SCIENCE & TECHNOLOGY

A simple and adaptable logic

a simplest logic of all

by M. Soper, MA

A PROBLEM with standard logic as implemented in the 7400 series of integrated circuits is that, since the approach is functional, there is a natural tendency to organize any system into levels—a kind of nested hierarchical structure results. But suppose the requirement is more unilevel and free-wheeling—suppose true statements are chasing each other round in a loop in a system intended to have a use as associative memory in artificial intelligence applications: some expressed positively: 'all ships are intended for use in water' and some negatively: 'no normally conducting useful circuits are short circuits': then any hierarchical approach to analysing what can follow from statements like these will confound the flexibility of the system by deciding the functional pattern of logical analysis at the outset. The logic suggested in this article, being relational, not functional, gets round this.

Building the new system

The system used here is very simple:

EVEN is true ODD is false

and the relation is, for three integers $a\ b\ c$:

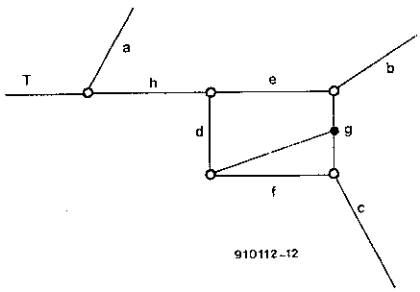
$a+b+c$ is odd abc is even

Here, addition and multiplication are carried out as usual.

To show that this simplest logic is what is claimed we must show that all logic can be done by this relation. Consider the notation $((a,b,c))$ used to indicate that numbers a,b,c obey this rule: $a+b+c$ is odd and abc is even. We must show that this relation is adequate for all logic and this can be done if the existence of the functions (relations) NOT and OR (first of all NOT) must be shown to exist:

- $((a,b,2))$ is equivalent to a=NOT b, since one of a,b is even and the other is odd. In fact, the relation is exactly two of a,b,c are even numbers.
- to show we can make OR: consider $((a,h,2))$, $((h,d,e))$ $((e,b,g))$, $((g,f,d))$, $((f,g,c))$—these entail that a OR $b=c$ is TRUE.

Here is a diagram of this logic circuit where circles denote the relation:



The proof is thus:
Let a,b be odd: then h is even and, since b is odd, from $((e,b,g))$ we have e and g is even; then $((h,d,e))$ implies d is odd so that from $((f,g,d))$ f and g are even and, since f and g are even, from $((f,g,c))$, c is odd.

Let a be even and b odd, then as before, e and g are even, but in this case h is odd since a is NOT h. Since h is odd, from $((h,d,e))$, d and e are even now since $d\ g$ are even, f is odd so that $((f,g\ c))$ implies c is even.

Let a be odd and b be even: then h is even: $((h,d,e))$. $((e\ b,g))$ imply e is even and d,g odd; OR e is odd and d,g are even in either case because $((g,f,d))$ f is odd. Moreover f is odd and $((f,g,c))$ implies c is even.

Let a be even and b be even, then h is odd and from $((h\ d,e))$, d and e are even; from $((e,b,g))$ and e,b even, we have g odd: then from $((f,g,c))$, c is even.

From all this we now have the table

a	b	c
odd	odd	odd
odd	even	even
even	odd	even
even	even	even

which is the correct pattern for the logic function OR with even meaning true, and odd, false.

Why should this logic be preferred?

There are two reasons for this logic to be preferred: the first is that no easy logic could be simpler, and the second is the fact that a relation is not hierarchical and can thus operate on converging data streams to create new true and false statements without any hierarchical structure being necessary. One can imagine triples of related strings, each one of which represents a 'proposition', moving through a data network like 'trains with three carriages' and interacting with other trains at junctions to produce more trains of statements on the data lines. Each junction can become a source of proliferation and segments on the 'lines' when triples of propositions are formed in this way can 'float about' on the communication lines of a distributed processing system preceded by a special symbol or token (like three carriage trains running freely through a number of junctions). Thus for an artificial intelligence system, each triple can be compared at each junction according to interference rules like these:

$((A,D,E))$, $((A,B\ C))$ $((B,C,D))$ implies $((B\ C\ \overline{E}))$ and E false;

$((A,D\ T))$, $((B\ D\ X))$, NOT$((A\ D,B))$ implies $((A\ \overline{B}\ T))$;

$((B\ C\ E))$. $((A\ B\ C))$. $((C\ D\ E))$ $((A\ D\ I))$ implies $((B\ E\ F))$;

$((A\ B\ C))$, $((C,D,E))$, $((A\ D\ F))$ NOT$((B,C\ E))$ implies $((\overline{B}\ E\ \overline{F}))$

$((A,B,C))$, $((B\ C\ D))$. $((B\ C\ E))$ implies NOT$((A\ D\ E))$

Liberation from standard binary functional logic

The most useful rules for generating a few of the many possible relations are:

$((A\ D\ T))$ $((B\ D\ X))$ iff A v B is true where X has any possible value and

$((A\ B\ C))$. $((B\ C\ D))$ iff $A=D$

Note that $((A\ B\ T))$ iff $A=\overline{B}$ and

$((A\ B\ X))$ implies A or B true.

These relations, therefore, can generate all standard logic. But we need, perhaps to

generate from any propositions ones that are true and false. Here is one such method:

$((A\ B\ C)), ((B\ C\ D)), ((A\ D\ E))$ implies E is false, and

$((A\ B\ C)), ((B\ C\ D)), ((A\ D\ E))\ ((U\ V,E))$ implies $U\ V$ true

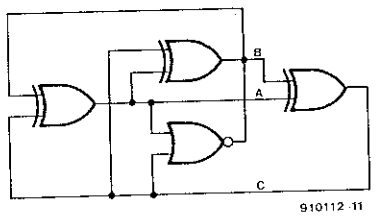
Thus the hierarchical structure caused by functional approaches is not necessary

Continuity with the old system

Stripped down to bare essentials, the new system of logic expressed as a table looks like this

a	b	c
F	F	*
F	T	T
T	F	T
T	T	F

which is the function table for exclusive OR (XOR) with one line deleted. This can be implemented by the following logic circuit:



This is cumbersome but it shows that the simplest logic we are dealing with here can actually be built. In practical terms, the reason for using relational logic, and for perhaps building such units as available ICs, is the fact that one relational logic circuit can easily perform more than one task, because lines are not specifically inputs and outputs.

All lines can be connected as outputs—impossible or paradoxical networks simply result in unstable or partially competing output circuits, so that the units built for this purpose should be output protected. In practice, lines to have a use as inputs should be fed by high impedance buffers (non-inverting) or, more simply, fed by some impedance. Thus all logic functions and naturally also oscillators (three-phase) and other derived circuits, can be implemented with the use of buffers and our symmetric logic element. Hence continuity with the old system is established.

Some higher versions of the logic

There are infinitely many relational logics, of which ours is the simplest. The next interesting logic is a six-logic, where the consecutive lines coming out have this cyclic pattern of true and false values: (TTFTFF), and a 10-logic, where the consecutive lines are (TTTFFFTFTF) in a cyclic pattern. In the six-logic, for readers with the patience to work this out, $(CDABMN)(MNFCGH)$ is equivalent to $C=A$ NAND B and in the 10-logic, the pattern $(T.A\ B.\ C.)$ forms $C=A$ NOR B. These ten lines form a kind of device (or insect) with ten radially disposed legs. They are based on what is know as the *Theory of Quadratic Residues*.

Transformation model of the relation

For some purposes, a transformational model is preferable. When this is the case the following model can be used—thus

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} c \\ a \end{bmatrix}=\begin{bmatrix} b \\ c \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} b \\ c \end{bmatrix}=\begin{bmatrix} a \\ b \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}=\begin{bmatrix} c \\ a \end{bmatrix}$$

where all the numbers are modulo 2—that is, throwing out 2s and taking only the remainders, 0 or 1.

Note the fact that $a,b\ c$ here are not allowed all to be zero; we could remove this condition by adding one more row and column to the three matrices, but instead we merely note $abc <> 0$. One interesting fact is that this matrix can be square-rooted; the square root is

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

The original matrix could generate the Fibonacci series from $(1,1)^T$ and so can this one, but inverted. This square-root property suggests that our Δ of transformations can be turned into a Y of transformations and, indeed this is possible—let all transformations operate from the same vector at the middle $(a\ b\ c)^T$. The three matrices need three rows and three columns since they are distinct in the three directions. These

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

can do this, each operating in turn.

Buffer stages

The logic outlined above does require in the practical implementation the use of buffer stages, but these are used anyway. From the point of view of logic rather than electronics, these are not easily modelled in a system without time. They can be called compellers and merely *insist* that output equals input. Logically (without a sense of evolution), these are simple equalities.

Is there any reason for relational logic to dominate? The systems, being equivalent, can coexist, but note that relational logic is very highly suited to the continuous ordering and relation of isolated facts, whereas a more functional approach cannot do this naturally.

For example, the system associating propositions with numbers can link each proposition with a multiple of three, and add 0 or 1, depending on truth or falseness; adding 2 could be used, when the truth of a statement has not yet been decided, if required.

Adaptation

The strange feature of this logic system is that a 'cubic' network is used. For example: define $[xyz]$ by $((x\ s,t)), ((y\ t,u)), ((z\ s\ u))$, $((t\ s\ u))$ for some s,t,u; then this network: $((a\ c,n)), ((a\ c\ 1)), ((b\ q\ n)), ((n\ c,p)), [qdp]$ performs $d=a$ iff b when a,b are inputs and in this case c is NOT a. Using $c\ d$, as inputs results in b iff c XOR d.

This is a strange feature to adapt to—instead of rank upon rank of ordered functional logic we have what appears as a graph with two sorts of vertice or node and a rank of buffers (one way round or another). This utilization of flexibility of options is difficult for people, but very easy for computers, which can quickly print out all available uses of a net, whethere they can be used immediately or not.

Here, then, is a logic based merely on arithmetic ■