A CURIOUS NEW RESULT IN SWITCHING THEORY

by Lee C.F. Sallows

"*Gödel turned out to be an unadulterated Platonist, and apparently believed that an eternal "not" was laid up in heaven, where virtuous logicians might hope to meet it hereafter*." - Bertrand Russell [1].

Introduction

The following account relates how a puzzle brought to light a remarkably simple, highly intriguing, probably useless, but undeniably fundamental new result in switching theory. Spice is added to the story through the role played by construction of a wildly improbable electronic device in helping to establish the new finding. "Switching theory" has a slightly old-fashioned ring to it, what exactly does it signify? A brief remark on this and a couple of related matters will set our subject in perspective and prepare the way for issues arising later.

Computer science emerges into view as a separate discipline from a cluster of related topics, chief among them symbolic logic, Boolean algebra, switching and automata theory. Logic, originating with Aristotle, concerns the study of deductive inference, of the conditions of truth-preservation in deriving one statement from another. More than two millenia following Aristotle, George Boole was to design his algebra to model logic, a step largely intended to replace reasoning with calculation, with the rule-governed manipulation of symbols. Boolean algebra, we remind ourselves, comprises a so-called *formal system*: a well-defined set of signs and conventions by means of which, starting with certain symbol strings, certain others may be legally substituted, the latter being deemed equivalent to the former. No meaning is attached to these transformations, except in the loose identification of sign with signified usual when *applying* such formalisms to external systems (such as logic). Whether the algebra applied really is an accurate model of the system in question is of course a problem not resolvable within the algebra itself.

A notable success in the practical application of Boolean algebra occurred with the appearance of C.E. Shannon's "Symbolic Analysis of Relay and Switching Circuits" in 1938 [2]. Ever since, the analogy of "0" and "1" with open and closed switch contacts, and of series/parallel switch connections with AND/OR Boolean operators has been a stereotypical textbook example. Then, as today, a *relay* was an electro-magnetically operated switch, a device opening up new realms of complexity in the possibilities it offered of switches controlling still other switches in endlessly convoluted networks. The problems thrown up in this new domain soon became the concern of "switching theory".

Ten years following Shannon, switching theory advanced to a new level of maturity with G.A. Montgomerie's "Sketch for an Algebra of Relay and Contactor Circuits" [3]. By now a vital distinction had been recognized in the division of networks into *combinational* and *sequential* types. Combinational circuits were those in which the open or closed states of every switch depended purely upon current input values (0,1) to the network. A Boolean formula, simple or complicated, would always decribe this relation satisfactorily. Sequential circuits, on the other hand, were those whose response to input patterns also depended in part on their past history: on the foregoing *sequence* of values presented. The behaviour of the circuit might thus change significantly after receipt of some critical input, the latter event thereby being in some sense "remembered". In fact *memory* (introduced via feedback effects) was the key property of such networks. Flow tables and state transition diagrams now displaced static formulas in the need to capture this temporal context-dependent behaviour. Thus was launched the study of what came to be called sequential or *finite state machines*, a field later to be known as automata theory.

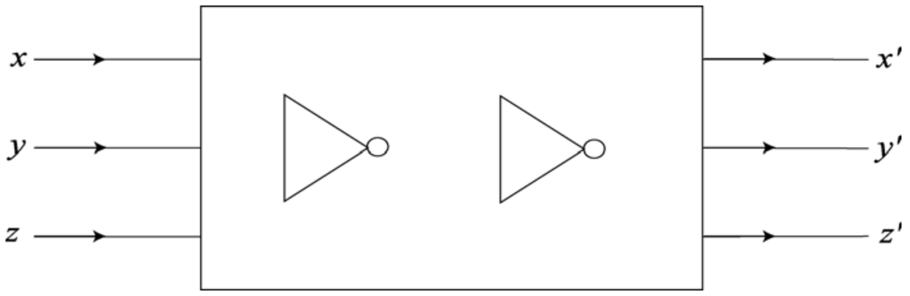
The progress of developments in automata theory is beyond our purpose here: advances were rapid, leading to theoretical results of great moment in connection with Turing machines and mathematical linguistics, the subject soon shading seamlessly into computer science proper. Back in the mainstream of switching theory however, by the 1960's advances in technology had shifted emphasis away from relays and onto "electronic digital logic" realized in micro-packaged integrated circuits or "chips". Mechanically actuated contacts gave way before "AND-gates" and "OR-gates" etc., the binary states (0,1) of whose input and output lines were represented by two discrete voltage levels. Soon Boolean algebra was a standard item on the training syllabus of electronics engineers; formerly recondite chapters of the now slightly outmoded-sounding "switching theory" became the stock-in-trade of every technician.

Hence, overtaken by studies into the more challenging finite state machines, as a subject of research, switching theory dropped into the background, furnishing instead a well-knit body of established results that found daily application in electronic logic design. This is not to say that all the theoretical questions raised had been successfully answered. Many problems, especially in the area of minimization, remained unsolved; later these would provide a point of departure for the currently vigorous *theory of circuit complexity* (see [4]), a field closely related to, yet historically distinct from the old switching theory. In any case, mass-production techniques had extinguished any practical need for such solutions. Gone forever was the pioneering impetus of the early days.

Who then would have expected to stumble across an undiscovered nugget still reposing amid the slag-heaps of this abandoned mine?

A Knotty Problem

Recently browsing through *A Computer Science Reader* (Selections from *Abacus*, Springer-Verlag 1988), my eye was caught by an article on Automated Reasoning by Larry Wos [5]. Wos illustrated the working of his reasoning program by means of a few example problems, one of which immediately captured my attention. It was this:



The black box above receives binary inputs (0, 1) at *x*, *y* and *z*. Each output line yields the complement of the corresponding input; that is, if *x* is 0, *x'* is 1, and so on. Each of the eight possible 3-bit input words thus gives rise to its complementary word at the outputs. Normally speaking such a transfer function would be achieved by using three inverters (NOT's) connected between each input and output.

Problem: Design a network using any number of AND and OR gates, but *not more than two* (2) NOT's to achieve exactly the same input-output function. (The AND's and OR's may have as many inputs as required.)

Now relays, gates, switchery and logic hold powerful fascination for some. The possibility of simulating three inverters by means of two had never so much as crossed my mind before; the bare contingency hinted indefinably at something wonderful. It seemed to call for ingenious circuitry. The puzzle had me hooked in no time.

It turned out to be a far tougher conundrum than first imagined. So much so, in its elusiveness it became hypnotic. In fact, on and off I took almost a fortnight to solve it, succeeding even then only through reasoning aided by trial and error. But the solution was worth waiting for: an intricate network of true Platonic elegance and inevitablity. It is a logical constellation that was always there, sooner or later someone was bound to find it; a sheer poem for the switching theorist. As the sequel shows, the name of the man who *did* find it first turned out to be Edward F. Moore, a distinguished pioneer in the field of automata theory. From now on I shall refer to the basic arrangement as Moore's circuit. Incidentally, Wos’s automated reasoning program was successful in solving the problem; his method being too complex to outline here, a detailed account can be found in [6].

One version of Moore's circuit is shown in Figure 1. The network admits of a number of (essentially minor) variations, some more economical in gates than others; our example is picked for its functional clarity.

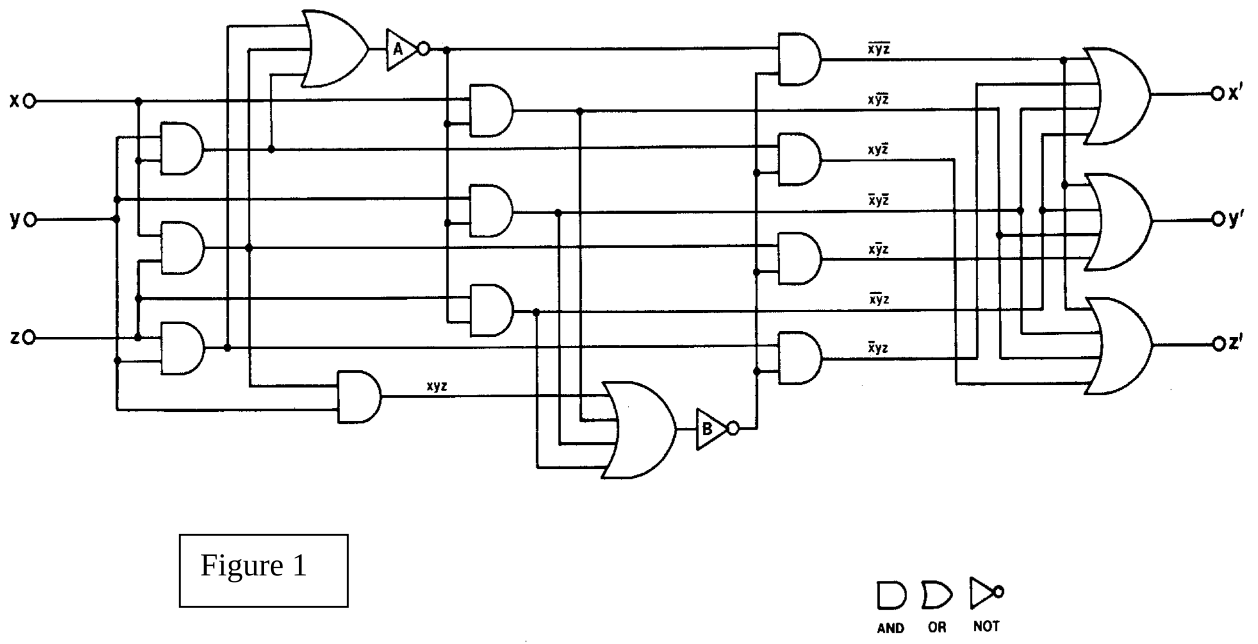


Figure 1

Interested readers might like to seek for a more parsimonious circuit using one gate fewer than Figure 1 (multi-input gates then being counted as if built up from 2-input equivalents; Figure 1 thus containing 11 AND’s and 14 OR’s). In view of its importance to what follows, a few comments on Moore’s circuit will be worthwhile.

Central to every variant of Moore's solution is circuitry leading to a binary representation of the number of zeros present in the input word (*xyz*) by the four possible states of the two inverter outputs: 00 = none, 01 = one, 10 = two, 11 = three zeros. Simple as this may seem, there is but a single way to achieve it. In effect, each inverter's output state (0 or 1) must represent a classification of *xyz* according to whether the number of ones it contains falls in the top or bottom row (first inverter), and in the left or right column (second inverter) of the following table:

	1	0
1	0	1
0	2	3

In this way the intersection of A's row and B's column choice pinpoints the number of ones (and thus, zeros) in the input word.

In our circuit, use of AND gates to combine this information with the specific input pattern enables a complete *decoding* of the input word. See how each of the seven lines feeding the three OR gates at the right is uniquely activated by a different input word (indicated). The circuitry to the left is thus a "3-bit to parallel decoder". Note that although available, the eighth line ($2^3 = 8$) is unused since, when active (i.e. when $x = y = z = 1$), all outputs are to remain 0.

Similarly, the three interconnected output OR's comprise a "parallel to 3-bit *re*-coder", the coding in this case ensuring that *xyz* inputs that are 0 result in corresponding outputs that are 1, and vice versa: an active "$x\bar{y}\bar{z}$" line turns on outputs *y* and *z*, for instance. A point to observe though is that alternative OR combinations could replace (or supplement) this one to produce any desired input-output functions: an output word may have as many bits as we please, and distinct recoders working in parallel could realise unlimited *simultaneous* output words, if required. Already one senses a surprising latent potency here, although, as we shall see, most of the magic in Moore's circuit lies exactly in the mischievous recoding he *did* choose.

Speaking of coding and recoding serves to recall that a circuit diagram is a kind of coded representation and thus itself capable of translation into different symbol systems. A change of medium often brings new aspects into view. An obvious alternative in this connection is Boolean algebra. Re-expressing Moore's circuit in these terms is a mere mechanical exercise.

Designating the output of inverter A as *A*, for instance, we can work backwards through the circuitry towards the inputs, transcribing directly as we go:

$$A = \text{Not}[(x \mathbin{\&} y) \text{ Or } (x \mathbin{\&} z) \text{ Or } (y \mathbin{\&} z)].$$

Comparing formula with circuit we find the inverter is replaced by Not, the 3-termed, square-bracketed Or expression deputizes for the OR-gate wired to its input, and the three parenthesized terms stand in for the AND-gates communicating between the input pairs *xy*, *xz* and *yz* and the OR inputs. Note how the nesting of expressions reproduces the pattern of outputs feeding into inputs in the circuit. We are looking at a fragment of Moore's circuit written in a different language.

Analogously, and taking advantage of the above, a compact expression representing the output *B* of inverter B can also be written:

$$B = \text{Not}[(x \mathbin{\&} A) \text{ Or } (\text{y} \mathbin{\&} A) \text{ Or } (\text{z} \mathbin{\&} A) \text{ Or } (x \mathbin{\&} y \mathbin{\&} z)].$$

Note how the presence of *A* as an argument in the function describing *B* is more than a convenient abbreviation, it reflects *A*'s antecedence in the signal processing path: the value of *A* must already be available in determining that of *B*, but not *vice versa*, a point we shall have cause to recall later. However, the real convenience of these partial descriptions becomes clear in the crisp encapsulation of the complete Moore circuit they now facilitate:

$$x' = [(y \mathbin{\&} A) \text{ Or } (z \mathbin{\&} A) \text{ Or } (B \mathbin{\&} y \mathbin{\&} z) \text{ Or } (A \mathbin{\&} B)]$$

$$y' = [(x \mathbin{\&} A) \text{ Or } (\text{z} \mathbin{\&} A) \text{ Or } (B \mathbin{\&} x \mathbin{\&} z) \text{ Or } (A \mathbin{\&} B)]$$

$$z' = [(x \mathbin{\&} A) \text{ Or } (y \mathbin{\&} A) \text{ Or } (B \mathbin{\&} x \mathbin{\&} y) \text{ Or } (A \mathbin{\&} B)]$$

See how the equations expose a (predictable) 3-fold functional symmetry hinted at, but less successfully conveyed, by their equivalent circuit diagram, an obfuscation resulting from the latter's confinement to two dimensions. (An amusing exercise is to design a 3-D version of the circuit recapturing the trilateral balance.) Still later we shall have occasion to recall these formulas. So much then for a preliminary look at Moore's circuit.

Networks and Notworks

This was all fine as far as it went: an intriguing puzzle with a beautiful solution, if lacking in practical application. During an early stage in reaching that solution however, a rather astounding thought hit me. As an electronics engineer the idea occurred to mind quite easily and, although perhaps ingenious in small degree, is certainly no creative *tour de force*. Nevertheless, the implications struck me as luminous and compelling. The idea was simply this:

If it is possible to simulate three independent NOT-functions using only two primary NOT's (or real inverters) then couldn't we *use* two of those three in order to simulate a *second* set of three NOT-functions? At this stage, having used only two of the first set of three, there would still be one over. That means that a total of FOUR independent NOT-functions would have been simulated while still using only two real inverters. Figure 2 makes the proposal explicit.

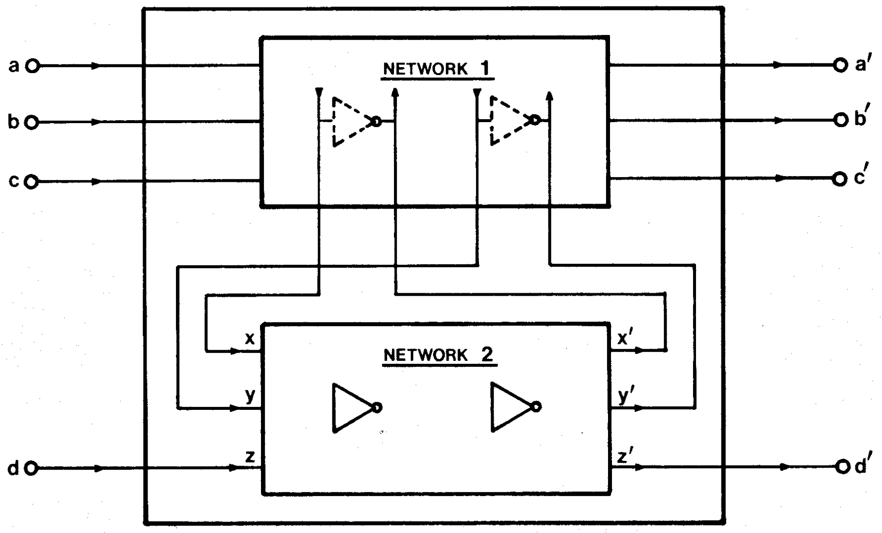
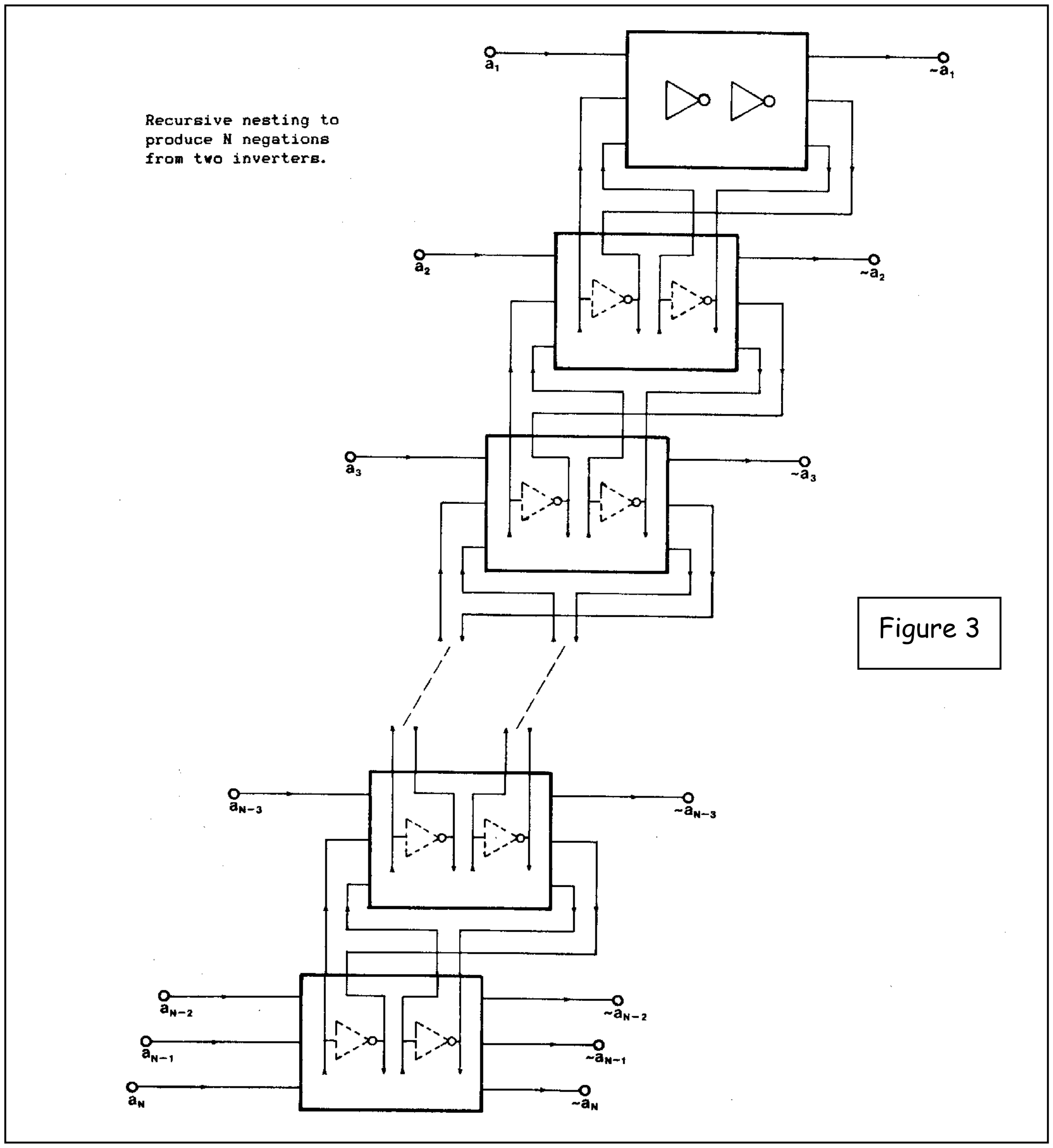


Figure 2

Consider the circuit shown. Network 2 is the straightforward Moore circuit; as such its behaviour is functionally equivalent to an outwardly similar box containing three separate NOT's or inverters connected between each of its three inputs and outputs.

Network 1 is identical to Network 2 except that its two inverters have been removed. The internal input-output connections normally made to the missing inverters have been brought out and connected instead to two channels of Network 2. Network 2 thus furnishes the two NOT functions required for normal working of Network 1 (channels *a, b, c*) while still leaving a fourth independent complement function over (channel *d*).That is all.

The ramifications of this stratagem ripple swiftly outwards. For clearly the four newly created NOT-functions can again be nested in an endlessly expandable recursive hierarchy to produce an unlimited number of independent negation functions; see Figure 3.



In other words (and striving for the infinite in the name of logic):

Theorem I

In any universe, exactly two fundamental negators suffice for concomitant synthesis of all others.

But this soon leads us to a couple of other interesting consequences:

Theorem II

A device whose input-output relations are described by some system of Boolean functions is always constructible using a network comprising some number of AND- and OR-gates but no more that two inverters.

Or, still more ambitiously, (and relying on other well-known results in the field):

Theorem III

Every possible finite state machine (automaton) is realizable using no more than two primary complement functions.

Am I alone in continuing to feel a sense of wonder in this simple discovery?

As I say, the idea for the above configuration occurred to me at the time of reading Wos' article, even before solving his problem. Taking it to be a merely personal rediscovery of a presumably well-established result in logic, thought of any further development never arose. Being satisfied the idea was sound, as an engineer I felt only sheer surprise that, in principle, all the millions of inverters in use throughout the world could be "seeded" from a single pair. Having a romantic turn of mind, it conjured an imaginative vision of a sort of Yin-Yang dyad of inverters occupying a dusty, temperature-controlled glass case at the National Bureau of Standards. Wires leading away from the four old-fashioned knurled brass input and output terminals lead off for distribution to other boxes scattered about the nation. (I should say *five* terminals: a "common" or reference would also be required.)

Well-established result or no, the self-duplicating inverter circuit was a revelation to me and continued to exercise fascination. Having nothing better to do, for fun I typed out a devilish new version of Wos' problem, sending it around to tease friends and colleagues at computer science and mathematics departments at the University of Nijmegen. In the new version, otherwise identical to the old, *four* complement functions are to be realized instead of three. As before, of course, only two inverters are allowed. (As a matter of fact, by Theorem II above, the input-output functions demanded by any severer version of the problem could be made as complicated as one wished. Asking for four NOT-functions is the obvious choice, this representing the least jump in difficulty at the new level of complexity.)

In a few cases the response to this teasing was sharper than anticipated. I suppose the problem is so clear-cut and inescapable it poses a provocative challenge to one's self-estimate as an engineer, mathematician, logician or whatever. Prevarication in the face of this kind of simplicity is difficult; admitting one cannot solve such an apparently elementary problem, even more so. The trouble is, unless you happen to be aware of Moore's circuit (as most people are not), the two separate insights needed for reaching the solution put it well beyond all reasonable ingenuity. It would be a creative act to re-invent Moore's circuit from scratch; penetrating to the fact that such a circuit is necessary as a *componen*t in the 4-complement configuration asks too much of human imagination. In light of this, some of the scepticism poured on my assurances that the solution was complex but straightforward, involving absolutely no hanky-panky, becomes explicable.

It was at this stage that Hans Cornet, a mathematical friend at The Hague, ran across what proved to be the original source of the 3-complement problem. This was in Marvin Minsky's book *Computation: Finite and Infinite Machines* (Prentice-Hall Inc., 1967, p. 65), a confirmation of my assumption that Wos had merely borrowed rather than invented the problem. Looking up Minsky's book in Nijmegen I learned the problem had first been "suggested by E.F. Moore". Admittedly the solution circuit (shown only in skeletal form) is not overtly attributed to Moore but surely no one could pose such a riddle without first having unravelled it?

A comment by Minsky following the problem statement drew from me an appreciative smile: "The solution net ... is quite hard to find, but it is an extremely instructive problem to work on, so keep trying! Do not look at the solution unless desperate." It was a final remark of his however, that brought me up with a jolt. With deepening puzzlement I ran my eye again and again over his two terminal sentences: *To what extent can this result be applied to itself - that is, how many NOTs are needed to obtain K simultaneous complements? This leads to a whole theory in itself; see Gilbert [1954] and Markov [1958].*

Clearly the sufficiency of two NOT's in obtaining *K* (an arbitrary number of) complements was unknown to Minsky. Yet to speak of "applying the result to itself" was a pretty reasonable description of exactly the trick used in my 4-complement circuit. How *could* it be that he had envisioned the self-same possibility without ending up at the same idea? Why on earth should a "whole theory" be required?

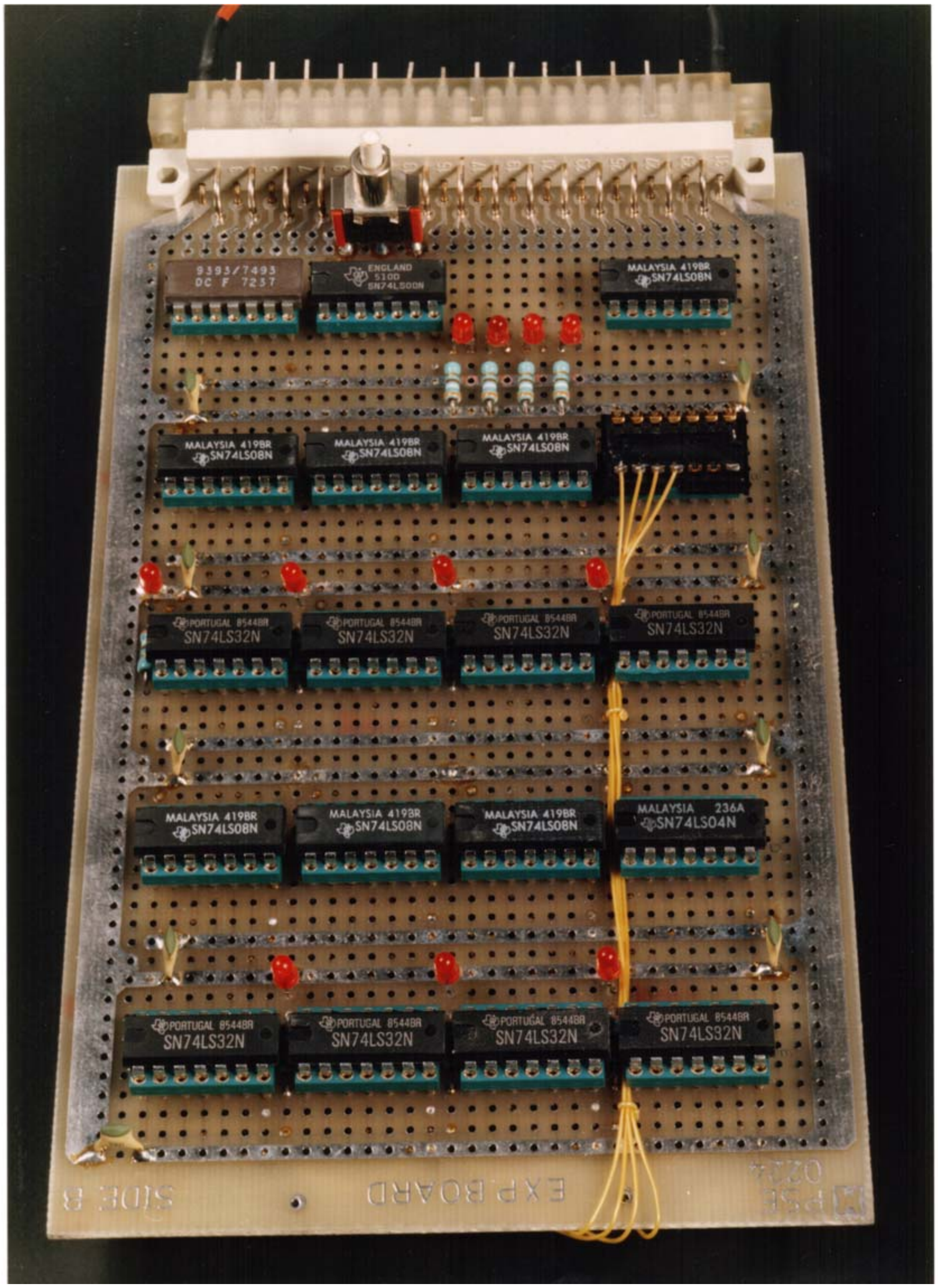
My thoughts sped back to those sceptical friends who could "almost *prove* your 4-complement problem is insoluble". Previously I could afford to be smug, now it was me against Minsky, Gilbert and Markov — the latter a name of intimidating authority in the world of mathematics. Was it likely his theory would turn out to be wrong? Hadn't I after all overlooked some inherent logical flaw that rendered reflexive re-application of the circuit to itself in fact unworkable? I lost no time in hunting up the papers from Gilbert and Markov. Alas, the journals were not available in Nijmegen; there was nothing for it but to order copies. That would take a week or so. In the meantime I returned to the 4-complement circuit, re-examining it from every angle.

Later that evening I banged a defiant fist on the table. It was no good: Markov or no Markov, theory or no theory, there was nothing wrong with that circuit: it *had* to work! —And why not demonstrate my reasoning agreed with reality by *building* it? The very next day saw me launched on construction.

The 4-Complement Simulator

Physical realisation of the circuit followed conventional electronic practice. Taking standard ttl integrated circuits lying to hand (six SN74LS08's and eight SN74LS32's: 14 pin packages containing four 2-input AND's and OR's, respectively) and a prototype-development printed circuit card fitted out with 14-pin chip holders, using a wire-wrap pistol to make interconnections, assembly was completed within a matter of hours.

An obvious approach in implementing the device was dictated by the very principle of operation: first build and test two quite independent Moore circuits, afterwards remove the inverters from one (a single SN74LS04 chip) and replace with connections to two inputs and outputs on the other. This is exactly what I did. In the photograph on page 12 the twin Moore circuits are formed by the two groups of eight chips furthest from the connector. In one circuit, four wires leading from the underside of the card to a small plug that replaces the discarded inverter chip are plain to see.



Finally, to facilitate testing, a push button controlled 4-bit binary counter and a sprinkling of light-emitting diodes (LEDs) were added. Successive presses on the button (seen adjacent to the main connector) run the counter through 0000, 0001, 0010, .. , 1111, the sequence of sixteen possible 4-bit words. Counter outputs are wired to the four NOT-simulator inputs, the presently activated word being indicated by a line of four adjacent LEDs situated close by (on = 0, off = 1). Six remaining LEDs dotted about the board report on the high/low status of the 2 X 3 Moore circuit outputs. For ease of comparability one of these is duplicated so as to form a single line of four evenly spaced LEDs monitoring the four main outputs.

These additions account for two of the three extra chips at one end of the board: an SN74LS93 binary counter and an SN74LS00 4 X two-input NAND used as a so-called set-reset flip-flop to eliminate push-button contact bounce problems. The need for still a further chip made itself felt when, having completed and tested the two separate Moore circuits, the final 4-complement simulator produced by combining them failed to work as anticipated!

At first this was unnerving. Using an oscilloscope, however, the source of the trouble was soon tracked down: under certain input transitions Moore's circuit exhibits *race conditions*. Race conditions arise when delays introduced by hardware inertia result in unintended overlaps between logical state durations, leading to transitory "spikes" or pulses of very short duration (the antecedence of inverter A in the signal processing path now shows its significance; see Figure 4).

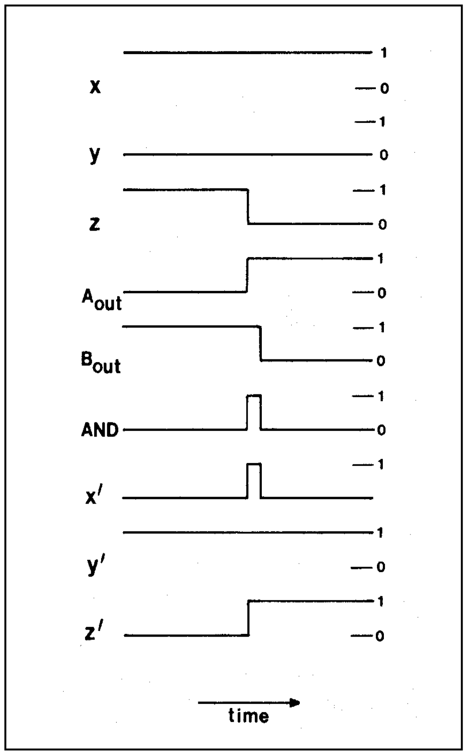


Figure 4

Race condition: Input word *xyz* changes from 101 to 100. Delayed reaction of second inverter (B) to first (A) causes brief pulse at the output of the AND to which they are connected.

Such spikes can be innocuous enough in many applications, but not so in the 4-complement simulator. Here, a spike emerging from output *x'* of the nested circuit becomes gated through the outer circuit to the input of the second inverter (B, see Figures 1 and 2) - itself, however, now simulated by channel *y* of the nested circuit.Our spike, in other words, traverses a sneaky feedback loop and now finds itself re-entering an input of the inner Moore circuit! A vicious circle has been established: regenerative oscillation sets in.

Notice that the culprit here is not the feedback loop — an intrinsic feature of the nested scheme (to which we shall return) — but the pulse generated by the race condition. Happily, a cure is easily effected through interposing a delay in the appropriate line (connecting the output of inverter A to the AND-gate input so as to ensure the latter cannot receive a 1 from the former until after the output of inverter B has changed to 0). This accounts for the last remaining chip in the photo (another SN74LS08; four AND's connected together head to tail, each contributing its own share to the aggregate delay thus created). With this modification completed, turning on the power once again, I finally had the satisfaction of verifying a perfectly functioning 4-complement simulator. It was a happy moment of vindication and triumph.

The 4-complement circuit thus stood acquitted — though in hindsight it is amusing to recall exultation on completion of one of the most futile or, at least, redundant items of electronic apparatus ever constructed! The Great Unanswered Question now remaining, however, was how this success could ever be reconciled with the apparently contradictory theory of Gilbert and Markov? The working device was an unshakeable fact, yet a theorem in logic cannot be validated via any empirical demonstration, however suggestive. Could some sort of a disillusionment still lurk in the publications awaited?

A Gordian Not Unravelled

Following eventual receipt of the anxiously awaited material, a rapid glance at Markov's and Gilbert's conclusions confirmed Minsky's original remark: blatant contradiction of the two-inverters-always-suffice idea. Steeling myself to the mathematics, I settled down to read. Gilbert's is the earlier, exploratory paper, his partial result later subsumed by Markov's more embracing work, "On the Inversion Complexity of a System of Functions" (translated by Morris D. Friedman). We confine ourselves to the latter.

Markov begins his monograph with a series of careful definitions. A small *alphabet* of the signs familiar from Boolean algebra is introduced, *constants* and *variables* included: {0, 1, x_1, ... , x_n , &, Or, Not, (,)}. Certain *words* or strings of these are specified as *formulas* and *sub-formulas, negative sub-formulas* being characterized as those prefixed by "Not". The so-called *inversion complexity of a system of sub-formulas* is now identified with the number of distinct negative sub-formulas occurring in it.

My prècis lacks his precision but the outline of what is going on here is already clear: substituting concatenations of discrete symbols for the tangled Celtic knotwork language of the switching engineer, Boolean formulas replace circuit diagrams: "Not"s are to be counted instead of inverters.

So far so good. Moore's problem itself might well have been so reformulated as to ask for a system of Boolean functions equivalent to x' = Not(x), y' = Not(y), z' = Not(z), but in which "Not" (preceeding a distinct sub-formula) would occur no more than twice. Our previously derived set of three formulas describing Moore's circuit is just such a solution. As before, we are merely talking about the same thing in a different language.

Markov's list of definitions ends abruptly with a bold statement of his result, followed by a one page lemma running into sub-subscripted, sub-superscripted variables that "plays an essential role in the proof". The full proof is spared us — the author doubtless feeling that a recapitulation of the obvious would be too tedious — and so ends his paper. It is just as well: that lemma might have been written in Celtic for all I could make of it (printing errors abound too). Not that I questioned his result for a moment. This was a good instance of what Richard Guy calls *proof by intimidation*.

But what was that result? Following Markov we must be quite precise here.

Consider a system of *m* Boolean functions of *n* arguments. It can be defined by different systems of *m* formulas in *n* variables. Take now a worst case instance of such a system of functions in which the number of distinct negative sub-formulas necessary to their definition is at its greatest. Then, says Markov, the least number of negative sub-formulas that will have to appear in the formulas defining the functions will be $|\log_2 n|+1$ = the number of digits in the binary representation of *n*. (The vertical strokes indicate the truncated value of $\log_2 n$) $|\log_2 n|+1$, in other words, otherwise known as *I* or the inversion complexity of the system of functions, is indeed the Markovian equivalent to the minimum number of separate inverters that would be required in any network implementation of its formulas. Note that *m*, the number of functions (or formulas) does not actually enter into it. (Think of all the recoders that can be connected to Moore's 3-bit to parallel decoder, each yielding a new output function, none demanding extra negations).

We can examine this further by taking Moore's problem as a example. Translating into Boolean terms, our question concerns a system of three functions (x' = Not(x), y' = Not(y), z' = Not(z), of three arguments x, y, z). Applying Markov's result we find $n = 3$, $\log_2 3 = 1.5849...$, hence $I = 1 + 1 = 2$. That agrees with our conclusion: two inverters sufficient.

But what about the 4-complement problem? Now $n = 4$, $\log_2 4 = 2$, $I = 2 + 1 = 3$. *Three* inverters are required. That disagrees with our conclusion. In effect, Theorem I above would assert that $I = 2$, irrespective of *m* and *n*. Here is the contradiction.

The collision here is so acute that something will have to give way. And so it proves. Forcing the issue to a head, an obvious step now is to produce a counter-example to Markov's result by writing out the 4-complement circuit as a system of Boolean formulas, thus demonstrating that only two distinct negative sub-formulas need appear.

And indeed, with this comes a breakthrough and the resolution to this whole curious dilemma. The scales, so to speak, are about to fall from our *I*'s. For with an attempt to write out a set of formulas depicting the 4-complement circuit comes the discovery that *no Boolean representation of it exists*.

The barrier to deriving a Boolean representation is revealing. Looking back at the 4-complement block diagram (Figure 2), recall that Network 2, the nested box, is a pure Moore circuit for which we already have a system of formulas. To represent the complete 4-complement box, however, we first need to respecify *x*, *y* and *z* — the inputs to the nested box — in terms of the new set of arguments: *a*, *b*, *c* and *d*, the main inputs. The obstacle to achieving this appears in finding that no expression for *y* can be derived without *y* occurring as one of its own arguments!

How does this come about? It is our old friend the sneaky feedback loop, reminding us that this is no longer a simple *combinational* circuit like Moore's. Through the nesting of one box in another a primitive form of memory has been introduced whereby it has become a *sequential* switching circuit whose subsequent internal state depends both upon present inputs and current state: the present value of *y* plays a part in determining *y*'s new value. In short, the 4-complement circuit is really a *finite state* *machine*, a device whose context-sensitive action lies beyond the descriptive scope of Boolean formulas. The facile notion that everything can always be "talked about in a different language" is thus not without its pitfalls.

Still sneakier, (and this really is rather subtle) the 4-complement circuit is a finite state machine *mimicking* the behaviour of a non-sequential machine, the latter comprising a humble combinational circuit of just four inverters: a' = Not(a), b' = Not(b), c' = Not(c), d' = Not(d). Here we have the peculiar case of a higher or meta-Boolean form of life disguised as a lower or Boolean form. The camouflage is truly effective too, since no experiment conducted on the terminals of the 4-complement (black) box could determine whether it contained Boolean or non-Boolean-representable entrails. (Although curiously — and here is another tricky twist — the non-Boolean circuit is actually composed entirely of *Boolean components*: AND's, OR's and NOT's, an indication both of the import of their *interconnection pattern* and of the source of weakness in the algebra that cannot describe it.)

A fine distinction is involved in all this that it is worth being clear about. A Boolean *function* describes a relation or mapping between one two-valued variable (the value of the function) and others (its arguments). As such it may be expressed or specified in different ways; in a tabulation of corresponding values, for instance. Often we represent it as a Boolean *formula*, that is to say, as a legal expression in the formalism called Boolean algebra. In that case, the dependence of the formula's value on that of its variables will strictly mirror that of the function on its arguments. Moreover, any Boolean function *can* always be described by a Boolean formula.

But that is not to say that it *has* to be so represented or that a specification or implementation of the function must depend on some analogous structure or mechanism. The 4-complement finite state machine is an example of an alternative implementation, its *effect* representable by a' = Not(a), etc., but its internal operation (as embodied in its circuit diagram) having no counterpart in Boolean algebra. The importance of this is that generalizations about Boolean functions are not to be reliably based soley on inferences about Boolean representations of those functions.

So it is that the supposed discrepancy between Markov's conclusion and Theorem I turns out to be illusory. The meticulous definitions at the beginning of his paper are not for nothing. As a careful re-examination of the account above will show, the result he proves is explicitly restricted to Boolean functions *realized in Boolean formulas*. Our concern, on the other hand, (if only lately appreciated) has been with Boolean functions realized otherwise. Minsky's implication notwithstanding, Markov's work is simply *inapplicable* to the case in hand. Like the 4-complement box, the *K*-complement box need employ no more than two inverters. But at least $|\log_2 K|+1$ distinct negative sub-formulas will be required in any Boolean formulas describing the input-output functions of the latter. No contradiction is implied. E.N. Gilbert's paper, incidentally, which also addresses the minimum inverter requirement question is equivalently restricted, his analysis being confined to loop-free networks.

Even so, doesn't a suspicion linger that the *K*-complement simulator is in some way yielding something for nothing? After all, inverting binary signals is a concrete if trivial operation, analogous to flipping over coins so as to make heads from tails or tails from heads. In the end, just how *is* it that *K* such reversals can be effected given only two reversing machines?

The answer is simple. It is done by using those machines more than once. Through reiterated application we can achieve serially the same result as *K* single-action machines working in parallel. But at a price, to be sure. Here is how John E. Savage puts it in *The Complexity of Computing* [4]: "Sequential machines compute logic functions, just as do logic circuits. However, since sequential machines use their memories to reuse their logic circuitry, they can realize functions with less circuitry than a no-memory machine but *at the expense of time*" [my italics]. As we saw earlier, hardware-implemented logic introduces lag. As *K* increases, so will the number of passes through feedback paths in the nested circuitry, and the longer final outputs will take in responding to changing input patterns. In practice this would be a serious factor to consider.

Lastly, note how Savage casts incidental light on the reason why a single inverter – however combined with AND's and OR's – is inadequate for simulating further negators. Negation of externally presented bits on one channel will always require one inverter. But at least a second will be demanded in creating the *memory* needed in re-utilizing that first. In fact, as we have seen, two inverters are both necessary and sufficient.

Simple but hard-won insights are compressed into the foregoing paragraphs. Having gained clearer understanding, a letter to the author whose casual remarks unwittingly triggered this improbable detective story seemed not inapposite. I was gratified thus when, in a subsequent communication, Marvin Minsky warmly concurred in the above analysis, graciously conceding a too hasty perusal of Gilbert and Markov's articles. Likewise, his "To what extent can this result be applied to itself?" turned out to be a mere chance form of words, no reference to recursion intended, but resonant to me under the circumstances.

Thus were *K*-nots disentangled from a Markov chain of deduction, and the sufficiency of two negators in producing moore inverters *ad libitum* confirmed.

Conclusion

The inception of this narrative was a puzzle appearing in *Abacus*. As a matter of fact, the question there posed came in two, supposedly equivalent, versions: Moore's original problem in circuit design and an analagous problem in computer programming. In the latter form we are asked to "write an [assembly language] program that will store in locations U, V, W the 1's complement of locations *x*, *y* and *z*. You can use as many COPY, OR and AND instructions as you like, but you cannot use more than two COMP (1's complement) instructions." This second version is absent from Wos et al’s *Automated Reasoning* [6], appearing only subsequently in his synoptic *Abacus* article. The trouble is, although aimed at preserving the essence of the former, the conditions imposed are actually more restrictive than Moore's: a whole class of solutions becoming inadvertently excluded.

What is it that makes the program version different? In effect, it is a silent prohibition against certain kinds of perfectly valid circuit configurations: a ruling out of the use of *feedback loops* implicit in the *preclusion of a JUMP instruction*. Self-modifying functions would be excluded from representation in software. That is, for every program solution there would be an equivalent circuit, but not *vice versa*. Just as sequential networks defy description in the notation of Boolean algebra, so loops in any circuit solution will defeat implementation in such a program.

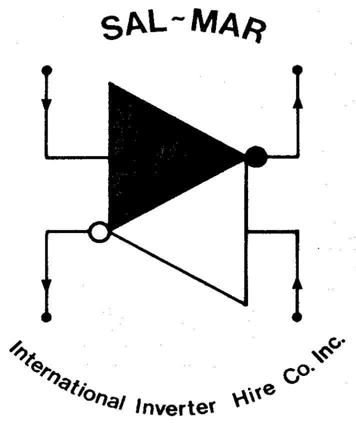
The slip is an easy one to make, and especially so when Moore's own circuit uses no feedback. Perhaps it was familiarity with this that unconciously acted to restrict Wos’s contemplation to combinational type solutions only. Let us make no mistake however: discarding one channel from the 4-complement simulator would leave a three channel device answering all the demands of Moore's problem. Here we have a finite state machine solution (one of an infinity) that cannot be represented in the reduced instruction code. I suspect that in the urge to translate Moore's problem into terms suited to his automatic reasoning program, Larry Wos temporarily underestimates and thus misrepresents the complexity and potential of networks using AND's, OR's and NOT's. [In passing - and without any reference to the aforementioned author - the tendency to see circuit diagrams as engineer's easy-to-read-picture-book-explications of "real mathematics" envisioned in putative formulas, is not uncommon among mathematicians. Engineers, I may add, humble as their mental endowment may be, will be more impressed when condescension can be matched with insight into the advantages of a two-dimensional language.] Whether or not the automated reasoning technique could be successfully applied to the 4-complement problem is a further interesting question.

Following the lead suggested here, the 4-complement circuit is elegantly modelled in a simple computer program using iteration to imitate the feedback loop; see page 22. A series of assignment statements based on the earlier derived formulas describing Moore's circuit make up the body of the program. Figure 5 shows a version written in Turbo Pascal. Read in conjunction with Moore's circuit and Figure 2, the program is self-explanatory: more eloquent in fact than any verbal commentary on circuit operation. Interested readers may like to try the effect of including a write statement in the Repeat loop so as to expose the behaviour of *y* under different input sequences.

A final observation on Markov's result must bring this account to a close. Figure 3 depicted the endlessly expandable system of recursively nested Moore circuits for producing an arbitrary number of NOT's. Winning three NOT's from two, every level of nesting yields a spare inverting channel. In practice, however, the mass-production of NOT-functions can be enormously accelerated. How? Notice that Markov's *I* is still only 3 for *n* as high as 7. But this is another way of saying that a simple combinational circuit exists that can simulate seven inverters directly from three. Similarly, from these seven a further 127 can be produced at only the third level of nesting (2 -> 3 -> 7 -> 127 -> ...). Readers may like to test their grasp of the foregoing by writing a program that implements seven inversions while using only two Not operators.

In conclusion, and before any false hopes are raised though, I ought to say that the above suggestion is intended merely as an exercise. Patents, it must be explained, have already been granted and the Sal-Mar International Inverter Hire Company Inc. is due for launching at an early date. Prompt negations of the highest quality will be available to customers via standard phone lines. Charges are expected to be modest.

In the meantime, call me an *adulterated* Platonist if you will, up in heaven *two* eternal NOT's await the arrival of virtuous logicians (and the occasional virtuous engineer). I look forward to rubbing that in with Godel and Russell hereafter.



[Grateful thanks are due to Jim Propp, formerly of the Department of Mathematics, University of Maryland, whose searching criticisms brought to light various errors and made for substatial improvements to an earlier draft of this paper.]

The above article was first published in *The Mathematical Intelligencer*, Vol. 12, No. 1, 1990, pp 21-32.

References

1 B. Russell, *The Autobiography of Bertrand Russell*. Unwin (1978), p. 466.
2 C. E. Shannon, A symbolic analysis of relay and switching circuits. *Trans. AIEE* 57 (1938), 713-723.
3 G. A. Montgomerie, Sketch for an algebra of relay and contactor circuits, *Jour. IEE* 95, Part III (1948), 303-312.
4 J.E. Savage, *The Complexity of Computing*, Wiley (1976).
5 L. Wos, *A Computer Science Reader*, Ed. E. A. Weiss, Springer-Verlag, (1988), pp 110-137. Orig. pub. *Abacus*, vol. 2, no. 3 (Spring 1985), pp. 6-21.
6 L. Wos, R. Overbeek, E. Lusk & J. Boyle, *Automated Reasoning, Introduction and Applications*, Prentice-Hall (1984).
7 M. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall (1967), p. 65.
8 E.N. Gilbert, Lattice theoretic properties of frontal switching functions, *Jour. Math. & Physics* 33 (April 1954), 57-67.
9 A.A. Markov, “On the inversion complexity of a system of functions” (translated by M. D. Friedman), JACM 5 (1958) 331-334. Orig, pub. *Doklady Akad Nauk SSSR* 116 (1957), 917-919.

```
Program Four_Complement_Simulator; {Turbo Pascal Version 3}

{Compare Fig.2 and Moore circuit diagram for all that follows}

Var
   a,b,c,d,x,y,z, {main and nested inputs}
   A1,B1,A2,B2,   {inverter outputs Networks 1/2}
   aa,bb,cc,dd,   {main outputs a',b',c',d' in Fig.2}
   initial_y      {previous y state} : Boolean;

Procedure Specify_inputs; Var ai,bi,ci,di : char;
  Begin
     Writeln('Input 4 truth-values for a,b,c,d: T(rue)/F(alse)');
     Read(Kbd,ai,bi,ci,di);
     If ai='T' Then a:=True Else a:=False;
     If bi='T' Then b:=True Else b:=False;
     If ci='T' Then c:=True Else c:=False;
     If di='T' Then d:=True Else d:=False;
     Writeln('Inputs:  ',a:8,b:8,c:8,d:8);
  End;

Begin {Main}
   Specify_inputs;
       {Nested box inputs x and z first respecified in terms of a,b,c,d:}
   x:= ((a And b) Or (a And c) Or (b And c));
       {Expression for 1st inverter input in Moore circuit}
   z:= d;
       {z is connected to input d}
       {Input y feedback involvement calls for iteration:}
     Repeat
       initial_y:=y;
       A2:= Not((x And z) Or (x And y) Or (y And z));
            {Nested box first inverter output defined}
       B2:= Not((x And A2) Or (y And A2) Or (z And A2) Or (x And y And z));
            {Nested box second inverter output defined}
       A1:= ((y And A2) Or (z And A2) Or (y And z And B2) Or (A2 And B2));
            {A1 = x' output of nested box, see Fig.2}
       y:=  ((a And A1) Or (b And A1) Or (c And A1) Or (a And b And c));
            {Expression for 2nd inverter input in Moore circuit. y may have
             changed value, or not, depending on previous input pattern}
     Until  y = initial_y;
            {Remain in loop until y stabilizes; two loop passes always
             suffice: y's value self-confirming after one change. A
             simple 2-cycle Do-loop would serve equally well here}
  B1:= (x And A2) Or (z And A2) Or (x And z And B2) Or (A2 And B2);
            {B1 = y' output of nested box, see Fig. 2}

            {Standard Moore circuit formulas follow}

  aa:= ((b And A1) Or (c And A1) Or (b And c And B1) Or (A1 And B1));
  bb:= ((a And A1) Or (c And A1) Or (a And c And B1) Or (A1 And B1));
  cc:= ((a And A1) Or (b And A1) Or (a And b And B1) Or (A1 And B1));
  dd:= ((x And A2) Or (y And A2) Or (x And y And B2) Or (A2 And B2));

  Writeln('Outputs: ',aa:8,bb:8,cc:8,dd:8);
End.
```