \hline \end{array}$	$\rightarrow$, $\leq$	$\begin{array}{\|cc} 1 & 1 \\ 1 & 0 \\ \hline \end{array}$
nright	$\begin{array}{\|cc} 0 & 0 \\ 1 & 1 \\ \hline \end{array}$	$\leftarrow$, $\geq$	$\begin{array}{\|cc} 0 & 1 \\ 1 & 1 \\ \hline \end{array}$
nand	$\begin{array}{\|cc} 1 & 0 \\ 1 & 1 \\ \hline \end{array}$	**true**	$\begin{array}{\|cc} 1 & 1 \\ 1 & 1 \\ \hline \end{array}$

although six of them are not strictly binary, and only $\leq$, **xor**, **nand** and **nor** are logically interesting besides **and** and **or**. (Interpretation: e.g., X **right** $Y = Y$.

$$\begin{array}{c|cc} 1 & 1 & 1 \\ 0 & 0 & 0 \\ \hline & 0 & 1) \end{array}$$

Let's show that we don't need both **and** and **or** as basic operators, by getting **and** from **or** and **not**.



Of course, it's just de Morgan's law, but seen in a new way.

We can get unary operators from these binary ones in several ways:

$$\begin{array}{lcll} \text{fix } X & = & 0 & \text{(i.e., use left column)} \\ \text{fix } X & = & 1 & \text{(i.e., use right column)} \\ \text{fix } Y & = & 0 & \text{(i.e., use bottom row)} \\ \text{fix } Y & = & 1 & \text{(i.e., use top row)} \\ \text{fix } X & = & Y & \text{(i.e., use diagonal, bottom left to top right)} \end{array}$$

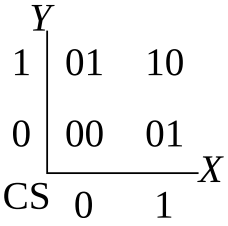
Thus X **nand** $X =$ **not** X.

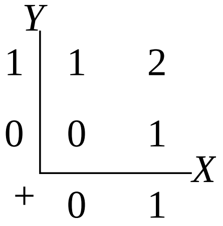
5. The two-element boolean algebra also describes *switching circuts.*

These are built from primitive elements such as **not**, **and**, **or**—or **not**, **or**—or just **nand**.
Let's build an adder out of **not**, **and**, **or**.
"Half adder", `h`, for the least significant bit; "full adder", `f`, for the other bits.

```
 fffh
 1011
+ 101
10000
```

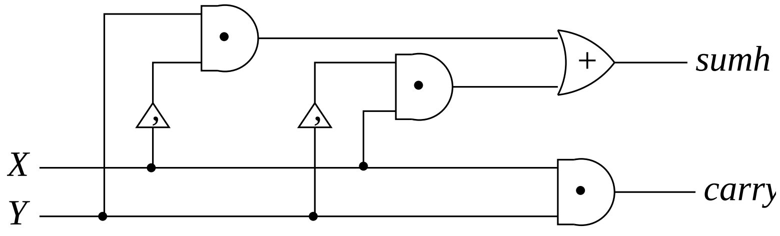
X	Y	*carry*	*sumh*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





This is easy:

$$\begin{aligned} carry &= XY \\ sumh &= X \textbf{ xor } Y = XY' + X'Y \end{aligned}$$



Half adder

This will do to add the first bit of each two multi-bit numbers.

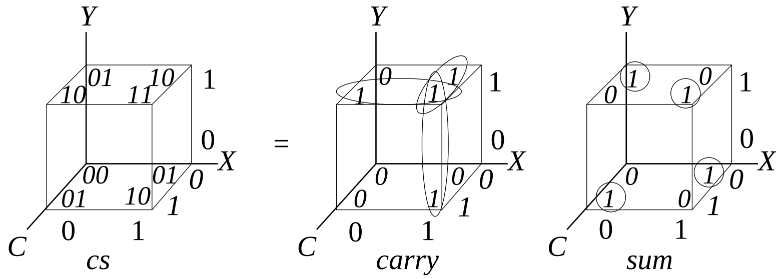
Here it is in MATLAB.

```
% function [carry,sumh] = halfadder(x,y)
% THM 060828
function [carry,sumh] = halfadder(x,y)
carry = and(x,y);
sumh  = xor(x,y);
```

To add subsequent bits requires a bunch of *full adders*

C	X	Y	*carry*	*sum*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(Note: *carry* $= \geq 2$ bits are on;
sum = odd # bits are on.)



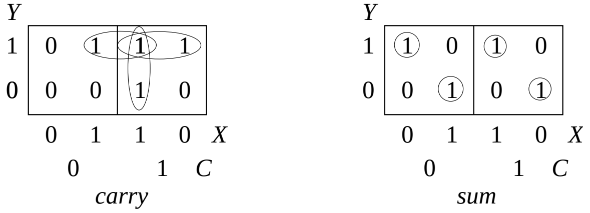
Carry is made up of three 1-dimensional pairs, each requiring only $3 - 1 = 2$ variables:

$$\textit{carry} \ = CX + CY + XY$$

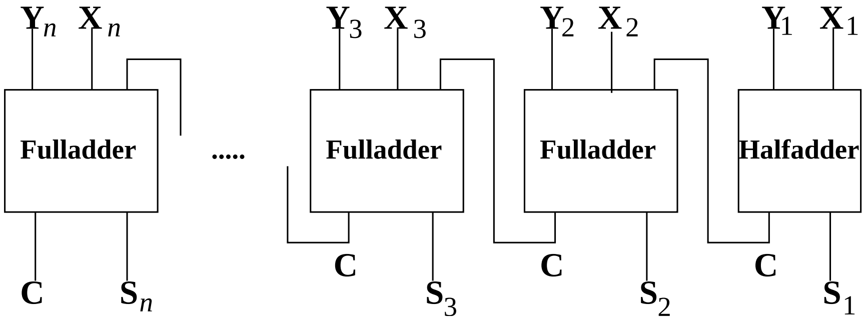
Sum is made up of four 0-dimensional pieces, so it cannot be simplified: each requires $3 - 0 = 3$ variables:

$$\textit{sum} \ = CX'Y' + C'XY' + C'X'Y + CXY$$

Since some of us are straining at these three-dimensional representations, and since four or more input variables would require four or more dimensions, where we would all have difficulty, let's look at a 2-D encoding of 3, 4, 5, .. dimensions: the *Karnaugh map.*



Once we have both halfadder and fulladder, we can add any number of bits.



6. Reversibility

None of our binary operators is reversible: given the output we cannot reconstruct the inputs. Charles Bennett showed in 1973 that a reversible calculation can be done, in principle, consuming *no* energy, so reversibility is significant.

Which of our boolean operations is reversible?

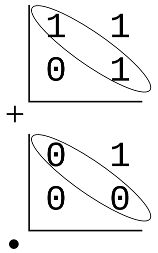
- **not**: $X'' = X$

This is a unary operator: 1 input → 1 output.

Binary operators give only 1 output for 2 inputs, so they cannot be reversible.

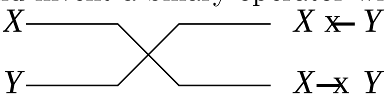
What if we combine them?

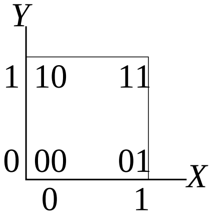
X	Y	$X+Y$	XY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1



In the rectangle in the truth table (and the ellipses in the operator tables), $X + Y$ and XY stay the same, but give different results for X and Y: thus **or** (+) and **and** (.) cannot be combined in any way to give back X and Y.

We could invent a binary operator with two outputs which is reversible, such as *exchange.*



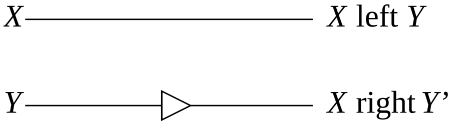


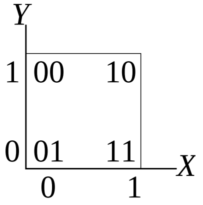
exchange

X	Y	$X \times\!-Y$	$X -\!\times Y$
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

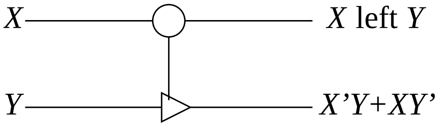
but this does not at first seem interesting: we certainly cannot get **and**, **or** or **not** out of this, or any of the operators that generate a boolean algebra.

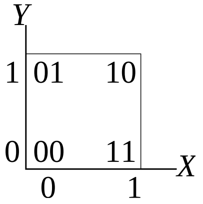
Similarly apparently trivial is





But what about a "controlled not" (**CN**)?

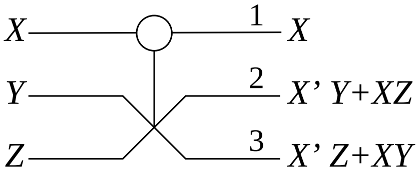




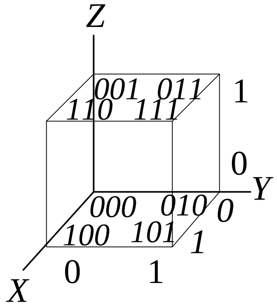
Now we're getting an **xor**: that does not generate a boolean algebra but it is at least less trivial.

How about 3 inputs and 3 outputs?

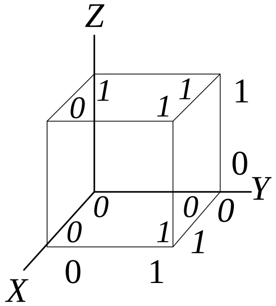
A "controlled exchange" (**CX**) operator.



Fredkin (controlled exchange)



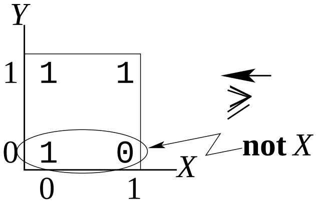
CX $\boxed{123}$



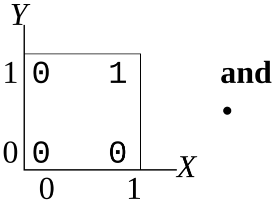
output 3

I've pulled out output 3 so that we can look at it from various points of view, which give different binary operators.

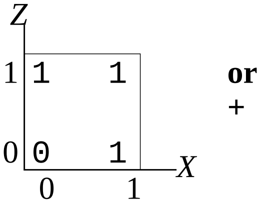
$Z = 1$



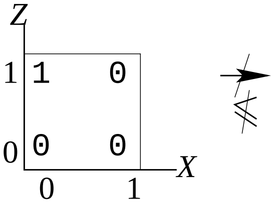
$Z = 0$



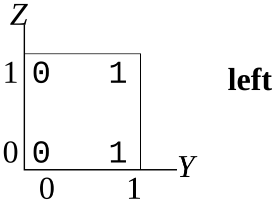
$Y = 1$



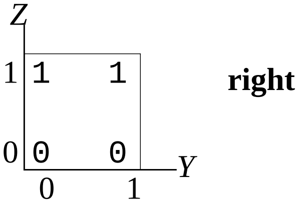
$Y = 0$



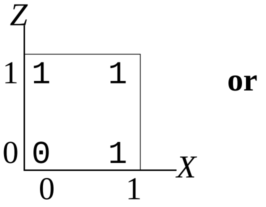
$X = 1$



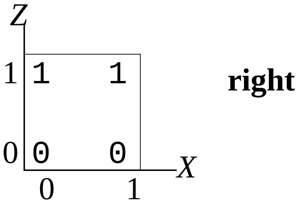
$X = 0$



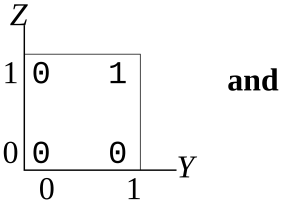
$X{=}Y$



$Y{=}Z$



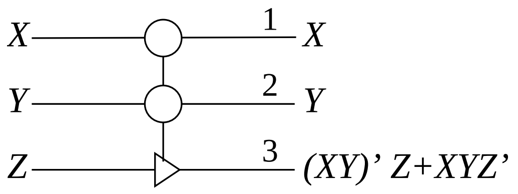
$X{=}Z$



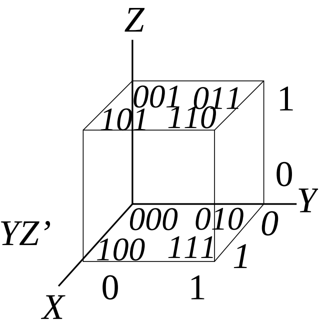
This third output can give us **or**, **and** and even **not** if we consider the 1-dimensional result for $Y = 0$ and $Z = 1$. It also gives implication and other operators.

Thus, the **CX** operator is reversible and contains all of boolean algebra. It is a reversible and *universal* operator.

Another reversible operator is "controlled controlled not" (**CCN** or Toffoli).

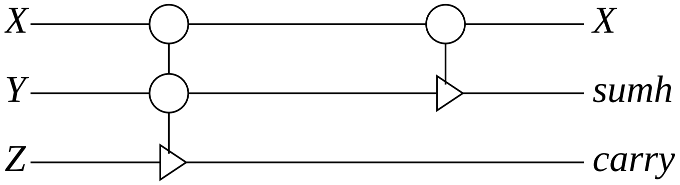


controlled controlled not



CCN

CCN and **CN** give a reversible half-adder.



7. The operator tables look like matrices in `0,1` but they're not.

Can we describe logic/switching operators as matrices?

How about $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f \\ t \end{pmatrix} = \begin{pmatrix} t \\ f \end{pmatrix}$ for **not**?

$$\textbf{not not} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Seems right.

So maybe **and** is

$$\begin{array}{c|cccc} & \textbf{ff} & \textbf{ft} & \textbf{tf} & \textbf{tt} \\ \hline \textbf{f} & 1 & 1 & 1 & 0 \\ \textbf{t} & 0 & 0 & 0 & 1 \end{array}$$

No: 1) it does funny things:

$$\textbf{and} \begin{pmatrix} \textbf{ff} \\ \textbf{ft} \\ \textbf{tf} \\ \textbf{tt} \end{pmatrix} = \begin{pmatrix} \textbf{f} + \textbf{f} + \textbf{f} \\ \textbf{t} \end{pmatrix}$$

2) the matrix is not square.

Let's focus on operators with the same number of outputs as inputs.

Then the matrices could describe a *change of state* from input state to output state.

$$\textbf{CN} \quad \begin{array}{c|cccc} & \textbf{ff} & \textbf{ft} & \textbf{tf} & \textbf{tt} \\ \hline \textbf{ff} & 1 & & & \\ \textbf{ft} & & 1 & & \\ \textbf{tf} & & & & 1 \\ \textbf{tt} & & & 1 & \end{array} \quad \begin{array}{c} X_0\,Y_0 \\ \begin{pmatrix} \textbf{ff} \\ \textbf{ft} \\ \textbf{tf} \\ \textbf{tt} \end{pmatrix} \end{array} = \begin{array}{c} X_1\,Y_1 \\ \begin{pmatrix} \textbf{ff} \\ \textbf{ft} \\ \textbf{tt} \\ \textbf{tf} \end{pmatrix} \end{array}$$

Let's look at this as a tensor product of matrices for X and Y.

$$\begin{array}{rcl} \left(\begin{array}{c|c} 1 & \\ \hline & 0 \end{array}\right)_X \begin{pmatrix} 1 & \\ & 1 \end{pmatrix}_Y & + & \left(\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}\right)_X \begin{pmatrix} & 1 \\ 1 & \end{pmatrix}_Y = \\ \textbf{ifnot}_X\ 1_Y & + & \textbf{if}_X\ \textbf{not}_Y \\ (\text{Compare } X'Y & + & XY' \end{array}$$

but that gives only one of the outputs.)

Feynman [Fey99, Chap. 6] uses this as the basis for analyzing reversible operators in a quantum computer: quantum states can be *superpositions* of classically mutually exclusive states.

8. Summary

(These notes show the trees. Try to see the forest!)

- boolean algebra
  - logic: **and**, **or**, **not**
  - logic: **if .. then ..**
    16 binary operators
- circuits: adders, boolean simplification
  - Extracting boolean expression from truth table (expression = term + term + .. + term; term = component.component. .. .component; component = variable or variable′)
    1. Ignore output entries of `0`. Each output entry of `1` contributes a term to the expression.
    2. Each input entry of `1` contributes that variable to the term. Each input entry of `0` contributes the negation of that variable to the term.
  - Simplifying the boolean expression (arrange the truth table as a hypercube or as a Karnaugh map, and apply the previous rules)
    1. Look for patterns of output `1`s: if any input variable contributes both `0` and `1` to output `1`s, that variable may be dropped from the corresponding terms and the terms combined into a single term. (Patterns include lines parallel to axes, squares, ..)
- reversible operators and universal operators
- matrix formulation

9. **Appendix: Binary Arithmetic**

We count on 10 fingers. Computers count on 2 electrical states.

Binary	0	1	10	11	100	101	110	111	...
Decimal	0	1	2	3	4	5	6	7	...

1	1	10	11
0	0	1	10
+	0	1	10

```
Decimal                                 Binary

        825                                     101     (5)
     + 9346                                  + 1101    (13)
      -----                                   -----
carry   1101                           carry   1101
sum     10171                          sum     10171    (18)

          1   10^0                                0   2^0
         70   10^1                                2   2^1
        100   10^2                              0*4   2^2
       0000   10^3                             0*8    2^3
      10000   10^4                             1*16   2^4
```

Adding least significant bits (digits) requires 2 inputs: `halfadder(x,y)`.
Adding any other pairs of bits (digits) requires 3 inputs: `fulladder(c,x,y)`.

10. **Excursions for Friday and beyond.**
You've seen lots of ideas. Now *do* something with them!

1. Derive $X + X = X$ and $XX = X$ from axioms (e.)

2. Run the MATLAB function

   ```
   % function OT= OperatorTableOr()
   % THM 060828
   function OT = OperatorTableOr()
   for x = 0:1
    for y = 0:1
     OT(y+1,x+1) = or(x,not(y));
    end
   end
   ```

   and write corresponding ones for **and** and **not**.

3. For a two-element boolean algebra, verify all the axioms of boolean algebra by using the truth tables for +, ., and '.

4. How do the boolean axioms change if we swap 0 ↔ 1 and + ↔ .?

5. Show that *if* **not** Q *then* **not** P is equivalent to *if* P *then* Q, for any propositions P and Q.

6. Since **not not** X is X, boolean logic imposes the "law of the excluded middle".
   This allows us to make proofs by contradiction: suppose what you want to prove is untrue, then derive an impossibility from this supposition. This is used, for instance in automated reasoning (J. A. Robinson's "resolution principle").
   We'll do something simpler, which we'll need in Week 12.
   Use contradiction to show that the greatest common divisor of integers x and y,
   $$\gcd(x, y) = \gcd(y, x \bmod y),$$
   where $x \bmod y$ is the remainder after x is divided by y. Note that $x = (x \div y)y + x \bmod y$, where $x \div y$ is the integer quotient of dividing x by y, and that if $\gcd(x, y) = g$ then $x = gp$ and $y = gq$ for some integers p and q such that $\gcd(p, q) = 1$.

7. *More than one kind of infinity* a) Use contradiction to prove that there are numbers beyond the "rational" numbers, which are ratios, p/q, of integers p and q: suppose that $\sqrt{2} = p/q$, with p and q in lowest terms (i.e., $p \bmod q = 1$). Hint: show that if p is odd then p^2 must

be odd (for any integer, p). (This discovery so upset Pythagoras' belief in the supremacy of integers that he reputedly had Hippasus, who made it, drowned: South Italy, 550BC.)
b) Show that the collection of all rational numbers is *countable*, i.e., ways can be found to order each rational number, p/q, in a sequence so that each number can be paired with a non-negative integer in the sequence 0, 1, 2, .. Hint: make a two-dimensional grid for p and q and find a way to draw a connected line through all the grid points.
c) Check that each rational number, p/q, can be converted to decimal form using a procedure such as the following for 1/7: for digits $a, b, c, d, e, ..$ (which can only take on the values 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9)

$$\begin{aligned}
1/7 &= a/10 + b/10^2 + c/10^3 + d/10^4 + e/10^5 + .. \\
&= \frac{1}{10}(a + \frac{1}{10}(b + \frac{1}{10}(c + \frac{1}{10}(d + \frac{1}{10}(e..))))) \\
\frac{10}{7} &= a + \frac{1}{10}(b + \frac{1}{10}(c + \frac{1}{10}(d + \frac{1}{10}(e + ..))))
\end{aligned}$$

So $a = 1$.

$$\begin{aligned}
\frac{10}{7} - 1 = \frac{3}{7} &= \frac{1}{10}(b + \frac{1}{10}(c + ..)) \\
\frac{30}{7} &= b + \frac{1}{10}(c + \frac{1}{10}(d + \frac{1}{10}(e + ..)))
\end{aligned}$$

So $b = 4$.

$$\begin{aligned}
\frac{30}{7} - 4 = \frac{2}{7} &= \frac{1}{10}(c + ..) \\
\frac{20}{7} &= c + \frac{1}{10}(d + \frac{1}{10}(e + ..))
\end{aligned}$$

So $c = 2$. At this point, let's stop and look ahead. How many *different* remainders can there be, at most? How soon, at most, will we see any given remainder *again* in this process? What does this mean for the digits in the decimal form of 1/7?
Now carry on with 1/7 by finding digits $d, e, f, ..$ and see if you were right. Try it for some other fractions. Persuade yourself that the sequence of digits always repeats after a certain point. (The repetition may be of only a single digit, and this digit may even be 0: include some fractions that give these special cases in your experimenting.)
You can also use this procedure to find the "decimal" representation of a fraction in any base b: just replace all the 10s, above, by b. Here is the start for 1/7 in binary.

$$\begin{aligned}
1/7 &= a/2 + b/2^2 + c/2^3 + d/2^4 + e/2^5 + .. \\
&= \frac{1}{2}(a + \frac{1}{2}(b + \frac{1}{2}(c + \frac{1}{2}(d + \frac{1}{2}(e + ..))))) \\
\frac{2}{7} &= a + \frac{1}{2}(b + \frac{1}{2}(c + \frac{1}{2}(d + \frac{1}{2}(e + ..))))
\end{aligned}$$

where bits $a, b, c, d, e, ..$ can only take on the values 0 or 1.
d) How many numbers are there whose decimal representations contain no repetitions? Here is Cantor's argument that there are more than can even be counted. It is another argument by contradiction, the "diagonal" argument. It is sufficient to consider only fractions between 0 and 1, and to represent them in binary. Suppose we have found a sequence which includes all the infinitely-long fractions, and replace the diagonal elements by their opposites (shown in bold in the second column).

00000000...	**1**0000000...
10000000...	1**1**000000...
11111111...	11**0**11111...
01010101...	010**0**0101...
10101010...	1010**0**010...
11010110...	11010**0**10...
00110110...	001101**0**0...
10001000...	1000100**1**...

Convince yourself that the replaced bits themselves form a new infinitely-long fraction

11000001..

which is not in the sequence, contradicting our supposition that the sequence was complete. (To see that the new infinitely-long fraction does not repeat and so cannot be rational, look up www.mathpages.com/home/kmath371.htm: if it did, the rationals would be uncountable, contradicting (b), above.)
e) Cantor also showed that the *power set* (the set of all subsets) of an infinite set has a higher infinity of elements than the original set. The argument is also by contradiction and also uses a form of diagonal. Consider, for example, the (countably infinite) set of integers. Suppose we have paired off its power set with integers, so that we can count it.

$\mathcal{N}$	P($\mathcal{N}$)
1	{4,5}
2	{1,2,3}
3	{4,5,6}
4	{1,3,5}
:	:

In this example, 2 is paired with a subset that contains 2, but 1, 3 and 4 are paired with subsets that do not contain themselves. Which integer is paired with the subset, {1,3,4,..}, of all integers that are paired with subsets that do not contain themselves? Is it *not* in this subset? Then it is paired with a subset that does not contain itself, and so it is in the subset: contradiction. Conversely, if it *is* in this subset, then it is not in it: there is no escape.
This is a diagonal argument because it is asking whether, in row ? below, there is a • on the diagonal, and getting a contradiction for both alternatives.

	1	2	3	4	5	6	..
1				●	●		
2	●	●	●				
3				●	●	●	
4	●		●		●		
:							
?	●		●	●			?

So instead of writing ∞, we need more than one symbol. We could call the kind of infinity that the integers have $\aleph_0$ and the kind of infinity that the power set of the integers has $\aleph_1$. What about the power set of the power set? $\aleph_2$? Which of these is the infinity that the real numbers (rational numbers and irrationals together) have?

8. What are the differences among the following three propositions?

$$X \textbf{ and } Y, \quad X \textbf{ and/or } Y, \quad X \textbf{ or } Y$$

9. Why is it not true that
   if (the slope of $f()$ at x is zero) then
   (x is a maximum of $f()$ or x is a minimum of $f()$)?

10. Confirm by truth table that $X \rightarrow Y$ is $X'Y' + X'Y + XY$. Convince yourself that the sum-of-products rule that gave this equivalence works in general for any boolean combination.

11. Which of the sixteen binary boolean operators on 0 and 1 are commutative? Associative?

12. Show that X **xor** Y is $XY' + X'Y$.

13. Show that **or** can be derived from **and** and **not**.

14. Show that **or** and **and** can be derived from $\rightarrow$ and **not**.

15. Show that $X \rightarrow Y$ is $Y' \rightarrow X'$.

16. What is the relationship between $\rightarrow$ and $\leq$?

17. Show that X **xor** Y' is X' **xor** Y is (X **xor** Y)$'$.

18. Find eight ways we can get the not operator, $'$, from the sixteen binary operators.

19. Show that **or**, **and** and **not** can all be derived from **nand**. What other single binary operator gives **or**, **and** and **not**?

20. From the table for the half adder we can see directly that

$$\textit{carry} \; = C'XY + C(X+Y)$$

Show that this is the same as

$$\textit{carry} \; = CX + CY + XY$$

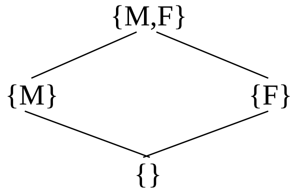
21. Draw the circuit diagram for the full adder.

22. Write the MATLAB function `[carry,sum] = fulladder(c,x,y)`. Combine it with `[carry,sumh] = halfadder(x,y)` from Note 5 to build a function which adds 2-bit numbers. Build a function which adds eight-bit numbers.

23. Which is the better way to schedule a meeting among several people next week: ask them to tell you what times they *are* available; or ask them to tell you when they are *not* available? Imagine that each person will fill out a schedule for the coming week in the form

	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00
Mon								
Tue								
Wed								
Thu								
Fri								

24. Boolean algebra can also represent sets, with the "product" standing for set intersection, the "sum" giving set union and "negation" giving the complement. *Mother* and *father* form a set of two members, $\{M, F\}$, and this set, together with all its subsets, form a four-element boolean algebra.



Here, {} is the *empty set* which is given by the intersection of $\{M\}$ and $\{F\}$. The union of $\{M\}$ and $\{F\}$ gives the full set. The complement of $\{M\}$ is $\{F\}$, and vice-versa.
a) Which of these four elements behaves as `1` and which as `0`? Invent symbols, such as `a` and `b`, for the remaining two sets.
b) Verify that the boolean axioms hold for the three operators on these elements.
c) How many boolean elements arise from a 3-member set? Draw the diagram corresponding to the diamond for the 2-member set, showing all connections between subsets (upwards for unions, downwards for intersections). To what three-dimensional figure is this diagram equivalent?

25. If we have a set, A, of apples and a set, R, of red fruit, the set AR is the red apples. (There might also be green and yellow apples, and red strawberries.) The set $A + R$ is all the fruit that is red or an apple.
This is especially useful in assessing probabilities: the probability of a set is the number in the set divided by the total number. The intersection of two overlapping sets is assigned as probability the (arithmetic) product of the probabilities of the two sets. (What is it if the sets are disjoint?) The union of two disjoint sets is assigned the probabality that is the (arithmetic) sum. But if the sets overlap, we must use De Morgan's laws and the "not" operator: $A + R = (A'R')'$. The probability that something is not in a set is $1 -$ the probability that it is in the set, so "not", $'$, becomes subtraction from 1. Thus the probability that a fruit is an apple or red is $\mathrm{prob}(A + R) = \mathrm{prob}((A'R')') = 1 - \mathrm{prob}(A'R') = 1 - (1 - \mathrm{prob}(A))(1 - \mathrm{prob}(R))$.
Using the counts

	red	green	yellow	TOTALS
apples	10	6	4	20
bananas			8	8
strawberries	20			20
grapes	15	18		33
lemons			3	3
limes		2		2
TOTALS	45	26	15	85

and assuming that this accounts for all fruit, calculate
a) the probability that a fruit is a red apple;
b) the probability that a fruit is not a red apple;
c) the probability that a fruit is a red apple and a green apple;
d) the probability that a fruit is a red apple or a green apple.
Now suppose that you know only the probability that a fruit is an apple and the probability that a fruit is red, and calculate
e) the probability that a fruit is a red and an apple;
f) the probability that a fruit is a red or an apple;
and compare these results to the actual number of fruit that satisfy the two conditions.
g) What is the probability that an apple is red (that is, you know it is an apple, and want to know how likely it is to be red)?

26. The "birthday paradox" states that, with 23 people gathered in a room, the probability that at least two of them share a birthday is over 50%.
a) If n people in a room take turns crossing off their birthday on a calendar of 365 days, and the $k + 1$st person finds k crosses on the calendar (none of the k predecessors shares a birthday with any other predecessor), how many ways out of the 365 can $k + 1$ place heir cross on a blank day?
b) What is the probability that $k + 1$ does not share a birthday with the previous k?
c) What is the probability that none of the n people share a birthday with any other. (Try this for $n < 365$. Think about it for $n \geq 365$.) What assumptions must you make to calculate this probability?
d) From (c), what is the probability that at least two people of the n share a birthday? What must n be for this to exceed fifty-fifty? (Use MATLAB to plot the probability versus n.)

27. Children get half their genetic inheritance from each parent. In the following questions, "genetic overlap" means the probability that the genetic material is the same in two individuals: count the number of ways it can be the same relative to the (total) number of ways it can be the same or different.
a) What is the genetic overlap between siblings?
b) What is the genetic overlap between grandparent and grandchild?
c) What is the genetic overlap between first cousins? Between two first cousins who marry and their children?
d) What is the genetic overlap between yourself and your aunt or uncle? (Consider both possibilities.) What if two brothers married two sisters?
e) What is the genetic overlap between yourself and a half-sibling? A step-sibling?
f) In "haplodiploid" species such as bees, fathers have only one chromosome so that the genetic inheritance from the father is identical in each offspring: what is the genetic overlap between siblings?
g) Supposing that genes control a species' behaviour so as to maximize their own (the genes' own) survival down the generations, would a bee be more likely to risk its own life for a sibling or for an offspring?
h) Do these calculations contradict the assertion that humans share 97% of our genes with chimpanzees?

28. Show that **right**, **left**, **xor**, **nxor**, **nright** and **nleft** can be combined in any pairs, except <op> with **n**<op>, whose results can be combined in turn to give back the original inputs. E.g., $X = X$ **left** $Y, Y = (X$ **xor** $Y)$ **xor** $(X$ **left** $Y)$.

29. Show that any boolean operator can be got from the third output of the controlled exchange operator, **CX**. That is, **CX** is universal.

30. Show that the second output of **CX** gives the same collection of operators as the third output does, and thus is also universal.

31. What operators does the first output of **CX** give?

32. What is the inverse of **CX**? Give two different arguments. (Hint: write the numbers, 0, .., 7, on the cube in two ways, first labelling the inputs, second labelling the outputs.)

33. Look up Edward Fredkin, 1934– . What is the Fredkin gate? What are tries?

34. Show that **CCN**, the controlled controlled not operator, is reversible. What is its inverse? Show that it is universal.

35. Show that **CCN** and the controlled not operator, **CN**, do give a half adder. What is the inverse of the half adder?

36. Can a operator with three inputs and three outputs be reversible if one output is the full adder sum and another output is the full adder carry?

37. Construct a reversible operator, $G(X, Y, Z)$, such that the first output gives **nand** and **nor** and **not** when inputs are equated (e.g., $G_1(X, Y, Y) = X$ **nand** Y) and the second output, $G_2(X, Y, Z)$, gives the full adder sum. What is its inverse?

38. Can a operator be built which is its own inverse and which gives **nand** by setting some inputs equal to each other?

39. How many different reversible three-input operators can there be? How many of these are their own inverse?

40. How can we get exchange from controlled not? Vice versa?

41. Using the matrix representation of operator, show that **CCN** is

$$(\mathbf{ifnot}_X 1_Y + \mathbf{if}_X \mathbf{ifnot}_Y) 1_Z + \mathbf{if}_X \mathbf{if}_Y \mathbf{not}_Z = 1 + \mathbf{if}_X \mathbf{if}_Y (\mathbf{not}_Z - 1_Z)$$

42. Look up George Boole's *The Laws of Thought* [Boo54]. What is the connection between logic and probability?

43. Look up Georg Ferdinand Ludwig Philipp Cantor (1845–1918) on sets and infinities.

44. Look up Richard Feynman's *Lectures on Computation* [Fey99]. What are the energy requirements for computing in principle and in current practice? How many atoms are involved in a modern transistor? What did Charles Bennett prove? What is the significance of reversible computation?

45. Look up Kee Dewdney's *The Turing Omnibus* [Dew89] or *The New Turing Omnibus* [Dew93] (call them [61] and [66], respectively) chapters 3, 13 and 20 [66] (3, 12 and 18 [61]).

46. Any part of the lecture that needs working through.

References

[Boo54] George Boole. *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities.* Dover Publications, 1858 and 1973, 1854. www.gutenberg.org/etext/15114.

[Dew89] A. K. Dewdney. *The Turing Omnibus: 61 Excursions in Computer Science.* Computer Science Press, Rockville, MD, 1989.

[Dew93] A. K. Dewdney. *The New Turing Omnibus: 66 Excursions in Computer Science.* Computer Science Press, Rockville, MD, 1993.

[Fey99] Richard P. Feynman. *Feynman Lectures on Computation.* Westview Press, Oxford, 1999. edited by Tony Hey and Robin W. Allen.