Applied Propositional Logic: theory, simulation code, and digital circuit design

By **Assad Ebrahim**, on May 8th, 2020 (209 views)

This article looks at **Propositional Logic**, also called **Statement Calculus**, from a combinatorial and algebraic point of view, its implementation in software, and its application to digital electronics. An historical section covers the shift in viewpoint from classical logic (based on Aristotle's syllogism) to modern symbolic logic, with the key papers (1830-1881) that drove this shift shared in the **online logic sourcebook** section of the references.

As a practical aid, we implement the grammar of the statement calculus in the **Symbolic Logic Simulator (SLS)**, a program written in 28 lines of **Forth**, that allows computer-aided verification of any theorem in Propositional Logic (see **Appendix 1** for source code). The program makes it straight-forward to explore non-obvious logical identities, and verify any propositional logic theorem or conjecture, in particular see **Appendix 2** for key identities in the statement calculus (duality, algebraic, and canonical identities).

The concept of linguistic adequacy is developed in detail and the NAND Adequacy Theorem is proved showing that NAND can generate all logical operations. A corollary is that any digital logic circuit can be built up entirely using NAND gates, illustrated using the free **Digital Works** software.

1. What is Mathematical Logic?

Mathematical Logic is concerned with (1) exploring the algebraic structures underpinning human reasoning, (2) identifying the limits of human and computer reasoning, and (3) establishing a secure logical foundation for all branches of mathematics.

Why study mathematical logic? Besides a better appreciation of how modern mathematics developed its abstract, formal character, a side benefit is the development of **better reasoning and proof (though see this caution)**.

Although developed quite late in the history of logic, Statement Logic (Propositional Logic) is the simplest logical system in the modern hierarchy of logics, with Set Theory, First Order Logic and Higher Order Logics following on from it. **Statement Logic** considers the logical relations that exist between individual statements, taken as a whole (hence the equivalent term Statement Logic). Statement logic is most typically binary (two-valued), but there are also finite-valued statement logics (aka Post algebras) and continuous valued statement logics (eg fuzzy logic, probability theory).

2. The birth of Statement Logic as a symbolical algebra

Logic prior to the mid 1800s focused on reasoning rules (schematic) for deduction based on relations between statements, such as Aristotle's syllogism: "All A are B. x is an A. Therefore x is B." After Aristotle, the syllogism became the centerpiece of common logic for almost 2000 years. In a similar pattern, the geometry axiomatized by Euclid in his Elements became the paragon for the logical development of a mathematical subject, and led to geometry remaining relatively stagnant for almost 2000 years. The erosion of these classical foundations was slow — so deeply held were the classical perspectives in the shared cultures of the literate civilizations, Muslim and Christian alike, that succeeded the Greeks.

Slow progress unwinding classical influence began in algebra in the 1500s and in geometry in the 1600s. Over the next 200-300 years these advances created the opportunity for new directions in both subjects while also shaking the certainty of the classical perspectives (see **The Rise of Mathematical Logic**).
The advances in Algebra were:
(A1) the discovery of the cubic and quartic formulas,
(A2) the insolvability of the quintic by radicals,
(A3) the impossibility of quadrature of the circle in rational or algebraic numbers, and
(A4) the impossibility of duplication of the cube by straightedge and compass.
In Geometry they were:
(G1) the invention of coordinate geometry,
(G2) the unification of the transcendental functions with imaginary numbers in the theory of series,
(G3) the discovery of the independence of the parallel postulate, and
(G4) the creation of non-Euclidean geometries.

A different direction for logic began to emerge in the 1800s at the hands of English symbolical algebraists, the most influential of whom were Peacock (1830), Gregory (1838), Hamilton (1843), and Cayley (1850). They were followed by the algebraic logicians Boole (1847), De Morgan (1847), and Jevons (1864), who rebuilt logic on an abstract foundation emphasizing the algebraic properties of logical operations.

This modern viewpoint was first introduced in Boole's 1847 treatise, **The Mathematical Analysis of Logic**, in which logic was presented as an uninterpreted **symbolical algebra** defined on a class of statements with reasoning expressed as transformations of statements using logical operations, or rules. As long as the rules are followed, reasoning is valid.

That this change in perspective in logic happened was consistent with the transition to abstract algebra that was underway in Britain (Cambridge) in the early 1800s, catalyzed by George Peacock, who made the case for the new and revolutionary symbolical approach to mathematics in his 730-page opus Treatise on Algebra (1830, see **How Algebra Became Abstract**).

> "The algebraists of the English school brought out first, between 1830 and 1850, the abstract notion of law of composition, and enlarged immediately the field of Algebra by applying this notion to a host of new mathematical objects: the algebra of Logic with Boole, vectors, quaternions and general hypercomplex systems with Hamilton, matrices and non-associative laws with Cayley." [Bourbaki/1991, p.14]

Hamilton, while a critic of the arbitariness of the new perspective nevertheless demonstrated its power with his creation in 1843 of the quaternions system for expressing 3-D geometry in algebraic form as a field of numbers. In each of these advances

> "[it was no longer] the meaning of the symbols involved in algebraic expressions but their laws of combination [that were of primary importance]" [Kleiner/2007, p.116]

3. Statement Logic as a body of knowledge: the abstract approach

Let us see what the modern symbolical approach to Statement Logic looks like.

Definition 1: Logical Truth We consider a class S of **statements** about a domain of discourse, and define a two-valued **truth function** $t : S \to \{0, 1\}$ whose values are typically interpreted as 1=true, 0=false.

The elementary statements in S can be about anything — logic is not concerned with whether a particular elementary statement is true or false (that determination is a matter for the discipline that studies the content). What logic is concerned with is the truth value of compound statements solely by virtue of the relation of elementary statements to each other. To separate subject matter content from logical content, replace all elementary statements with variables $p, q, r \in S$, so that the content is immaterial, and only the relations between distinct statements is captured.

In each of the following examples, see that you recognize the difference between subject matter content and logical content. Try translating each statement into a logical statement using statement variables p,q,r.

Statements about numbers:

1. 2+3=5 (true). 2+3=1 (true mod 4). These are elementary statements. Logical translation: p. q. No logical content.
2. The sum of two numbers is even if and only if either both numbers are even or both numbers are odd. Logical translation: p <-> q v r.

Statements about economics:

3. If demand has remained constant and prices have increased, then turnover must have decreased. (Hamilton/1988) Logical translation: p ^ q -> r.

Statements about politics:

4. If Jones is not elected leader of the party, then either Smith or Robinson will leave the cabinet, and we shall lose the election. (Hamilton/1988) Logical translation: ~p -> (q v r) ^ s.

Statements about topology:

5. c is a cube therefore c has 6 faces, 12 edges, and 8 vertices. Similarly "H a polyhedron therefore H has F faces, E edges, V vertices that satisfy $V - E + F = 2$ " (**Euler's**

theorem). Logical translation: p -> q. The content is about geometry. (The general statement about polyhedra which applies to an infinite number of solid bodies is a non-obvious theorem of topology which has at least **20 proofs**.)

Logical statements:

6. “NOT (p AND q)” is equivalent to “(NOT p) OR (NOT q)”, for any statements p,q. In symbols $\sim (p \wedge q) \equiv \sim p \vee \sim q$ (De Morgan). This general statement applying to an infinite number of statements p and q is a non-obvious theorem of logic.
7. “p IMPLIES q” is equivalent to “(NOT Q) IMPLIES (NOT P)”. In symbols $(p \rightarrow q) \equiv (\sim q \rightarrow \sim p)$. (Contraposition). Logical theorem.

Statements about sets:

8. $A \cap B = \emptyset$ (true depending on A,B disjoint). $A \cap B \subset A \cup B$ (true for any sets A,B). Logical translation: A ^ B = F. (conditional) A ^ B -> A v B. (true for any values of A,B, i.e. tautology)

Remarks:
A tautology is a logical statements that is true regardless of the truth values of the elementary statements that comprise it (see e.g. 3,7,8 above). Mathematical logic is a study of these statement forms, or *reasoning schematics*, and not (as is sometimes mistakenly believed) a study of truth *per se*. How to prove/disprove logical statements is covered in section 6 below.

The examples above introduced the five main logical operations:
^ (AND),
v (OR),
~ (NOT),
-> (IF/THEN),
<-> (IF & ONLY IF).

The set theory example is particularly interesting, as it turns out that elementary set theory has precisely the same algebraic structure as propositional logic, i.e. there is a one-to-one mapping between logical and set theory operations:
^ (AND) and $\cap$ (INTERSECTION),
v (OR) and $\cup$ (UNION),
~ (NOT) and $'$ (COMPLEMENT),
-> (IF/THEN) and $\subset$ (SUBSET), and
<-> (IF & ONLY IF) and = (EQUAL).

The key distinction is that set theory grammar has an additional membership operation $\in$ which distinguishes subjects and predicates within a statement, a capability that goes beyond statement logic grammar and puts Set Theory in between Propositional Logic and First Order Logic.

4. Statement Logic is finite and combinatorial

We have seen above the five main logical operations. **How many distinct logical operations are there?** What do these logical operations mean?

We can answer the first question without appealing to logical notions based solely on Definition 1 of truth function. (The second question is answered in section 5 below).

How many distinct logical operations are there?

Theorem 1 (Combinatorial Logic) The number of logical operations depends on the number of distinct inputs (elementary statements):

1 input p — 4 unary operations,

2 inputs p,q — 16 binary operations,

3 inputs p,q,r — 256 ternary operations

n inputs (p,q,r,…) $2^{(2^n)}$ n-way logical operation.

Proof:

We will count the number of function mappings (inputs to distinct outputs), based on Definition 1 of truth function.

Unary case: Input is p which has two possible input values (1,0), i.e. p is a 2-vector. A mapping must assign an output to each input, and the assignments are independent. So there are 2 choices for the first entry and independently 2 choices for the second for a total of 4 = 2*2 possibilities. We lay these out as a truth table:

4 unary operations

	1	2	3	4
p	T	p	~p	F
1	1	1	0	0
0	1	0	1	0

4 unary operations

Binary operation case: Input is p,q each of which is a 2-vector, forming a domain set of 4=2*2 possibilities. Against each of these, the truth function f(p,q) has 2 independent choices, giving a total of $2^4 = 16$ choices.

Domain for binary operation

		q	
		1	0
p	1	1,1	1,0
	0	0,1	0,0

p	q	f(p,q)
1	1	
1	0	
0	1	
0	0	

Domain for binary operation ha 4 elements

16 binary logical operations on a two-valued logic

	inputs		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	p	q	T	v	<=	=>	~^	p	q	<=>	∇	~q	~p	^	~=>	~<=	~v	F
1	1	1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	0	0
2	1	0	1	1	1	0	1	1	0	0	1	1	0	0	1	0	0	0
3	0	1	1	1	0	1	1	0	1	0	1	0	1	0	0	1	0	0
4	0	0	1	0	1	1	1	0	0	1	0	1	1	0	0	0	1	0

There are exactly 16 distinct binary operations on a two-valued logic (0,1). The enumeration order below is by increasing number of zeroes, i.e. the 16 columns = C(4;0) + C(4;1) + … + C(4;4) = 1+4+6+4+1.

Ternary: Input is p,q,r each of which is a 2-vector, forming a domain set of $8 = 2^3$ possibilities. Against each of these, the truth function as 2 independent choices, giving a total of $2^{2^3} = 2^8 = 256$ choices.

Domain for ternary operation

		r	
p	q	1	0
1	1	1,1,1	1,1,0
1	0	1,0,1	1,0,0
0	1	0,1,1	0,1,0
0	0	0,0,1	0,0,0

p	q	r	f(p,q,r)
1	1	1	
1	0	1	
0	1	1	
0	0	1	
1	1	0	
1	0	0	
0	1	0	
0	0	0	

8 inputs for ternary operation

General case: With n inputs, each 2-vector, the domain has 2^n possibilities, and against these the truth function assigns 2 independent choices, for a total of 2^{2^n} choices.

□

5. Assigning Logical Meaning to Operators

In the previous section, we saw the 16 logical operations derived combinatorially. **Does each logical operation have a logically relevant interpretation?**

Exploration:
Interpreting the truth function as 1=true and 0=false, each of the 16 binary operations above has a distinct logical meaning. Below we have re-arranged the operations to emphasize 8 primary operations followed by their "duals under negation" ($\sim$).

16 binary logical operations organized as 8 operations and their DUALS under negation

	inputs		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	p	q	T	p	q	^	v	=>	<=	<=>	F	~p	~q	~^	~v	~=>	~<=	∇
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	1	0	1	0	0	0	1	1	0	1	0	1
3	0	1	1	0	1	0	1	1	0	0	0	1	0	1	0	0	1	1
4	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1	0	0	0
			TRUE	id1	id2	and	or	if/then		iff	FALSE	not1	not2	nand	nor			xor
										~xor	~xor							~iff

8 primary operations and their DUALS under negation.

We now consider whether these truth function assignments map to our common sense understanding of the logical meaning of the the operations, paying especially close attention to the pairs OR/XOR and AND/IMPLY, which have had controversial histories.

The constants T and F:
(1,9) "T" ("ALWAYS TRUE") and "F" ("ALWAYS FALSE) are constant functions.
(2,3) p,q are identity functions, preserving the first, second input respectively. Also called projection or coordinate function.

(10,11) $\sim$ ("NOT") inverts its object, i.e. flips the value. Evident in $\sim T \equiv F$, $\sim p, \sim q$.
This covers 6 of the 16.

The connectives AND and OR, and their negations:
(4) $A \wedge B$ ("A AND B"), i.e. it is true when both p or q are true. Also called "conjuction".
(12) NAND is the negation.

"OR" is an interesting one as we have two notions, inclusive (which is the chosen convention) and exclusive (which was the original choice of Boole, but did not become dominent — see XOR section):
(5) $A \vee B$ ("A OR B") is of the inclusive variety, i.e. it is true when either or both p or q are true. Also called "disjunction".
(13) NOR is the negation.

Notice that $\vee$ is numerically a MAX function and $\wedge$ is numerically a MIN function. This will become interesting when pursuing multi-valued logics (Post algebras) and continuous logic (fuzzy logic).

The conditionals IF/THEN and WHENEVER, and their negations:
(6,14) $A \rightarrow B$ ("IF A THEN B"). Also called conditional, material implication or sufficient condition (A is sufficient for B).
(7,15) $A \leftarrow B$ ("A WHENEVER B"). Also called reverse implication or necessary condition (A is necessarily true whenever B is).

A closer look at sufficient and necessary conditions. Consider: rain -> clouds, or written differently, clouds <- rain. Translating literally we can say "rain implies clouds" or "clouds are present whenever it rains". In order words, "clouds are necessary but not sufficient for rain": there must be cloud whenever it rains, but it can be cloudy and yet not be raining. Consider another example: structured thinking is necessary but not sufficient for writing a good article. It is not sufficient because one still has to write out the article and polish it. It is necessary because no amount of polish can make an unstructured jumble of ideas intelligible without introducing structure. In symbols, "structured thinking" <- "a good article is produced", i.e. structured thinking whenever good writing. The first two rows of the truth table for Material Implication are identical to AND, and give rise to the IF THEN interpretation: when p ^ q and also ~(p ^ ~q), which by De Morgan's law is equivalent to p v ~q. But Material Implication goes beyond AND because of its two vacuous conditions:

- A false statement implies anything, even another false statement. (This is not the case with AND, which is false if either statement is false.)
- a true statement can be implied by anything, i.e. any statement, even a false one, can imply a true statement

This convention is visible in common speech in the retort to a false statement "sure, when pigs can fly", with the meaning that if pigs can fly (a false statement) then anything is possible.

This is significant because the choice of material implication in logic (vs. formal implication) asserts that there need not be any formal connection between p or q, i.e. they can be about mutually irrelevant subjects (see Tarski, 1946, Intro to Logic and the Methodology of the Deductive Sciences). For this reason **material implication (unlike formal implication)** cannot be considered CAUSE->EFFECT. (Example: the rain was blue yesterday -> I ate dragon for breakfast. This is true in material implication because a false statement implies anything.)

Note, conditional in the context of logic is about *implication*, and should not be confused with conditional *execution* (if 1 then jump to address X else jump to address Y).

The operations XOR and XNOR (also called IFF) that are related to OR:
(16) $A \nabla B$ ("A XOR B") is and OR of the exclusive variety, i.e. it is true when only p is true or only q is true but not both. Also called "exclusive or".
Application (Electricity): XOR is like a 2-way light switches in a corridor. Explain why.
Application (Computer Graphics): XOR, AND, OR are useful as **bit-masks** in graphics programming and specifically when manipulating bits. Exercise: $p \nabla T \equiv \sim p$ (p XOR T is bitwise logical negation, i.e. flips all the bits.

Historically, XOR was preferred by Boole because it has the property of + mod 2, meaning that "and" could be arithmetical multiplication and "or (XOR)" could be arithmetical addition (modulo 2). Even today, this is behind the convention that AB is A AND B (product notation) and A+B is A OR B (sum notation). Inclusive OR was chosen because its duality property with AND under negation (via De Morgan laws) is more convenient for a logical calculus than the forced arithmetic analogy.

(8) $A \leftrightarrow B$ ("A IFF B"). Also called biconditional and XNOR (observe that $\sim \nabla = \leftrightarrow$).

Note, IFF is shorthand for "if and only if". Why is this operation named IFF? A IF B is $A \leftarrow B$ which is the same as A WHENEVER B (see below). A ONLY IF is $A \rightarrow B$ which is the same as IF A THEN B, because in this case A can only be true if B is also true, since by the implication whenever A is true, then B must also be true.

Note, biconditional is operation of logical equivalence: both statements are always true together or false together. It is different from $\equiv$ because the latter is an assertion in our language of analysis, where biconditional is an assertion within statement logic itself.

This completes the explanation of the 16 binary operations of statement logic. We see that, yes, they do match the intuition of logical reasoning.
□

We now turn to two topics: proving logical identities, and reducing the number of operations without limiting expressiveness.

6. Truth Tables, Proof, and Computer Aided Verification

Truth tables are the "brute force" way of proving logical equivalence. They **first appeared in 1881** in the work of Christine Ladd-Franklin, a PhD student of C.S. Peirce [Peirce/1883], then in 1912 in one of Bertrand Russell's papers, and popularized in 1918 by Ludwig Wittgenstein, and 1921 by Emil Post (independently discovered).

Using truth tables is relatively straightforward for statements involving one or two variables (4 rows), but gets progressively more complicated with more variables (8 rows for 3 variables, 2^n rows for n variables).

An efficient way of writing truth table proofs is given in [Hamilton/1988, p.9], summarized below.

To show the De Morgan laws:

~(pvq) = ~p ^ ~q

~(p^q) = ~p v ~q

First write out the expression(s), then work from inside [1] outward [2],[3],etc. Beneath each statement, write its truth function, and beneath each operation, its result. Equality is shown if both sides ultimately generate the same truth vector, i.e. if [3]=[3] in this case. This is illustrated below.

De Morgan Law (OR)

~	(p	v	q)	=	~	p	^	~	q
0	1	1	1		0	1	0	0	1
0	1	1	0		0	1	0	1	0
0	0	1	1		1	0	0	0	1
1	0	0	0		1	0	1	1	0
[3]	[1]	[2]	[1]		[2]	[1]	[3]	[2]	[1]

YES

De Morgan Law (AND)

~	(p	^	q)	=	~	p	v	~	q
0	1	1	1		0	1	0	0	1
1	1	0	0		0	1	1	1	0
1	0	0	1		1	0	1	0	1
1	0	0	0		1	0	1	1	0
[3]	[1]	[2]	[1]		[2]	[1]	[3]	[2]	[1]

YES

Truth tables proving the two De Morgan laws, working from inside [1] outward [3]. Equality is shown if both sides generate the same final result [3]=[3] in this case.

From truth tables to a logical calculus.

By exploring relationships between different operations (symbolical algebraic exploration), we can identify a number of logical identities (tautologies) that lead to a logical calculus. After this, one can "derive" equivalent expressions through algebraic transformations, substitution, and the application of operations, following an algebraic approach. Underpinning all of this are the truth tables.

See **Appendix 2 for the list of key logic identities** and **Appendix 3 for algebra of logic** discussion. All identities which can be readily verified using truth tables or the Symbolic Logic Simulator (see next section).

7. Computer-aided verification of logical equivalence: a Symbolic Logic Simulator in Forth

How to make these explorations more efficiently? How to verify correctness of statements?

Write a computer program (domain specific language) for computer aided verification of logical expressions by automatically constructing the corresponding truth tables. For efficiency, we will use binary numbers to encode the truth table entries, and bit operations to compare columns. Write it as an **interactive domain specific language (DSL) coded in Forth or Ruby**. You will need to teach it the simplifying definitions below that allow reducing the number of logical operations to just three: $\sim, \wedge, \vee$. You can then use your the Logic Simulator to test/simplify complex sentences.

Symbolic Logic Simulator

Definitions: **logic2.fs** **logic3.fs** **logic4.fs**

Each definition file is 4 lines defining T,F,p,q

Core: **logic.fs**

Core of the program is 16 lines and defines logical operations and a bitfield viewer.

Total program length is 28 lines (=16 + 4×3) of **Forth code**.

Source code is given & explained in Appendix 1.

The code can be used without modification in **GFORTH** (0.7.0 for Windows **download from here**).

Let's see how the Symbolic Logic Simulator works in action.

First, we'll use it to evaluate De Morgan's law:

```
~(p v q)  =  ~p ^ ~q  (De Morgan)
```

Output:

```
Gforth 0.7.0, Copyright (C) 1995-2008 Free Software Foundation, Inc.
Gforth comes with ABSOLUTELY NO WARRANTY; for details type `license'
Type `bye' to exit

include logic2.fs
Symbolic Logic Calculator (2 variables) -- ready! ok

p q v ~ .. 0 0 0 1 b   ok
p ~ q ~ ^ .. 0 0 0 1 b   ok
eval Equivalent.  0 0 0 1 b ok
```

Evaluating De Morgan laws in Symbolic Logic Simulator, a 16-line program in Forth, that allows visual evaluation in 5 lines of code.

Note that Forth is a stack-based language, so operations use the Reverse Polish notation convention (e.g. 2 + 3 appears as 2 3 +, where + is an operator that takes its 2 arguments from the stack).

Switching to 3 variables:

```
p -> (q -> r)  =  ~p v ~q v r
```

Output:

```
include logic3.fs

Symbolic Logic Calculator (3 variables) -- ready!
 ok
p q r -> -> .. 1 1 1 1 0 1 1 1 b   ok
p ~ q ~ v r v .. 1 1 1 1 0 1 1 1 b   ok
eval Equivalent.  1 1 1 1 0 1 1 1 b ok
```

Evaluating p -> (q -> r) EQUIV ~p v ~q v r.
Note this is 3 variables, so we switch to 3 variable mode using "include logic3.fs"

Finally 4 variables:

```
(p^q) v (r^s)  =  (pvr) ^ (pvs) ^ (qvr) ^ (qvs)
```

Output:

```
include logic4.fs

Symbolic Logic Calculator (4 variables)-- ready!
 ok
p q ^ r s ^ v .. 1 1 1 1 1 0 0 0 1 0 0 0 1 0 0 0 b   ok
p r v p s v q r v q s v ^ ^ ^ .. 1 1 1 1 1 0 0 0 1 0 0 0 1 0 0 0 b   ok
eval Equivalent.  1 1 1 1 1 0 0 0 1 0 0 0 1 0 0 0 b ok
```

Evaluating (p^q) v (r^s) EQUIV (pvr) ^ (pvs) ^ (qvr) ^ (qvs).
Note this has 4 statement variables, so need to switch to 4 variable mode, using "include logic4.fs"

See **source code which is given & explained in Appendix 1**.

See **Appendix 2 for the list of key logic identities** and **Appendix 3 for algebra of logic** discussion. Identities have verification examples using the Symbolic Logic Calculator.

8. Expressiveness, Adequacy and the search for mathematical simplicity

The 16 logical operations on S are more than one typically has on a mathematical system. Consider that on $\mathbb{N}$ we have 6 operations: +,-,*,/,^,root, even though they do not become algebraically closed until numbers are expanded from the natural numbers $\mathbb{N}$ to the integers $\mathbb{Z}$ (closing +,-) to the rationals $\mathbb{Q}$ (closing *,/) to the reals $\mathbb{R}$ (closing powers and roots of positive numbers) to the complex numbers $\mathbb{C}$ (finally closing powers and roots for all numbers, but losing order in the process).

The 16 logical operations make mathematical logic a highly expressive language with many different ways of saying the same thing. But this comes with increased complexity: many more algebraic identities and transformation laws, e.g. how to simplify p->(q xor r). The proliferation of facts and identities, theorems and counter-examples creates clutter that obscures the clarity of the underlying situation. This leads to the questions:

Do we need all 16 operations? Could we get by with fewer? How few do we really need?

These meta-theoretic questions consider logic as a language.

A less expressive but adequate language would allow all things to be said (i.e. no loss of expressiveness), but would typically require longer sentences (more verbose because takes more symbols to express the same concept). The desire is for a streamlined, spare structure.

The following Propositions explore this possibility. Note that you can follow along with the derivations using truth tables or the Symbolic Logic Simulator. Or, for a challenge, you can stop reading now and attempt to discover adequate subsets on your own!

Theorem 1. (Combinatorial Logic) There are precisely 16 distinct logical operations on 2 statement variables p,q.
Proof: We have seen this above.

Proposition 2. Nine operations are adequate to represent the 16.
*Proof:*Nine operations are adequate to express all 16: 8 primary operations **T,p,q,^,v,->,<-,<->**, and **NOT**. Taking the NEGATION of the 8 gives the dual 8 operations:
~T = F
~p
~q

~^ = NAND

~v = NOR

~->

~<- ~<-> = XOR

We have derived all sixteen operations from the 9 (=8+1).

□

Proposition 3. Six operations are adequate to represent the 16.

Proof: The 6 operations are: **NOT, ^,v the constant T, and the variables p,q.**

We have already established that 9 operations are adequate (Proposition 2) so it remains only to show that the 6 can build up the nine, specifically three operations ->, <-, and <-> that are missing.

We have:

(A) p -> q = ~p v q = ~(p ^ ~q).

(B) p <- q = q -> p

(C) p <-> q = (~p v q) ^ (p v ~q)

Verification (SLS):

```
p q ->   p q ~ ^ ~   eval
p q ->   p ~ q v     eval
p q <->  p ~ q v  p q ~ v  ^  eval
```

This establishes ~, ^, v as adequate.

□

Proof:

(A) is directly from the definition of -> in the table of operations: p->q is true either when p is true (1) making q true (1), or when p is false (0) since a false statement implies any statment. The second part is by De Morgan applying double negation

(B) is from the definition of <- in the lookup table

(C) Proof:

C0. p xor q = (p^~q) v (~p^q) directly from description of xor in table of operations: p xor q is true when either p is true but not q, or vice versa. Both cannot be true or both false.

C1. p<->q = ~(p xor q) from duality identity, see Table 2 above.

= ~[(p^~q) v (~p^q)] (using C0)

= ~(p^~q) ^ ~(~p^q) (distributing negation through brackets using De Morgan duality)

= (~p v q) ^ (p v ~q). (distributing negation through each parenthesis using De Morgan)

Theorem 4. (Canonical Representation — Adequacy of AND, OR, NOT) Five operations are adequate to represent the 16.

Proof: We have already established that six operations are adequate (Proposition 3) so it is enough to show that T is obtainable from the remaining five **NOT, ^, v, and the variables p, q**.

We show:

T = p v ~p.

Verification (SLS):

```
include logic2.fs
```

```
T   p p ~ v  eval
```

□

Remark: This proposition shows the logical adequacy of Boolean Algebra, which is formed using AND, OR, NOT. Note that Boolean Algebra is the algebraic structure that underpins set theory as well as propositional logic.

Can we go smaller? Yes.

Proposition 5. (Adequacy of AND, NOT) Four operations are adequate to represent the 16.
Proof: We have established that five ops are adequate (Theorem 4), so it is enough to show that v is obtainable from the remaining four **NOT, ^, p, q**.
We show:
pvq = ~(~p ^ ~q)

Verification (SLS):

```
include logic2.fs
p q v   p ~ q ~ ^ ~   eval
```

□

Proof:
1. ~(pvq) = ~p ^ ~q. This is the De-Morgan duality expressing OR in terms of AND and NOT.
Then:
2. p v q = ~~(p v q) double negation is identity
= ~(~p ^ ~q) substituting 1

This is called the Conjunctive Normal Form.

Proposition 6. (Adequacy of OR, NOT), i.e. the Disjunctive Normal Form.
Left as an exercise.

Proposition 7 (Hilbert's Representation — Adequacy of IMPLY, NOT) The four operations **NOT, ->, p, q** are (also) an adequate set to represent the 16.
Proof: We have established that four ops are adequate (Prop 5), so it is enough to show that ^ is obtainable from these four.
We show:
p^q = ~(p -> ~q)

Verification (SLS):

```
include logic2.fs
p q ^    p q ~  ->  ~   eval
```

□

Proof:

1. ~(p->q) = p ^ ~q this is the definition of NOT-IMPLY from Table 1 (p true and q not)
Then:
2. p ^ q = ~(p -> ~q) by 1, substituting q for ~q

Note that Proposition 7 is the basis for David Hilbert's axiomatization of Statement Logic (see section 9 below).

Theorem 8. (Adequacy of NAND) NAND is entirely adequate (with inputs p,q) to represent the 16 logical operations.
Proof: The three operations allowed are **NAND, p, q**. We have established that four ops are adequate (Prop 5), so it is enough to show that NOT and ^ are each obtainable using only the combined operation NAND (denoted D — note D is recognized by the Symbolic Logic Simulator).

Claim:
p^q = (pDq) D (pDq).

Verification (SLS):

```
include logic2.fs
p q ^    p q D   p q D   D   eval
```

Proof:
1. pDq := ~(p^q) by definition of NAND (D).
2. ~x = x D x for any statement variable x.
Then:
3. p^q = ~~(p^q) (since double negation is identity)
= ~(pDq) (by 1, consuming one negation)
= (pDq) D (pDq) (by 2, substituting pDq for x)
□

Exercise. Derive the 13 other operations using p,q, NAND as the only operations.

Exercise. Are any other logical operations adequate by themselves (with p,q)?
Explore the conjecture that NOR is adequate (it is non-associative and duplication is negation, like with NAND).
Explore the conjecture that IMPLY is not (it is non-associative and idempotent).
Explore the conjecture that there are no others (all the rest are associative), i.e. only NAND and NOR are adequate as singletons.

Proposition 9. (Adequacy of NOR) NOR, p, q are (also) an adequate set.
Proof: We have shown that NAND, p, q are an adequate set (Theorem 8), so it is enough to show that NAND is obtainable from NOR, p, q.

Claim:
p D q = (~p NOR ~q) NOR (~p NOR ~q)

Verification (SLS):

```
include logic2.fs
```

```
p q D    p ~ q ~ NOR   p ~ q ~ NOR   NOR  eval
```

9. Axiomatization and Aesthetics: Hilbert's Axiomatization of SL

We have seen that there are multiple adequate sets for Statement Logic, including the remarkable fact that only 1 operation is required to adequately express all 16 operations of statement logic. NAND and NOR are each adequate sets of operations (see Props 7 and 8 above).

It also becomes clear that when there are multiple representations possible, it becomes interesting which representation is chosen and why.

Axiomatics is the business of choosing a set of primitive definitions, relations, operations and statements, and deriving the rest of the theory upon this foundation. The model for this style of organization is Euclid's Elements of Geometry.

The axiomatization of Statement Logic was achieved at the hands of David Hilbert. **Hilbert was a towering figure in mathematics, physics, and logic**. In 1899, he wrote Foundations of Geometry, in which he attempted to remedy the defects in Euclid's program (he required 20 axioms to do this, as opposed to the 5 from Euclid). This investigation led him to reflections on foundations of arithmetic and analysis, both of which had imperfections, the latter (from Dedekind) was complex requiring cuts and choice, and the use of the newly created set theory of Cantor. Hilbert was in correspondence with Cantor, who had created the set theory, and was familiar with the logical paradoxes associated with a naive construction of sets (the paradoxes were called Zermelo's paradox in Gottingen, before it was Russell's paradox).

This led him to search for an axiomatization of logic, as the underpinning of mathematics, and indeed, it is apparent from his famous address **"23 Problems for the 20th Century" (1900)**, that he viewed questions of logical certainty and decidability as of paramount importance for mathematics. Hilbert and his ideas were highly influential for modern mathematics and physics. His students were the physicist Hermann Weyl, axiomatic set theorist Zermelo, computer scientist John von Neumann, and mathematical physicist Richard Courant. His peers at Gottingen included the abstract algebraist Emmy Noether, and logician Alonzo Church. His assistants and research partners included logician Paul Bernays and physicist Eugene Wigner. (see **here**)

He went on to publish the Foundations of Physics (1915) in which he laid out the field equations of gravitation in axiomatic form (simultaneously with Einstein's publication of the field equations).

Starting in 1918, Hilbert began to publish his ideas on logic and the foundations of mathematics. He went on to establish the formalist program in 1920, and, in collaboration with Paul Bernays, published the Foundations of Mathematics (1927) in which these ideas, and the foundations of logic, are explored. (Four years later, in 1931, Godel would establish his incompleteness results which would show that the formalist program of Hilbert could never be achieved in totality.)

Hilbert's axiomatic foundation of Statement Logic took negation and material implication as the primary operators (see Proposition 7 above).

Hilbert's Program

10. Application to Digital Logic

The adequacy of NAND (Theorem 8) is pivotal for digital electronics. It allows building any logical circuit with just one type of gate, a significant improvement in the cost efficiency of manufacturing, as all ICs can then fundamentally be made from 1 logic gate with various interconnects. So while circuits may be easier to conceptualize and build using a more expressive set of logical operations, NAND allows for maximum frugality.

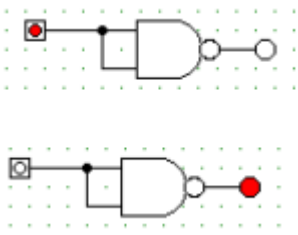
Example NAND chip: **IC 7400 is a Quad 2-input NAND chip** on 14-pins (or 4011B) for 18p ea. with minimum 10 pieces

Below the derivations of the 9 logic operations (see Proposition 2 above).

- First line is written in standard logic notation.
- All subsequent lines are translations using Forth (postfix) notation, ready for entry into the Symbolic Logic Simulator.
- Last line is the translation using additionally the Forth stack operation `dup` which duplicates the last entry on the stack. It is this translation that maps most directly to the circuit diagram.
- Circuit diagrams are given in **Digital Works** (**v2 is free to download, v3 is paid license**).

In the below, recall that D stands for NAND operation.

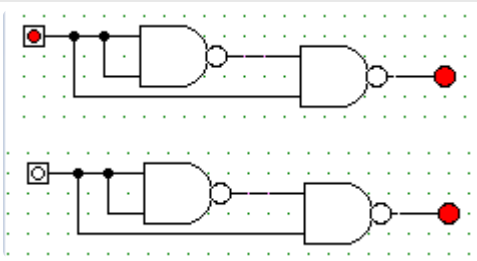
```
(0) p .. = 1 1 0 0 b
    q .. = 1 0 1 0 b

(1) p D p = ~p
p p D
p dup D .. = 0 0 1 1 b
1 NAND gate

q D q = ~q
q q D
q dup D .. = 0 1 0 1 b
1 NAND gate
```



NOT digital circuit using 1 NAND gate.
Test result:
1 0 = 0 1

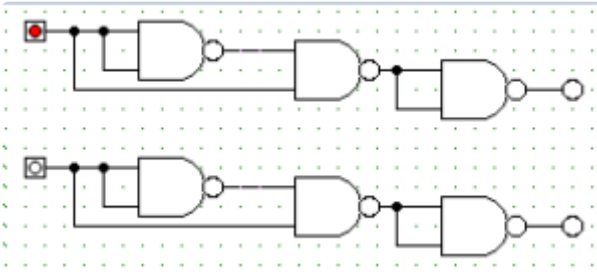
```
(2) p D ~p = T
p D ( p D p )
```

```
p p ~ D
p dup D p D .. = 1 1 1 1 b
2 NAND gates
```



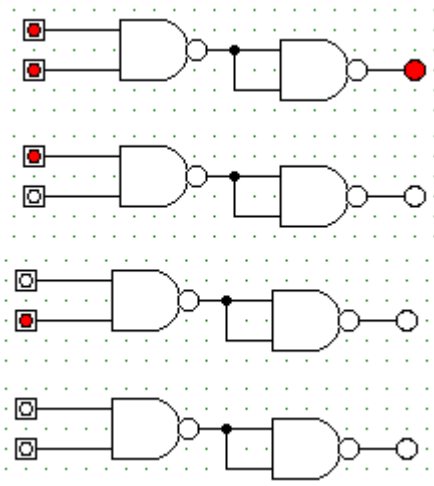
TRUE digital circuit using 2 NAND gates.
Test result:
1 0 = 1 1 (always on).

```
(3) ( p D ~p ) ~ = F
( p D ~p ) D ( p D ~p )
[ p D ( p D p ) ] D [ p D ( p D p ) ]
p p ~ D  p p ~ D D
p dup D p D dup D .. = 0 0 0 0 b
3 NAND gates
```



FALSE digital circuit using 3 NAND gates.
Test result:
1 0 = 0 0

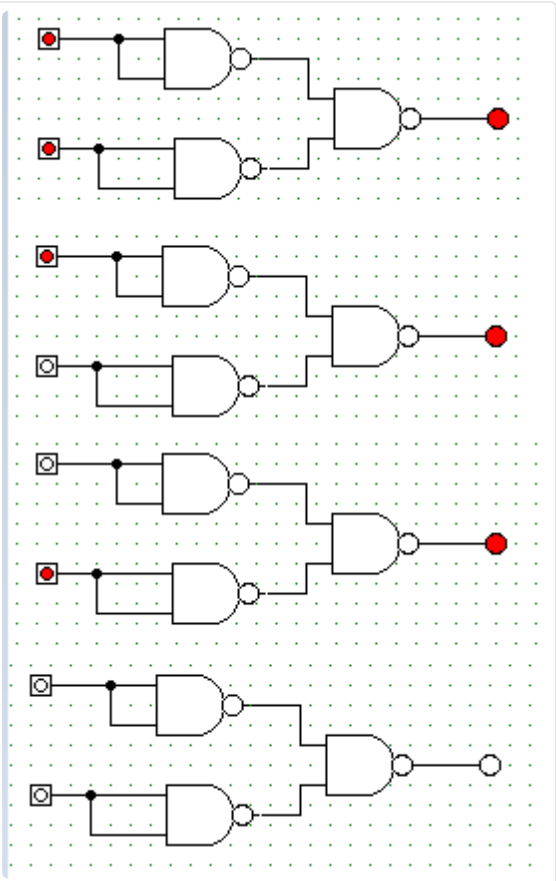
```
(4) ( p D q ) ~ = p ^ q
( p D q ) D  ( p D q )
p q D  p q D  D
p q D dup D .. = 1 0 0 0 b
2 NAND gates
```



AND digital circuit using 2 NAND gates.

```
Test result:
   1 1 0 0
   1 0 1 0
      =
   1 0 0 0
```

```
(5) ~p D ~q = p v q
( p D p ) D ( q D q )
p dup D  q dup D D .. = 1 1 1 0 b
3 NAND gates
```

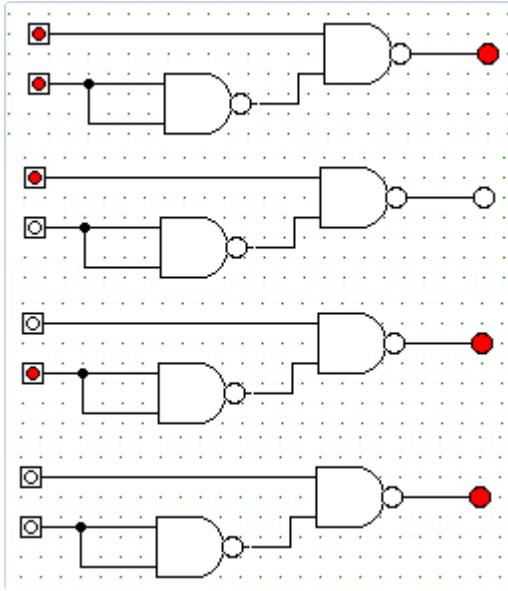


OR digital circuit using 3 NAND gates

```
Test result:
   1 1 0 0
   1 0 1 0
      =
```

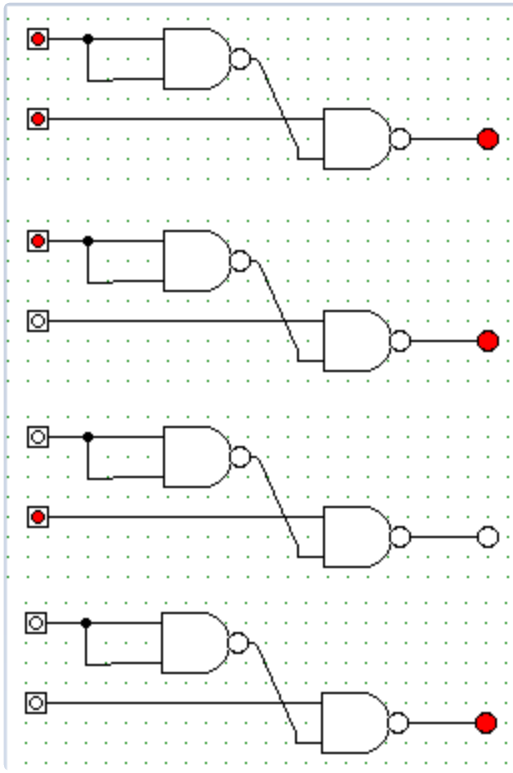
```
1 1 1 0
```

```
(6) p D ~q = p -> q
p q ~ D
q q D p D
q dup D p D .. = 1 0 1 1 b
2 NAND gates
```



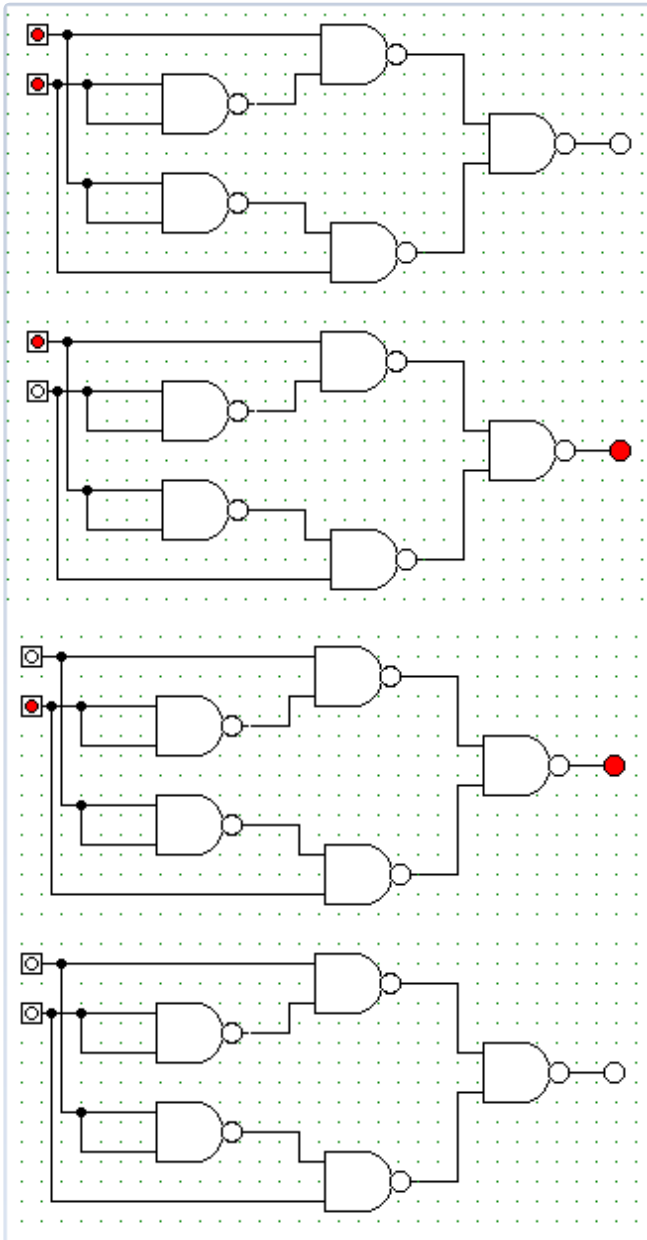
```
IMPLY digital circuit using 2 NAND gates.
Test results:
1 1 0 0
1 0 1 0
=
1 0 1 1
```

```
(7) q D ~p = p <- q
p p D q D
p dup D q D .. = 1 1 0 1 b
2 NAND gates
```



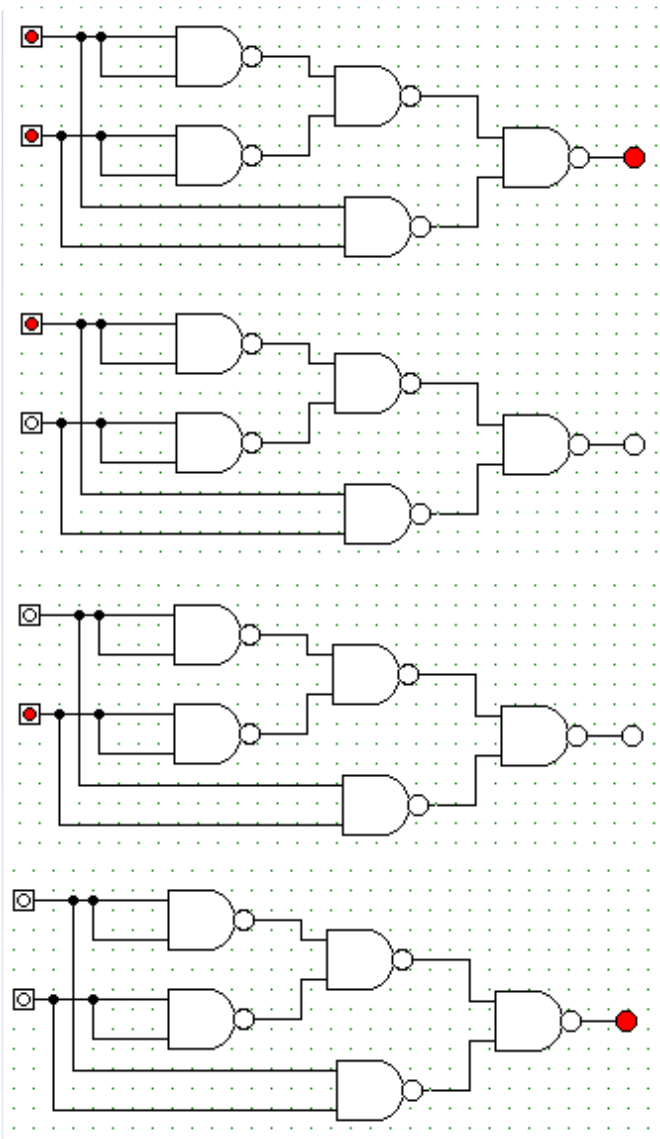
WHENEVER digital circuit implemented in 2 NAND gates.
Test results:
1 1 0 0
1 0 1 0
=
1 1 0 1

```
(8) (p -> q) D (p <- q) = p XOR q
(p D ~q) D (~p D q)
[p D (q D )] D [(p D p) D q]
q dup D p D  p dup D q D  D .. = 0 1 1 0 b
5 NAND gates
```



XOR digital circuit using 5 NAND gates.
Test result:
1 1 0 0
1 0 1 0
=
0 1 1 0

```
(9) ( ~p D ~q ) D (p D q) = p <-> q
p dup D  q dup D  D  p q D  D .. = 1 0 0 1 b
5 NAND gates
```



IFF digital circuit using 5 NAND gates.
Test results:
1 1 0 0
1 0 1 0
=
1 0 0 1

What might these circuits be used for? NOT, AND, OR have clear applications when combining signals. XOR and IFF are useful for various signal comparison (distinct or same). TRUE/FALSE are useful when one needs to maintain steady signal regardless of the variation. An example applications for IMPLY is to detect cases when the implication fails, e.g. NOT-IMPLY, as this will alarm (signal high) when p=1 but q=0, while all other cases will output signal low.

Historical Interlude III: From Boole to Shannon, from theory to computing

Claude Shannon encountered Boolean logic and Boolean algebra while studying at MIT in 1930, and realized the Boolean logic (Propositional Logic) could be implemented using electrical relays. The first application was at AT&T Bell Telephones (Shannon was one of the most **influential Bell Labs scientists**) using relays for telephone switching circuits. The next step was the jump from ON/OFF states to 1/0 states. And if numbers were involved, then counting was possible, in which case arithmetic was possible, and the fundamentals for computing were established. [Platt/2015] Note that Electronic computers would not arise till 1937 with Konrad Zuse's Z-1 followed by a spate of US and UK efforts in early computing -- ABC, Colossus, Harvard Mark, ENIAC, EDVAC, Manchester Mark, EDSAC, etc..

For computing history:

1. **the birth of Silicon Valley (1950)** with the departure of William Shockley from Bell Labs
2. the **world's first microprocessor (MP944, Ray Holt, for real-time calculations for the F-14 Tomcat**
3. **The birth of the microprocessor chip (1968)**

Challenge problem How to know if an entirely NAND based digital circuit design is the one that uses the fewest NAND gates? (Design an algorithm or procedure to test for this optimality.) (Hint: K-map, or Karnaugh map is an algorithm for algebraically simplifying expressions with 3 or more logical variables. See [Clements/2000]).

Dual-input NAND chips
74804 is a Hex 2-input NAND chip on 20 pins. This chip with 6 NAND gates on it is the perfect size for wiring XOR and IFF logic which take 5 gates each. It is also the perfect size for wiring two of the 3-gate circuits: FALSE and OR, or three of the 2-gate circuits: TRUE, AND, IMPLY, and WHENEVER.

IC 7400 is a Quad 2-input NAND chip on 14-pins (or 4011B) for 18p ea. with minimum 10 pieces. This chip with 4 NAND gates is readily available in dual-inline-pin (DIP) package from hobby supply stores. While it is a decent size for implementing 3-gate and 2-gate circuits, it needs 2x chips to implement the 5-gate circuits XOR or IFF.

Multi-Input NAND chips
As we have seen above, NAND is non-associative, so multi-inputs cannot be viewed as cascaded NANDs

```
( p D q ) D r or p D (q D r)
```

but rather as negation of a multi-AND. By De Morgans law this is multi-OR of negatives, e.g.
For three-input NAND:

```
~ ( p ^ q ^ r ) = ~p v ~q v ~r
```

For four-input NAND:

```
~ ( p ^ q ^ r ^ s) = ~p v ~q v ~r v ~s
```

Similarly for the 8-input chip.

7410 is 3-input NAND chip, **7420 is 4-input NAND chip**, **7430 is an 8-input NAND chip** (28p)

Appendices

Appendix 1: Source code for Symbolic Logic Simulator

Symbolic Logic Simulator
Definitions: **logic2.fs** **logic3.fs** **logic4.fs**
Each definition file is 4 lines defining T,F,p,q

Core: **logic.fs**

Core of the program is 16 lines and defines logical operations and a bitfield viewer.

Total program length is 28 lines (=16 + 4x3) of **Forth code**.

Source code is given & explained in Appendix 1.

The code can be used without modification in **GFORTH** (0.7.0 for Windows **download from here**).

Variables:

We start with the logical variables. These are coded in binary. T ("tautology") determines whether the variable length operations (e.g. ~ and bits) are 4-, 8-, or 16-bits, allowing accordingly the use of use 2,3, or 4 statement variables.

For two logical variables (4-bits), use:

```
(   logical variables  )
:  T   $F  ;  \ 1111b    T determines bit-width of calculations, here 4-b
:  F   $0  ;  \ 0000b
:  p   $C  ;  \ 1100b
:  q   $A  ;  \ 1010b
```

For three logical variables (8-bits), use:

```
:  T   $FF  ;
:  F   $00  ;
:  p   $CC  ;
:  q   $AA  ;
:  r   $F0  ;  \ 1111 0000b
```

For four logical variables (16-bits), use:

```
:  T   $FFFF  ;
:  F   $0000  ;
:  p   $CCCC  ;
:  q   $AAAA  ;
:  r   $F0F0  ;
:  s   $FF00  ; \ 1111 1111 0000 0000b
```

Core:

Next we code the 5 primary logical operators: NOT, AND, OR, IMPLY, and IFF (biconditional), and 2 auxilliary operators: NAND and WHENEVER. XOR is already built in.

```
(   logical operators     )
:  ~    ( p -- ~p ) T xor ;          \ NEGATION takes bitfield size from
:  ^    ( p q -- p^q ) and ;
:  v    ( p q -- pvq ) or ;
:  ->   ( p q -- p->q ) swap ~ v ;   \ IF/THEN
:  <->  ( p q -- p<->q ) xor ~ ;     \ IFF
```

```
(    auxilliary definitions )
: & ( p q -- p NAND q ) ^ ~ ;        \ NAND
: <-  ( p q -- q p -> )  swap -> ;  \ WHENEVER
```

Visualization:

Finally, to be useful, we provide words to visualize results and inspect examples.

```
(   viewing bitfield of a number  - determines number of bits from T )
: .bit  ( n -- ) 0> if 1 else 0 then . ;
: init-mask  T 2 / ~  ;  \ initial mask is $8=1000b, $80=1000000b, or $80
: bits  (  bitfield  --  )
    init-mask begin      \  loop through bits, pick them off, and print
        2dup and .bit
        1 rshift dup 0= until
    2drop ." b" ;
: .. ( bf -- bf ) dup bits 2 spaces ;
: eval ( r1 r2 -- ) 2dup = if ." Equivalent.  " bits drop else ." Differe
```

Appendix 2. Key Identities of Propositional Logic

Duality Identities

These identities express how negation ~ interacts with logical operators.

1. ~~p = p (Double Negation), aka ~ is an involution of order 2
2. ~(p^q) = ~p v ~q (De Morgan AND), aka NAND product-to-sum p D q = ~p v ~q
   ~(pvq) = ~p ^ ~q (De Morgan OR), aka NOR sum-to-product p NOR q = ~p ^ ~q
3. ~(p -> q) = p ^ ~q (Not-Imply)
4. p -> q = ~p <- ~q (Contraposition)

Algebraic Properties

5. ~ is an involution of order 2 (see Double Negation above)
6. ^, v are commutative and associative
   xor, <-> are commutative and associative
7. ^, v are bi-distributive, each over the other
   ^ distributes over XOR, but not vice versa, i.e. ^ is like * and xor like + (modulo-2)
8. ^, v are related by duality (see Morgan Laws)
9. ->, <- are non-commutative and non-associative
10. -> distributes over itself

Verification in SLS:

```
p q r -> ->    p q ->  p r ->  ->   eval   \ Equivalent.   1111 0111
```

Canonical Form Identities

These identities express ->, <->, xor, NAND, and NOR in terms of the canonical set of connectives ^, v, ~ (see Theorem 4 above).

11. p -> q = ~p v q
    p <-q = ~q v p Verification in SLS:

```
p q ->   p ~ q v   eval   \ Equivalent.   1011b
p q <-   q ~ p v   eval   \ Equivalent.   1101b
```

12. p <-> q = (p -> q) ^ (p <- q) p <-> q = (~p v q) ^ (~q v p)
    p <-> q = ~(p xor q)
    p <-> q = (p ^ q) v (~p ^ ~q)

    Verification in SLS:

```
p q <->  p q ->  p q <-  ^   eval   \ Equivalent.   1001b
p q <->  p ~ q v  q ~ p v  ^  eval  \ Equivalent.   1101b
p q <->  p q xor ~  eval           \ Equivalent.   1101b
p q <->  p q ^  p ~ q ~ ^  v  eval  \ Equivalent.   1101b
```

13. p xor q = (~p ^ q) v (~q ^ p)
    p xor q = ~((p ^ q) v (~p ^ ~q))
    p xor q = ~(p <-> q)
    p xor q = ~(p -> q) v ~(p <- q) Verification in SLS:

```
p q xor  p ~ q ^  q ~ p ^  v    eval    \ Equivalent.   0110b
p q xor  p q ^  p ~ q ~ ^  v ~   eval   \ Equivalent.   0110b
p q xor  p q <->   ~  eval              \ Equivalent.   0110b
p q xor  p q -> ~  p q <- ~  v   eval   \ Equivalent.   0110b
```

14. p NAND q = ~(p^q) (NAND law: sum-to-product, see De-Morgan)
    p NOR q = ~(pvq) (NOR law: product-to-sum, see De-Morgan)

Appendix 3. Algebraic Aspects of Statement Logic

1. It is remarkable that logical thought, human reasoning, can be modeled by a symbolical algebra, i.e. a set of rules of manipulation on statements that is closed with respect to these logical operations. These facts can be proven using truth tables or derivation, and explored using the **Symbolic Logic Simulator**.
2. We have seen that of the 16 binary logical operations, we can identify increasingly smaller adequate subsets (Propositions 2-7). It is remarkable that these adequate subsets have very different algebraic properties. The most regular algebra is formed from the three operators (AND, OR, NOT) that define a Boolean algebra. Boolean algebras have nice properties: they are an algebraic group with respect to AND and OR. Each of these operations is commutative, associative, and indeed bi-distributive. This means that the Boolean algebra is almost a ring but not quite.
3. Hilbert's axiomatization uses IMPLY and NOT, which form a non-associative algebra, more

similar to the octonions than the integers.

4. The minimalist generating set consisting of the single operator NAND alone forms an algebraic structure that is closed but non-associative.

Algebra of ^ and v

The logical connectives AND and OR are commutative and associative just like + and * in algebra.

```
\ commutative
include logic2.fs
p q ^    q p ^    eval   \ Equivalent.
p q v    q p v    eval   \ Equivalent.

\ associative
include logic3.fs
p q r ^ ^     p q ^ r ^    eval   \ Equivalent.
p q r v v     p q v r v    eval   \ Equivalent.
```

Interestingly, they are EACH distributive over the other, unlike algebra where only * distributes over +.

```
\bi-distributive
include logic3.fs
p q r v ^     p q ^  p r ^  v   eval      \ p ^ ( q v r ) = (p ^ q) v (p
p q r ^ v     p q v  p r v  ^   eval      \ p v ( q ^ r ) = (p v q) ^ (p
```

They also EACH have idempotent index laws, i.e. squaring yields the same value (identity), unlike algebra where neither + nor * are idempotent.

```
p p ^  p  eval  \ Equivalent.
p p v  p  eval  \ Equivalent.
```

The arithmetic analogy is strongest when "or" is taken to be the exclusive variety XOR, then the analogy is exact modulo-2. And indeed, rather than AND and OR being bi-distributive, AND and XOR are simply distributive exactly as in the integers:

```
\ distributive, not bi-distributive
include logic3.fs
p q r v ^     p q ^  p r ^  v   eval      \ p ^ ( q v r ) = (p ^ q) v (p
p q r ^ v     p q v  p r v  ^   eval      \ p v ( q ^ r ) = (p v q) ^ (p
```

Duality of ~ over ^ and v

Duality in logic refers to operators that are linked by negation.

$\wedge$ and $\vee$ are duals, linked by $\sim$ through the DeMorgan distributive laws.

```
\ De Morgan distributive laws
```

```
include logic2.fs
p q v ~    p ~ q ~ ^  eval     \ ~ (pvq) = ~p ^ ~q  \ Equivalent.
p q ^ ~    p ~ q ~ v  eval     \ ~ (p^q) = ~p v ~q  \ Equivalent.
```

∇ and $\leftrightarrow$ are duals, linked by $\sim$ by definition: p XOR q = ~(p <-> q).

```
p q xor   p q <-> ~   eval    \ p XOR q = ~(p <-> q)   \ Equivalent.
p q <->   p q xor ~   eval    \ p <-> q = ~(p XOR q)   \ Equivalent.
```

$\rightarrow$ and $\leftarrow$ are duals, linked by $\sim$ through contraposition: p -> q = ~p <- ~q

```
p q ->   p ~ q ~ <-   eval    \ p -> q = ~p <- ~q   \ Equivalent.
```

$\rightarrow$ and $\wedge$ are duals, linked by $\sim$ through denial: ~(p -> q) = p ^ ~q

```
p q -> ~   p q ~ ^  eval  \ ~(p -> q) = p ^ ~q    \ Equivalent.
```

NAND, NOR, and conditional implication IMPLY and WHENEVER are non-associative

Algebra of NAND

The fact that NAND is adequate of itself to generate all the operations of logic, is due to it being both NON-ASSOCIATIVE and an INVOLUTION (of index two), i.e.

```
p p D    p ~   eval   \   p D p = ~p   \ Equivalent.
```

But it is not quite negation, as further powers oscillate between T and ~p

```
~p D p = T
T D p = ~p
etc.
```

If we were to list the operations on a 16-sided polygon, then the dual operations (two's complement numbers) are linked by NAND.

Algebra of NOR

NOR has similar property as NAND (NON-ASSOCIATIVE INVOLUTION). Hence NOR is expected to also be adequate (Exercise for the reader).

Algebra of IMPLICATION

IMPLY while NON-ASSOCIATIVE is IDEMPOTENT (has index 1). It requires NEGATION to form an adequate set. It is similar to AND, so it should not surprising that {IMPLY, NOT} form an adequate set since we know that {AND, NOT} do as well (see Proposition 5 above).

Appendix 4. Non-associativity

What common mathematical operations are non-associative?

1. Subtraction: 5-2-3. (5-2)-3 = 0. 5-(2-3)=6.
2. Power: 2^2^3. (2^2)^3 = 64. 2^(2^3)=256.
3. Division: 8/3/2. (8/3)/2 = 1.333. 8/(3/2)=5.333.
4. Imply: p->q->r. (p->q)->r = 1111 0100b. p->(q->r) = 1111 0111b.
5. NAND: p !& q !& r. (p!&q)!&r = 1000 1111b. p!&(q!&r) = 1011 0011b.
6. NOR: p !v q !v r. (p!v q)!v r = 0000 1110b. p!v(q!v r) = 0011 0010b.

Considering the number of non-associative operations, it is actually quite surprising that the familiar ones from algebra are indeed associative!
Addition
Multiplication
Conjuction (AND)
Disjunction (OR)

References

Books on Logic and its History

1. [Hamilton/1988] **Logic for Mathematicians**, A.G. Hamilton, 1988. *Provides a systematic treatment of mathematical logic (including first order logic and proof theory) with exemplary economy for a logic text (just 200 pages!). Highly recommended.*
2. [Walicki/2011] **Introduction to Logic**, Michal Walicki, 2011 *Extensive history section. Provides dense treatment of syntactic logic through first order logic.*
3. [Ebrahim/2020a] **The Rise of Mathematical Logic: from the modeling of reasoning to a search for the foundations of mathematics**, Assad Ebrahim, March 2020, Mathematical Science & Technologies
4. [Pycior/1981] **George Peacock and the British Origins of Symbolical Algebra**, 1981, Historia Mathematica 8, 23-45
5. [Peckhaus/2004] **Schroder's Logic**, 2004, by Volker Peckhaus, Handbook of the History of Logic, Vol 3. The Rise of Modern Logic: From Leibniz to Frege, 2004, pp.557-609. Additionally [**Peckhaus/1999**],
6. [Peckhaus/1996] **Axiomatic method and Schroder's algebraic approach to logic**, 1996
7. [Peckhaus/2002] Hilbert's Paradox, [Peckhaus/2005] Pro and Contra Hilbert: Zermelo's Set Theories, [Peckhaus/2004] Paradoxes in Gottingen. [Peckhaus/2003] **Pragmatism of Hilbert's Program**
8. [Peckhaus/2012] **The Reception of Leibniz's Logic in 19th Century German Philosophy**, 2012
9. [Bourbaki/1991] **Elements of the History of Mathematics**, Nicholas Bourbaki, 1991 (French original), Springer, 1994, 1998, 2008 republications
10. [Kleiner/1986] **The Evolution of Group Theory: A Brief Survey**, Israel Kleiner, 1986, Mathematics Magazine, Vol.59, No.4, October 1986
11. [Kleiner/2007] **A History of Abstract Algebra,** Israel Kleiner, 2007, Birkhauser

Sourcebook on Logic (Primary Sources)

12. Aristotle/360 BCE, **the Organon** is the combined set of his six writings on reasoning: Topics, On Interpretation, Prior Analytics, Posterior Analytics, Categories, On Sophistical Refutations.
13. **Theophrastus**, the successor of Aristotle, who strengthened proof theory, and was the first

to describe the principles of **implication (modus ponens)** and **contraposition (modus tollens)**. These are the 1st and 3rd of Hilbert's 3 axioms for statement logic.

14. Leibniz/1670 **works on logic (4 areas)**
15. Leibniz/1685, Two Appendices by Leibniz, in [Lewis/1918], pp.373.ff
16. [Peacock/1830] **Two page summary** of the 30pp original preface to **A Treatise on Algebra** (**ebook**), George Peacock, 1830. The 2nd edition was completely rewritten in 2 vols separating **vol 1. Arithmetical algebra (1842)** from **vol 2. Symbolical algebra (1845)** (**ebook**). **Reviews were still favourable 100 years later! (1942)**
17. [Boole/1847] **The Mathematical Analysis of Logic, Being an Essay Towards a Calculus of Deductive Reasoning**, George Boole, 1847, MacMillan and Co. Boole's work would lead to a Boole-Pierce-Schroder tradition of Algebra of Logic that extended then to Tarski/1941. (Note that Schroder developed his parallel work independently of any knowledge of Peacock or Boole.)
18. [DeMorgan/1847] **Formal Logic, or the Calculus of Inference, Necessary and Probable**, Augustus De Morgan, 1847, Taylor & Walton
19. [Boole/1854] **An Investigation into the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities**, George Boole, 1854, MacMillan and Co.
20. [Jevons/1864] **Pure Logic (1864), and The Substitution of Similars (1869)**, William Stanley Jevons; **Overview of work on logic**, 1864
21. [Peirce/1870,1897] **The Logic of Relatives (1897)**, The Monist, Peirce. **Overview of work on logic**.
22. [Peirce/1880] **On the Algebra of Logic, 1880**, C.S. Peirce
23. [Peirce/1885] **On the Algebra of Logic: Contribution to Notation (1885)**
24. Cantor/1874 work on **infinite set theory**
25. [Schroder/1877] work on algebra of logic (Operationskreiss), expanding Boole's system. Subsequently, Whitehead/1898, and Lewis/1918 would base their work on his 1877 work. (Note that Schroder developed his parallel work independently of any knowledge of Peacock or Boole.)
26. Frege/1879, **Conceptual Notation**, 1879
27. Frege, **List of Papers, in English**
28. [Venn/1881] **Symbolic Logic**, John Venn, 1881, MacMillan & Co., refined the representation of logical operations anticipated by Leibniz (see above), Euler (Euler rings), and Hamilton (1861)
29. The first truth table was introduced by Christine Ladd-Franklin's 1881 paper in Peirce/1883 **Studies in Logic**.
30. [Schroder/1890] 4-volume work on the Algebra of Logic: 1890, 1891, 1895, last volume published posthumously in 1905.
31. [Hilbert/1895] **His bibliography**. Hilbert's Foundations of Geometry had underlined the need for rigorous axiomatic, consistent basis for reasoning and the foundation of arithmetic to underpin the foundation of analysis (Dedekind) which in turn was needed to underpin the foundations of arithmetic. [**Ewald/Sieg/2013**] Hilbert's Lectures. [**Mancosu/1999**]. [**Sieg/2014**], 1917-1922 program, [Zach/2006] **Hilbert' Program Then and Now**
32. [Lewis/1918] **A Survey of Symbolic Logic, with two extracts from Leibniz**, C.I. Lewis, 1918
33. [Bernays/1918] Bernays/Hilbert results on completeness of propositional logic, and other

areas. See [**Zach/1999**]

Applications of Logic

34. [Clements/2000] **Principles of Computer Hardware (3rd ed.)**, Alan C. Clements, 2000. This version comes with **Digital Works**, an exceptional digital logic simulator allowing building real digital logic circuits and IC chip designs.
35. [Platt/2015] **Make: Electronics, 2nd Edition**, by Charles Platt, Maker Media, 2015, £17 *This is an excellent hands-on, learning-by-discovery primer for the new electronics enthusiast: burn things out, mess things up; that's how you learn. Contains an interesting chapter on Boolean Logic, including history going from theory to relays and computing.* See **Electronics Supplies** to get started.
36. [Ebrahim/2010] **Assembly Language for Digital Logic Design & Embedded Systems Development**, Assad Ebrahim, May 2010, Mathematical Sciences & Technologies
37. [Ebrahim/2020b] **Forth, Lisp & Ruby: languages that make it easy to write your own domain specific language (DSL).**, Assad Ebrahim, January 2020, Mathematical Sciences & Technologies
38. [Feferman/2003] **The Number Systems: Foundations of Algebra and Analysis**, Solomon Feferman, 2003. This contains one of the most important applications of logic -- its use in building up the modern number systems of mathematics and proving their various properties.
39. [Feferman/1998] **In the Light of Logic**, Solomon Feferman, 1998
40. [Heske/1996] **Fuzzy Logic for Real World Design**, Ted & Jill Heske, 1996

Please leave a comment! (your thoughts, corrections, suggestions)

« **Artifact analysis: proto-cuneiform during the first 1000 years of writing and mathematics** **Home** **How Algebra became abstract: George Peacock & the birth of modern algebra (England, 1830)** »