UNICIENCIA 7 (1-2)
pp. 43-57
1990

PROPOSITIONAL CALCULUS AND BINARY CALCULUS

Osvaldo Skliar and Víctor Medina
Universidad Nacional, Heredia, Costa Rica.

Abstract

We present an efficient method of propositional calculus which allows the manipulation of logical functions with an arbitrary number of propositional variables. This method is based on the use of binary sequences (in other words, sequences of digits which can only be either 0 or 1) and certain operations between them. This calculus is then implemented by using neural network type devices.

1 Introduction

The objective of this paper is to present a new operative approach for propositional calculus. This is based on the establishment of certain correspondences between:

1. Logical functions and natural numbers expressed in base 2, and
2. Operations carried out on logical functions and those numbers considered as "binary chains" (i.e. sequences of digits each of which can only be either 0 or 1).

Let $f(x_1, x_2, \ldots, x_n)$ [1]be a logical propositional function of n logical propositional variables $x_1, x_2, \ldots, x_n$. Each of this n propositional varibles can be replaced by a proposition. We accept that one of two possible truth values can correspond to each proposition, so each proposition is considered either true or false, and if a proposition is replaced by a false proposition, it will be indicated by substituting it with the number 0. Conversely, if it is replaced by a true proposition, it will be identified by the number 1. By specifying any given one of the 2^{2^n} existing logical functions of n propositional varibles we mean to establish the truth value for the logical function in question for each possible choice of truth values for the propositions which replace the propositional variables (which is usually done by using a truth table). This can also be accomplished by using an algorithm which allows the logical function's truth value to be established for each choice to which reference was made. To ilustrate this, a truth table for one of the 16 existing logical functions of two propositional variables is shown below. Note that the various choices of truth values for the propositional varibles are indicated by using the conventional 0 for "false", and 1 for "true".

[1]It is common to designate as "proposition" the logical function $f(x_1, x_2, \ldots, x_n)$. This nomenclature, while admissible for reasons of brevity for those who already have a clear undertanding of the basic concepts involved, is not correct in a strict sense. Actually, a proposition can only be obtained from a propositional function by substituting each logical propositional variable with a proposition or by "linking" it using a quantifier.

Notice in **Figure 1** that in two cases – one in which both propositional variables (x_1, x_2) have been replaced by false propositions, and the other in which both have been replaced by true propositions – the "true" value was assigned to the logical function. This example is referred to as the "equivalence" logical function of x_1 and x_2 ($f(x_1, x_2) = x_1 \leftrightarrow x_2$).

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	1

Figure 1

A system of orthogonal axes of coordinates can be introduced where each axis corresponds to one of the propositional variables. For each corresponding propositional variable, two values (0 or 1) are possible on each axis.

How can the logical function specified in the truth table in **Figure 1** be shown geometrically? One possibility is to draw a small black circle on each point of the cartesian plane shown in **Figure 2**, that correspond to the truth values of the propositions by which the propositional variables are replaced so that the logical function under consideration assumes the truth value "true" (**Figure 3**).

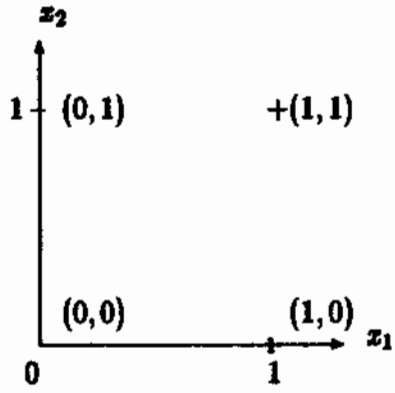


Figure 2

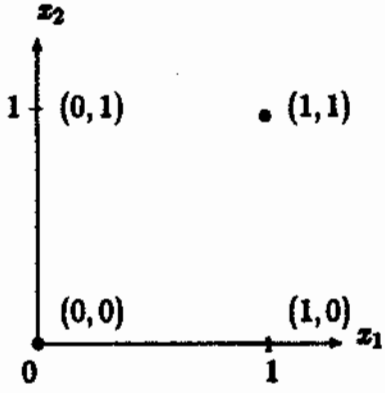
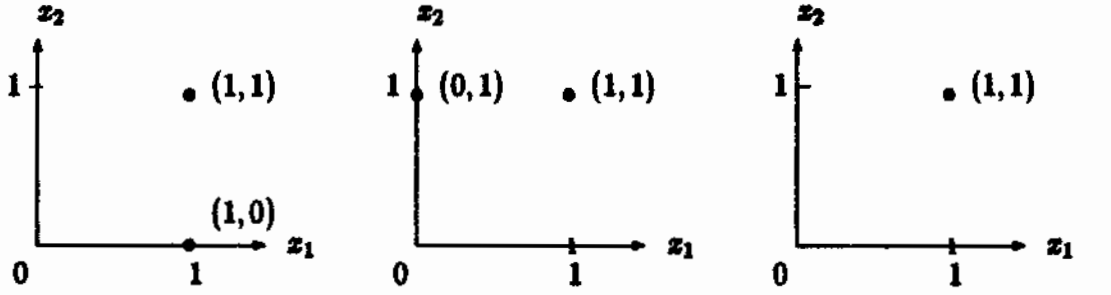


Figure 3

It is obvious from a conceptual point of view that we can generalize this approach to be used in the case of n propositional variables. In order to geometrically define one of the existing 2^{2^n} logical functions, a given subset of the existing 2^{2^n} must be chosen. This subset is comprised of the n-dimensional space's 2^n points that correspond to the 2^n different possible choices of truth values for those propositions that replace each of the n propositional variables (each of these choices is defined in the corresponding rows of the proper truth table). Let us denote by R_f the subset of points corresponding to the logical function f and let us call it the *region of validity* of f. The conjunction of two logical functions is a logical function whose validity region is the intersection of the

validity regions of the two operands. The validity region for the disjunction of two logical functions, on the other hand, is the union of the validity regions of its operands. So, for instance, if $f(x_1, x_2) = x_1 \wedge x_2$,[2] we obtain $\mathcal{R}_f$ as shown in **Figure 4**.



$\mathcal{R}_{x_1} = \{(1,0),(1,1)\}$ $\mathcal{R}_{x_2} = \{(0,1),(1,1)\}$ $\mathcal{R}_f = \mathcal{R}_{x_1} \cap \mathcal{R}_{x_2} = \{((1,1)\}$

Figure 4

Moreover, observe that $\mathcal{R}_f$ determines the truth table for f. So for the previous example we get:

x_1	x_2	$f(x_1, x_2) = x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Figure 5

Conversely, given the truth table of any logical function f, we can determine $\mathcal{R}_f$. To ilustrate this, let us look at the truth table for the function $f(x_1, x_2) = x_1 \leftrightarrow x_2$, shown in **Figure 1**. The validity region in **Figure 3** is obtained from this table (**Figure 1**); that is $\mathcal{R}_{x_1 \leftrightarrow x_2} = \{(0,0)(1,1)\}$. Also note that a geometrical analysis of the region $\mathcal{R}_f$ produces other equivalent ways to write the function f. So it clearly follows that from **Figure 3**:

$$f(x_1, x_2) = \begin{cases} x_1 \leftrightarrow x_2 \\ (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2) \\ \neg\{(\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)\} \end{cases}$$

Notice that two logical functions f and g are equivalent if, and only if, $\mathcal{R}_f = \mathcal{R}_g$. Likewise, $f \rightarrow g$ if, and only if $\mathcal{R}_f \subseteq \mathcal{R}_g$.

[2] We should point out here that x_1 can be considered either a logical variable or a logical function whose truth table, in the case of two propositional variables' logical functions, is the following:

x_1	x_2	$f(x_1, x_2) = x_1$
0	0	0
0	1	0
1	0	1
1	1	1

The same can be done for x_2.

It is clear that in this traditional geometric approach 0 and 1 are used only for the purpose of differentiating two points on each axis of coordinates. Therefore 0 and 1 can be changed to any pair of numbers a and b as long as $a \neq b$.

At this point we will develop a closer tie between propositional calculus and numeric binary calculus. Those interested in an alternative geometric approach to propositional calculus should look, for instance, at [1]. It is oriented towards simplifying the computation of the truth value for logical functions of a small number of variables.

2 Binary Approach to Propositional Calculus

We define the operation of disjunction on binary digits (each of which can be either 0 or 1) as:

$$i \vee j = \max \{i, j\},$$

or if preferred, by the following matrix:

$\vee$	0	1
0	0	1
1	1	1

The operation of conjunction, applied to the same operands is defined by:

$$i \wedge j = \min \{i, j\},$$

or if preferred, by the matrix:

$\wedge$	0	1
0	0	0
1	0	1

The operation of negation or "inversion" is defined by:

$$\neg i = 1 - i,$$

or by the following one–column matrix:

$\neg$	
0	1
1	0

Let us extend these operations to binary chains with the same number of digits by working on them digit by digit. So, let a and b be two binary chains comprised of n digits, each one being:

$$a = (j_1 j_2 \dots j_n) \text{ where } j_i \in \{0, 1\}, \text{ for } i = 1, 2, \dots, n, \text{ and}$$
$$b = (k_1 k_2 \dots k_n) \text{ where } k_i \in \{0, 1\}, \text{ for } i = 1, 2, \dots, n.$$

We have:

$$\begin{aligned} a \vee b &= (j_1 j_2 \dots j_n) \vee (k_1 k_2 \dots k_n) \\ &= (j_1 \vee k_1 \;\; j_2 \vee k_2 \dots j_n \vee k_n) \\ a \wedge b &= (j_1 j_2 \dots j_n) \wedge (k_1 k_2 \dots k_n) \\ &= (j_1 \wedge k_1 \;\; j_2 \wedge k_2 \dots j_n \wedge k_n) \end{aligned}$$

If $f(x_1, \ldots, x_n)$ is a logical function of n logical variables, there are 2^n different ways of assigning truth values (0 or 1) to the variables $x_1, \ldots, x_n$. We order these assignments according to the usual ordering of numbers with n binary digits. Thus for each assignment of the values of the variables $x_1, \ldots, x_n$, f will have a value of either 0 or 1, thus generating an ordered sequence of 2^n binary digits which completely define the function f. So we can identify $f(x_1, x_2) = x_1 \rightarrow x_2$ with the binary sequence (1101) because its truth table is:

x_1	x_2	$x_1 \rightarrow x_2$
0	0	1
0	1	1
1	0	0
1	1	1

Figure 6

Moreover, it follows that for each sequence $(i_1 \ldots i_{2^n})$ of 2^n binary digits, there is a corresponding logical function f, which is determined by the truth table:

x_1	x_2	$\ldots$	x_n	f
0	0	$\ldots$	0	i_1
0	0	$\ldots$	1	i_2
$\vdots$	$\vdots$		$\vdots$	$\vdots$
1	1	$\ldots$	1	i_{2^n}

We denote $\mathcal{L}^{(n)}$ as the set formed by the 2^{2^n} logical functions of n variables. These logical functions are identified by the 2^{2^n} binary sequences of 2^n digits. Since each sequence determines a binary number, we order $\mathcal{L}^{(n)}$ according to common oredering for such binary numbers. So we can write:

$$\mathcal{L}^{(n)} = \{ f_0, f_1, \ldots, f_{2^{2^n}-1} \}$$

where $f_0 = (0 \ldots 00)$, $f_1 = (0 \ldots 01)$, $\ldots$ $f_{2^{2^n}-1} = (1 \ldots 11)$.

In addition, we consider the logical connectives $(\vee, \wedge, \neg, \rightarrow, \leftrightarrow)$ as operators defined on $\mathcal{L}^{(n)}$, and consequently, we write $\leftrightarrow \{ f_i, f_j \}$ instead of $f_i \leftrightarrow f_j$, $\wedge \{ f_i, f_j \}$ instead of $f_i \wedge f_j$, etc. The operator $\neg$ acts on only one oprerand that can be any of the logical functions. Either binary or unitary operations, can be performed using those operators defined and closed in $\mathcal{L}^{(n)}$.

The disjunction operation of any m logical functions $f_{i_1}, \ldots, f_{i_m}$ of $\mathcal{L}^{(n)}$, which is usually written as $f_{i_1} \vee \ldots \vee f_{i_m}$ will be expressed as $\vee \{ f_{i_1}, \ldots, f_{i_m} \}$. Similarly, their conjunction will be indicated by: $\wedge \{ f_{i_1}, \ldots, f_{i_m} \}$. We will also use the following convention: an asterisk over an operator will indicate that the result of the operation on the logical functions involved (which is also a logical function) depends on the order in which operands are considered. For the operators considered in this paper, the asterisk will only be needed with the operator of implication. Thus " f_i implies f_j " is written as $\overset{*}{\rightarrow} \{ f_i, f_j \}$.

To illustrate, let us look at the operators $\wedge$, $\vee$, and $\neg$ on the elements of the set $\mathcal{L}^{(2)}$ formed by the 16 two variables' logical functions:

$$\begin{array}{llll} f_0 = (0000) & f_4 = (0100) & f_8 = (1000) & f_{12} = (1100) \\ f_1 = (0001) & f_5 = (0101) & f_9 = (1001) & f_{13} = (1101) \\ f_2 = (0010) & f_6 = (0110) & f_{10} = (1010) & f_{14} = (1110) \\ f_3 = (0011) & f_7 = (0111) & f_{11} = (1011) & f_{15} = (1111) \end{array}$$

Notice that as a consequence of the ordering of the set $\mathcal{L}^{(n)}$, $f_j = j_{(2)}^{(n)}$, where $j_{(2)}^{(n)}$ is the binary sequence of 2^n digits corresponding to the decimal number j. On the other hand, the disjunction, conjunction, and negation operations on binary sequences (as previously defined) correspond with the disjunction, conjunction, and negatiion operations on $\mathcal{L}^{(n)}$. For example, in $\mathcal{L}^{(2)}$:

$$\begin{aligned} \vee\{f_5, f_{14}\} &= \vee\{5_{(2)}^{(2)}, 14_{(2)}^{(2)}\} = \vee\{(0101), (1110)\} = (1111) \\ &= 15_{(2)}^{(2)} = f_{15} \\ \wedge\{f_5, f_{14}\} &= \wedge\{5_{(2)}^{(2)}, 14_{(2)}^{(2)}\} = \wedge\{(0101), (1110)\} = (0100) \\ &= 4_{(2)}^{(2)} = f_4 \\ \neg\{f_5\} &= \neg\{5_{(2)}^{(2)}\} = \neg\{(0101)\} = (1010) \\ &= 10_{(2)}^{(2)} = f_{10} \end{aligned}$$

Generally, if f_i and f_j, are elements of $\mathcal{L}^{(n)}$, then:

$$\begin{aligned} \vee\{f_i, f_j\} &= \vee\{i_{(2)}^{(n)}, j_{(2)}^{(n)}\} = f_k \\ \wedge\{f_i, f_j\} &= \wedge\{i_{(2)}^{(n)}, j_{(2)}^{(n)}\} = f_l \\ \neg\{f_i\} &= \neg\{i_{(2)}^{(n)}\} = f_r \end{aligned}$$

where k, l, and r are those decimal numbers that correspond to $\vee\{i_{(2)}^{(n)}, j_{(2)}^{(n)}\}$, $\wedge\{i_{(2)}^{(n)}, j_{(2)}^{(n)}\}$, and $\neg\{i_{(2)}^{(n)}\}$, respectively. In this way logical function calculus is reduced to simple binary calculus. If, as is usually the case, the subindices that specify various logical functions are expressed in base 10, then there must be changes from base 10 to base 2 and vice versa, in order to use this binary calculus.

In addition, observe that in $\mathcal{L}^{(n)}$ each logical variable corresponds to a logical function. So in $\mathcal{L}^{(2)}$, for instance, $x_1 = (0011) = f_3$ and $x_1 = (0101) = f_5$, and in general x_i, ($i = 1, \ldots, n$), considered as an element of $\mathcal{L}^{(n)}$, is f_j, where j is the number expressed in base 10 with the 2^n –digit binary expansion:

$$\underbrace{(\underbrace{0\ldots0}_{2^{n-j}}\underbrace{1\ldots1}_{2^{n-j}}\cdots\underbrace{0\ldots0}_{2^{n-j}}\underbrace{1\ldots1}_{2^{n-j}})}_{2^j}$$

This is the sequence formed by 2^{n-j} "zeros" followed by 2^{n-j} "ones" and so on (2^j times).

We wil now describe how neural networks can be implemented to perform these operations.

3 Implementation of the Binary Approach Using Neural Networks

The logical operations described above can be performed using neural net type devices which are easily simulated on digital computers. When working with logical functions (especially those with many variables), it is quite convenient to automate the involved calculations by using neural nets (see [2], and [3]).

Which are the general –structural and functional– characteristics that these neural nets have? In other words: what is the neural net pattern from which all the neural nets we will be dealing with can be considered as particular cases? To answer this question we will first describe the model.

The McCulloch-Pitts formal neurons will be used. Each of these neurons is characterized by:

1. the synaptic contacts that the various axon endings establish with it, and
2. a threshold for the "firing" of bioelectric pulses [3] quantified by an integer number.

Time is thought of as being made up of elementary lapses. During each of these lapses a synaptic contact is either active or inactive. This depends on wether the axon making contact shows bioelectric activity or not (the presence of a "bioelectric pulse" or not).

From now on, "1" will indicate either a truth value "true" or the presence of a bioelectric signal; "0" will indicate a truth value "false" or the absence of a bioelectrical signal. Additionally, the activation of an excitatory synaptic contact will have a value of "+1", and the value of "−1" corresponds to the activation of an inhibitory synaptic contact.

If, and only if, the algebraic sum of all excitatory and inhibitory stimuli acting on a given formal neuron in a given elementary lapse (e.g. in the t–lapse) is greater than or equal to the threshold value, then the neuron will "fire". In other words, it will generate a bioelectric signal in the next lapse (the $(t+1)$–lapse). This signal is then carried by the neuron's axon (its output channel).

We should note that with this approach the time it takes to carry the signal along the nervous pathways is null or negligible compared to the "unit time" (elementary lapse). The synaptic delay and the elaboration of the bioelectric activity that corresponds to the next elementary lapse take place during such unit time.

[3] The expresion "bioelectric pulse" or "bioelectric signal" (the physiologists' action potential) used in this paper acknowledges the origin of the neural nets (or neural networks) theory which came out of the explanation for certain functional aspects of the nervous system. This also applies to the terms "axon" (the output channel or exit pathway of a neuron), "synaptic contact" and "threshold". In this paper, the presence or absence of a bioelectric pulse will be indicated by a "1" or a "0", respectively.

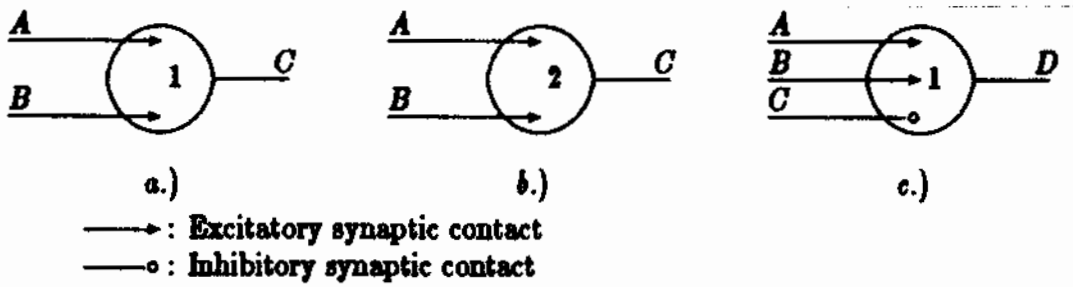


Figure 7

Note that the neuron shown in **Figure 7***a* carries out the logical sum or disjunction $C(t+1) = \vee\{A(t), B(t)\}$, and the one in **Figure 7***b* performs the logical product or conjunction $C(t+1) = \wedge\{A(t), B(t)\}$.

A disjunction neuron with an arbitrary number of input channels can be synthesized as follows:

1. For each input channel that should be active (i.e. carries a bioelectric sign) in a given elementary lapse in order to get a firing of the neuron in the next lapse, we assign an excitatory synaptic contact. This contact is made between the input channel and the neuron.

2. For each input channel that should remain inactive in a given elementary lapse in order to get a firing of the neuron in the next lapse, we assign an inhibitory synaptic contact between the channel and the neuron; and

3. The formal neuron threshold value equals the difference between 1 and the number of inhibitory synaptic contacts for that neuron.

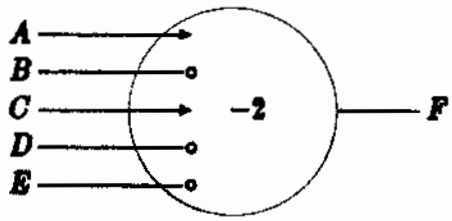


Figure 8

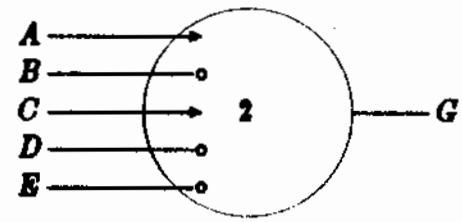


Figure 9

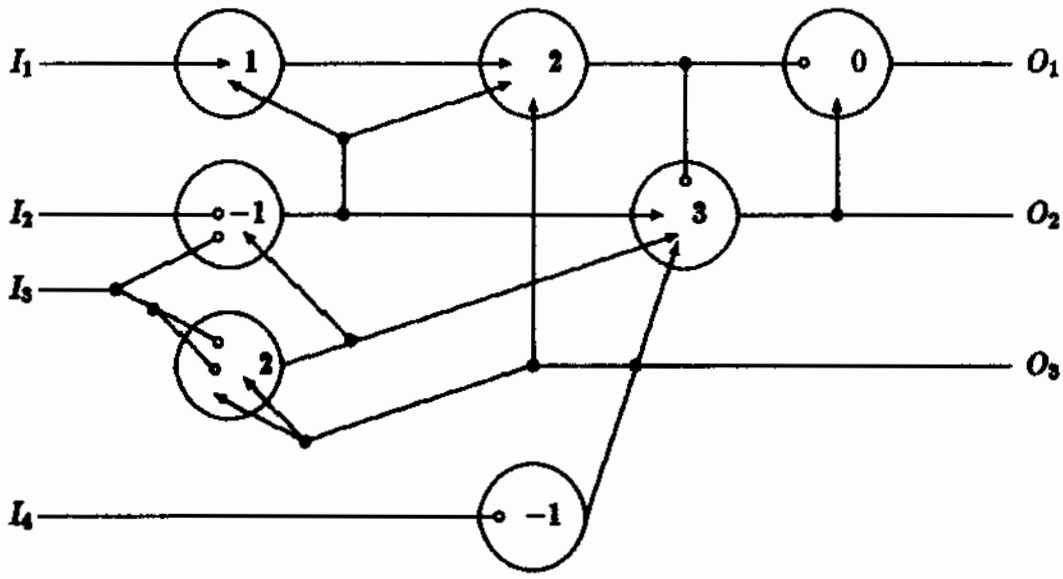


Figure 10

To ilustrate, in **Figure 8** we show a formal neuron which computes

$$F(t+1) = \vee\{A(t), \neg B(t), C(t), \neg D(t), \neg E(t)\}$$

The same criteria outlined above for the disjunction neuron hold for a conjunction neuron with an arbitrary number of input channels with one exception. This is that the threshold value should equal the number of excitatory synaptic contacts.

Figure 9 shows a formal neuron which computes

$$G(t+1) = \wedge\{A(t), \neg B(t), C(t), \neg D(t), \neg E(t)\}.$$

The disjunction (or conjunction) of n logical functions can always be determined by first finding out the disjunction (or, respectively, conjunction) of any two of these functions. Then, we find out the disjunction (resp., conjunction) between the result of this operation and any of the $(n-2)$ remaining functions, and so on.

Figure 10 illustrates a neural net where the various connections have been established in an arbitrary way. Notice the large number of possibilities, even with a few neurons.

Figure 11 is a general representation of a neural net with m input channels and n output channels (m and n are positive integers).

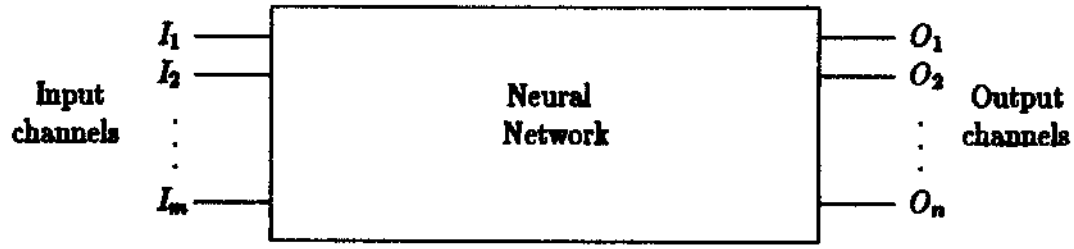


Figure 11

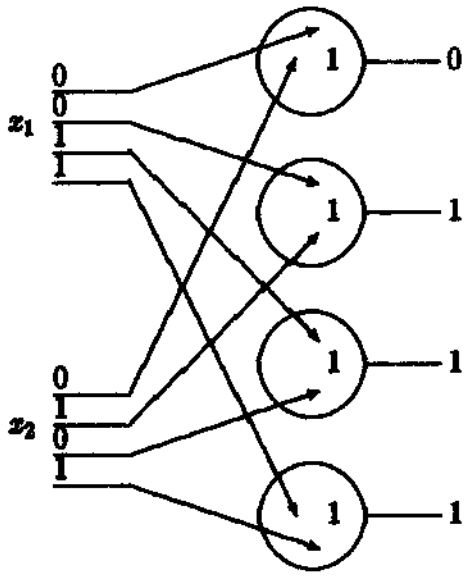
The input channels represent nerve fibres (axons) which carry bioelectric signals, generated somewhere outside the net. The net can emit bioelectric signals by means of its output channels –the axons of some neurons which belong to the net.

The question is: from a bioelectric point of view, what can happen in each input or output channel during each elementary lapse? There are two possibilities: either there is a presence or an absence of a bioelectric signal.

To specify a configuration for the signals entering a neural net, means to establish the presence or absence of a bioelectric signal for each input channel and for each lapse (from the initial to the last). This also holds for the output channels.

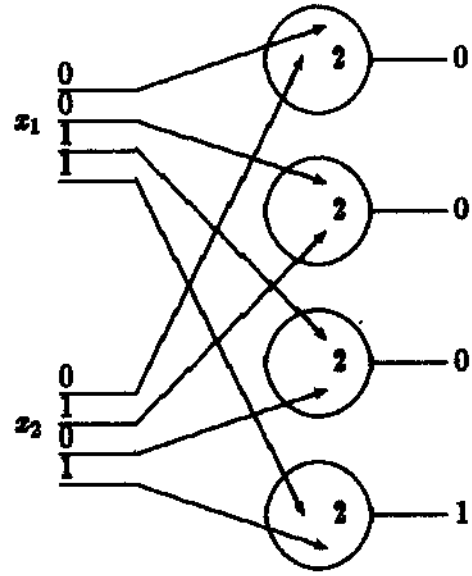
Let us now return to our binary approach to the calculus of logical functions. We will use as an example the functions $\vee\{x_1, , x_2\}$, $\wedge\{x_1, , x_2\}$, $\neg\{x_1\}$, $\overset{*}{\rightarrow}\{x_1, , x_2\}$, and $\leftrightarrow\{x_1, , x_2\}$ of $\mathcal{L}^{(2)}$. These functions can be computed using the neural nets previously described. This is illustrated in **Figures** 12, 13, 14, 15, and 16, respectively.

It is exactly the same for $\mathcal{L}^{(n)}$ as in $\mathcal{L}^{(2)}$. Thus, based on the neural net shown in **Figure** 17 which computes $C(t+1) = \leftrightarrow\{A(t), B(t)\}$, we can compute $\leftrightarrow\{f_i, f_j\}$ for a given pair of functions f_i and f_j of $\mathcal{L}^{(n)}$ as shown in **Figure** 18; where $f_i = (k_1 \ldots k_{2^n})$, and $f_j = (l_1 \ldots l_{2^n})$.



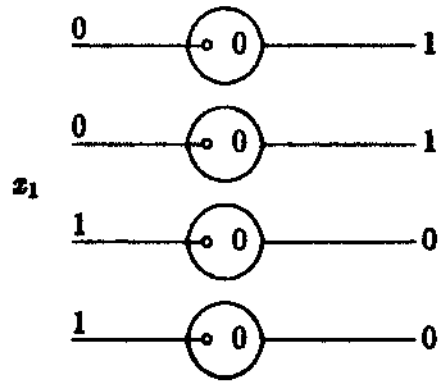
$f_7 = \vee\{x_1, x_2\} = (0111)$

Figure 12



$f_1 = \wedge\{x_1, x_2\} = (0001)$

Figure 13



$$f_{12} = \neg\{x_1\} = (1100)$$

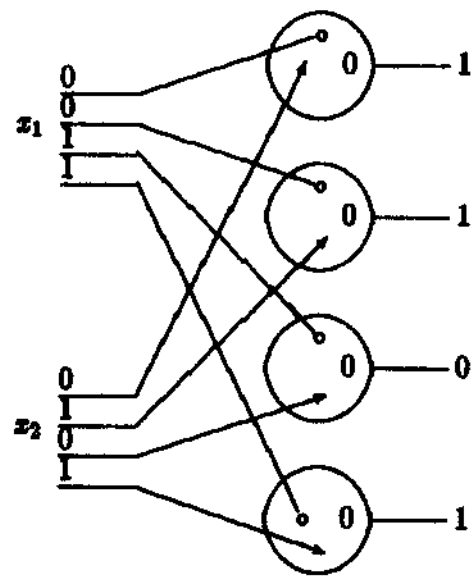
Figure 14

This procedure can be used as many times as desired. So, for example, to calculate the function:

$$f = \leftrightarrow \{\vee\{f_i, \overset{*}{\rightarrow}\{f_j, f_k\}\}, \overset{*}{\rightarrow}\{\neg\{\wedge\{f_j, f_k\}\}, f_i\}\}$$

we can use the procedure illustrated in **Figures 19***a*, *b*, *c*, *d*, *e* and *f*.

We should note here that most of the current computers have a conventional von Neumann design. That is to say, they have a sequential operative modality. When we proceed to simulate in these computers neural type devices, the "in parallel" operations that the neural nets can accomplish (in principle), will be performed in a sequential manner. Thus, when using sequential type computers in simulating neural nets there is no gain in "real time", even though these neural nets have the potential to perform many logical operations simultaneously. Only by using parallel-processing computers to simulate neural



$$f_{13} = \overset{*}{\rightarrow}\{x_1, x_2\} = \vee\{\neg x_1, x_2\} = (1101)$$

Figure 15

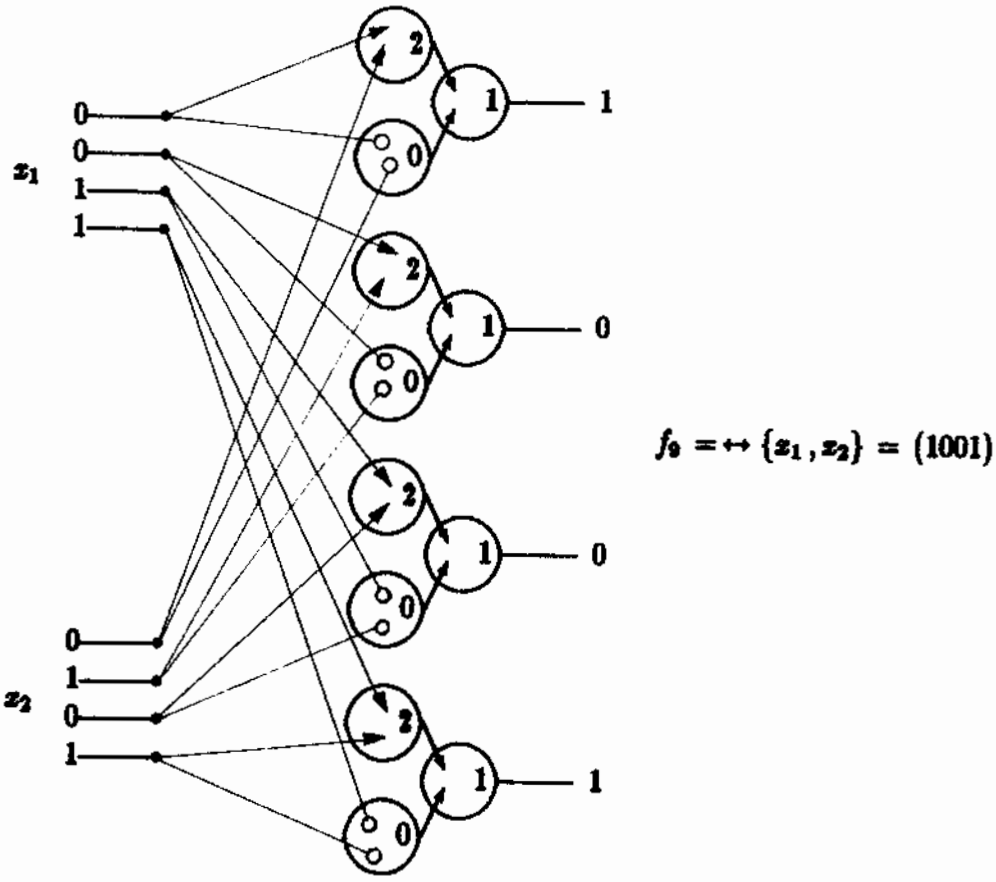


Figure 16

nets can we exploit all the potential of the last. Even so, when working with sequential computers, we can save computer memory when computing using neural nets simulation, by minimising the number of formal neurons to work with. Thus instead of using the net shown in **Figure 13** for computing the conjunction $\wedge \{x_1, x_2\}$, we can use a net consisting of just one formal neuron with two input channels and one axon (output channel –see **Figure 20**).

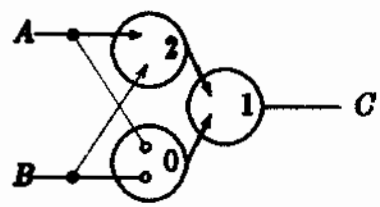
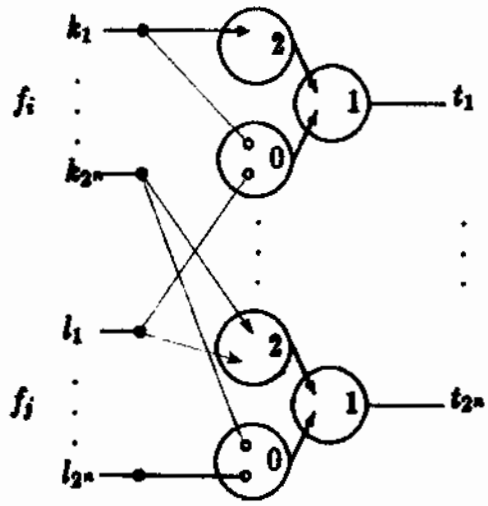


Figure 17



$$\leftrightarrow \{f_i, f_j\} = (t_1 \ldots t_{2^n})$$

Figure 18

The simulated formal neuron first calculates the conjunction for the first digits of x_1 and x_2 $(0 \wedge 0)$. The output is 0 which is identified by the absence of a bioelectric pulse and is the first digit of $\wedge\{x_1, x_2\}$; then, the neuron does the same process with the second digits, and so on. Therefore, those operations illustrated in **Figures 12, 13, 14, 15** and **16** can be performed by the nets shown in **Figures 21***a*, *b*, *c*, *d* , and *e* , respectively.

4 Further Comments

Propositional Calculus (isomorphic to Boolean Algebra) has become important not only because of its role in basic logic, but also for its direct application in designing digital circuits. It is therefore important to have a tool like that presented in this paper which works simply, and systematically with many variables' logical functions. The traditional approach (truth tables) is difficult to deal with even with a small number of variables. For example, with seven variables, we need a table with 128 (2^7) rows. With our approach, it is easy to work with seven variables even manually. Now, if we use devices such as the neural nets described above and simulate them on the computer, it is quick and easy to work with as many as a dozen variables.

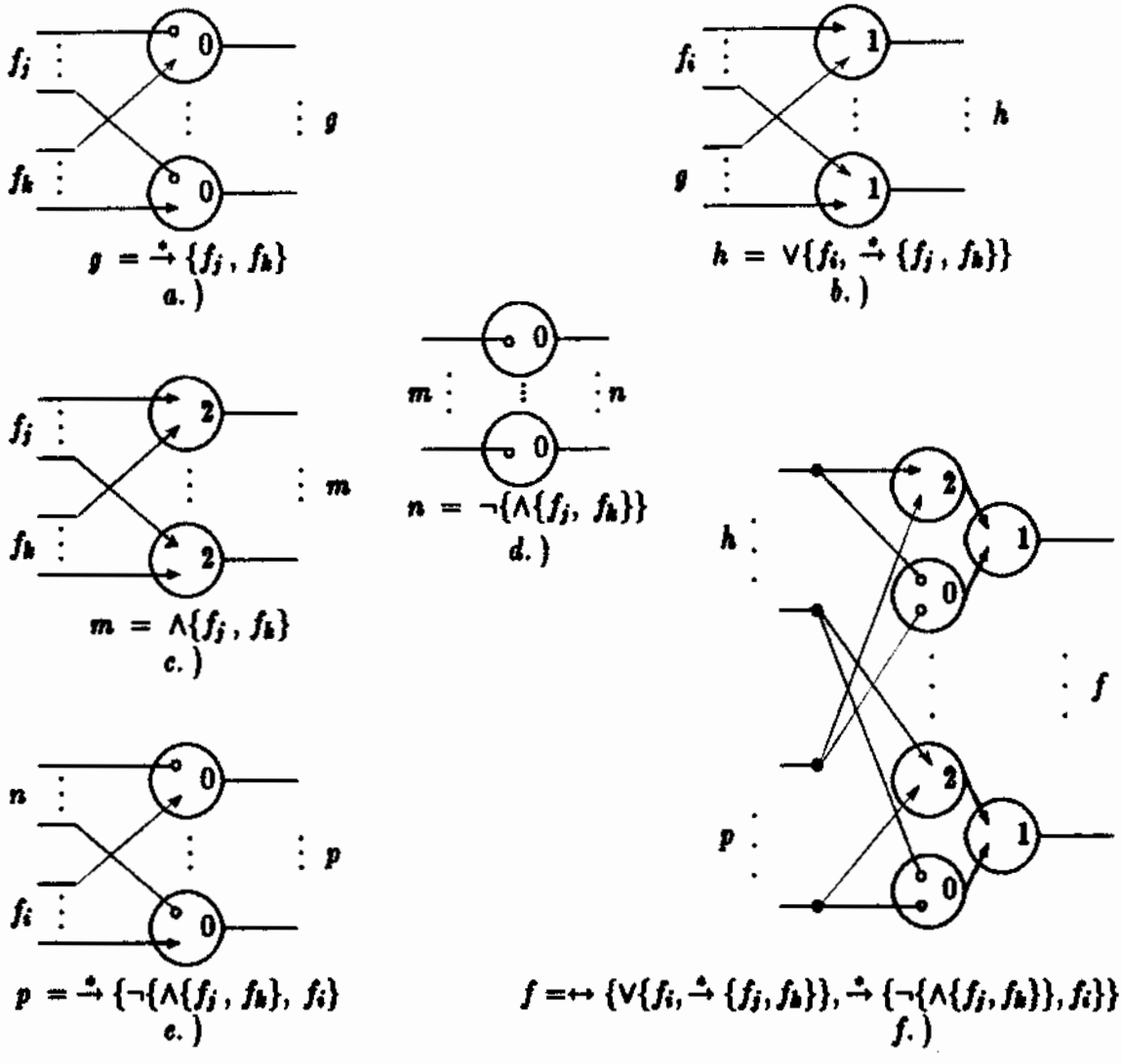


Figure 19

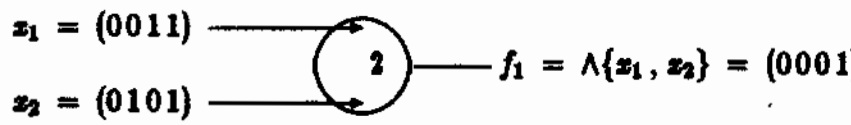


Figure 20

There has recently been a tendency to miniaturise and integrate more and more complex digital circuits while, at the same time, devices oriented towards parallel processing of information are developed. Both trends reflect growing needs to process huge amounts of data in a quick and efficient manner. This is especially true with situations that demand control in "real time" of different types of complex processes (processes which require,

either because of convenience or unavoidability, the use of logical functions with a large number of variables to model and/or control them. These are occuring more and more frequently.). The approach presented here fits well with these needs and serves as a tool for their development.

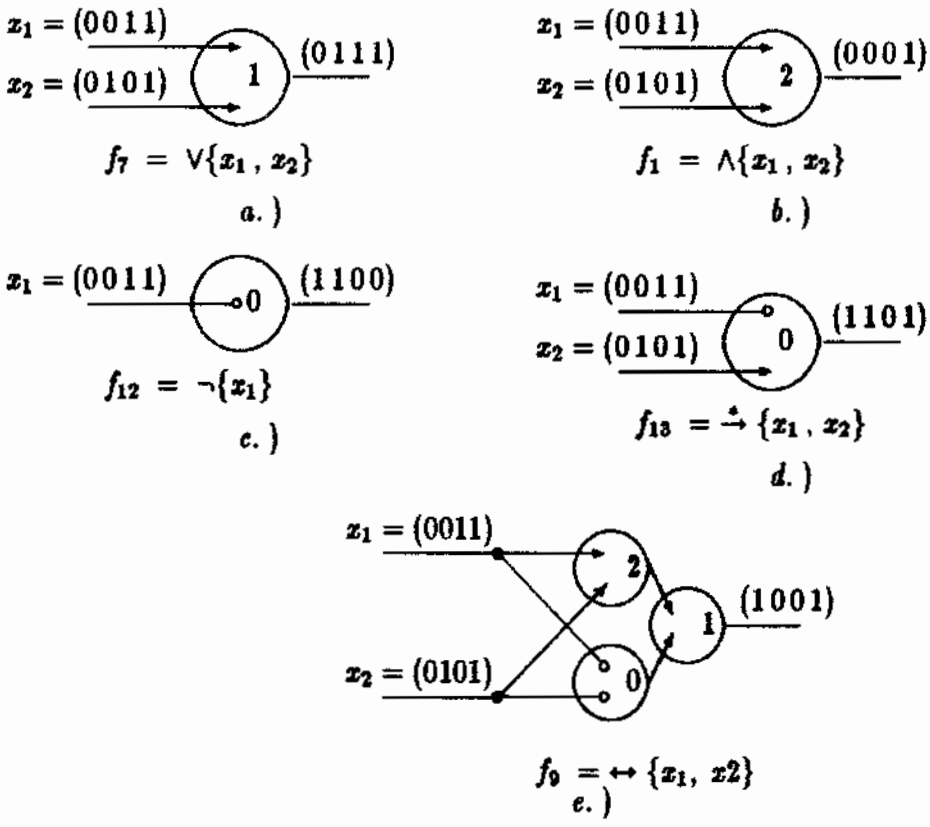


Figure 21

REFERENCES

[1] Randolph, J. F., *Cross-examining Propositional Calculus and Set Operations*, Amer. Math. Monthly 72, No.2 (1965) 117–127.

[2] Skliar, O., M. R. Nachón and V. Eandi, *Técnicas de Síntesis de Redes Neurales Determinísticas*, Investigación Operativa, Bulletin of the Argentine Operational Research Society 31, (1974) 5–38.

[3] Von Foerster, H., *Computation in Neural Nets*, Currents in Modern Biology 1, No.1 (1967) 47–93