A THRESHOLD GATE FEED-FORWARD SWITCHING NET ALGORITHM

Thesis by

Gordon Frierson Hughes

In Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1964

ACKNOWLEDGEMENTS

I would like to express my gratitude for advice and assistance to Dr. K. Lock, Dr. R.E. Miller, Dr. G.D. McCann, the Corning Glass Company, the Radio Corporation of America, and particularily to my wife Patricia.

ABSTRACT

A general algorithm is presented for the efficient computation of feed-forward nets of general threshold gates which realize given bi-valued switching functions. A simplified version of the algorithm is presented for the case of symmetric threshold nets which realize symmetric switching functions.

These algorithms produce near-minimal gate nets, and the results of a digital computer program for the general algorithm are presented to illustrate the degree of efficiency and minimality obtained in practice.

Both algorithms are proved to give a minimal one-gate net if one exists for a given switching function; a necessary criterion is given for the symmetric algorithm to produce a minimal two-gate net if one exists; and two-gate minimality is also demonstrated for the general algorithm, for a certain class of two-gate switching functions.

The case of partially defined switching functions is also treated.

TABLE OF CONTENTS

Part	Title	Page
I	Introduction	1
II	Problem Statement	5
III	Existence Calculation	12
IV	Tableau Relationships	26
V	Adjoining Calculation	38
VI	Experimental Results	54
VII	Conclusions	57
VIII	References	64

Appendices

I	List of Symbols	65
II	Summary of Algorithm Rules	68
III	Partially Defined Switching Functions	72

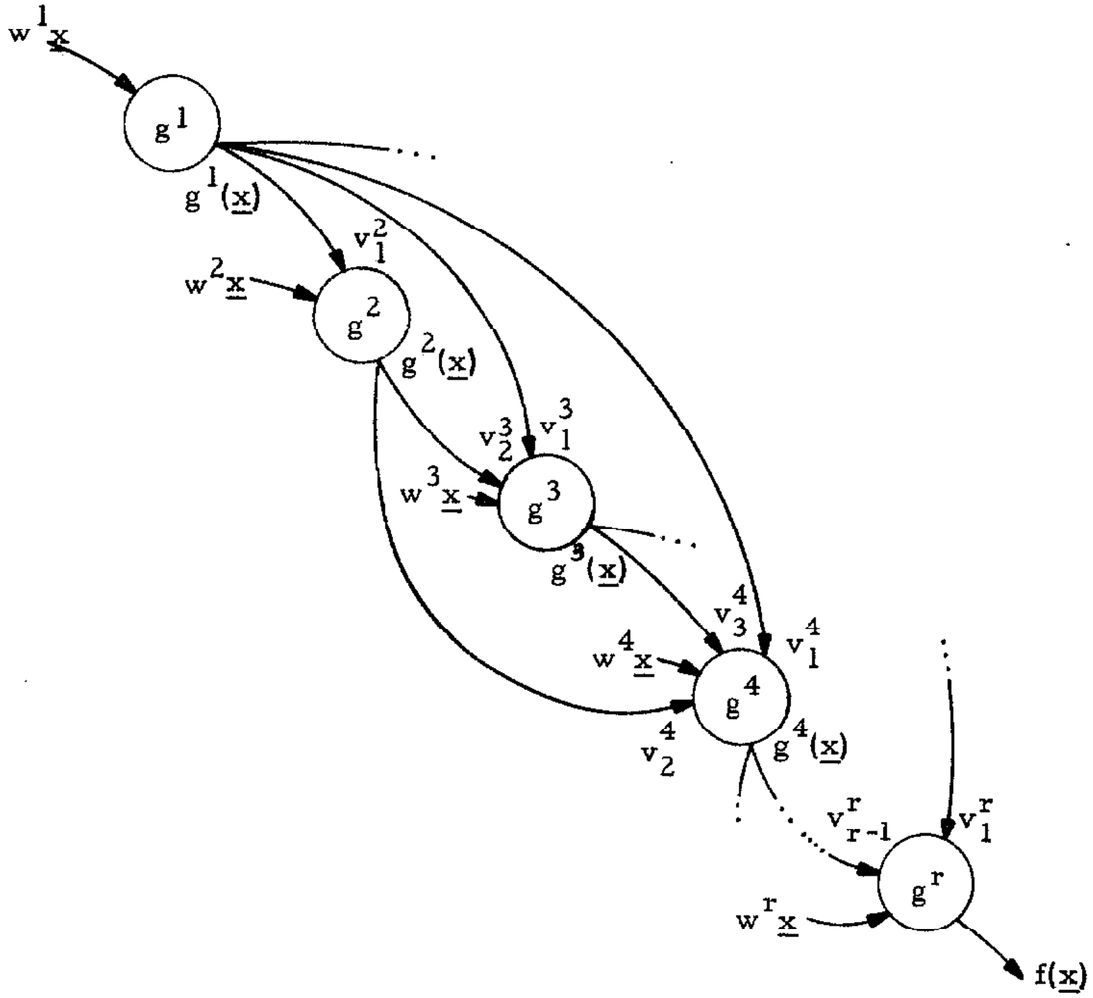
I. INTRODUCTION

An algorithm will be developed for the computation of general feed-forward nets of general threshold logic gates (fig. 1), which realize given bivalued switching functions with a near-minimal number of gates (2, 5, 6).

A threshold gate in such a net is defined by a weighted linear sum of bivalued logic variables (either direct inputs, or outputs of other gates), including a bias weight; i.e., for the s$\underline{\text{th}}$ gate in the net of fig. (1), the sum is:

$$\sum_{h=1}^{n} w_h^s x_h + \sum_{h=1}^{s-1} v_h^s g^h(\underline{x}) + w_{n+1}^s ,$$

where n is the number of direct logic variables x_h, which take on all 2^n combinations of input states using (-1, +1) logical values. Grouping the x_h into a vector, these input states are $\underline{x} = (x_1, x_2, \cdots, x_n)$ $= (-1, \cdots, -1, -1), (-1, \cdots, -1, 1), \cdots, (1, \cdots, 1, -1), (1, \cdots, 1, 1)$. The w_h^s parameters in the sum are the weights associated with input of the direct logic variables x_h to gate s, w_{n+1}^s is the bias weight, $v_h{}^s$ are the weights associated with the inputs of the lower-level gate outputs $g^1(\underline{x}), g^2(\underline{x}), \cdots, g^{s-1}(\underline{x})$; and $g^h(\underline{x}) = \pm 1$ are the logical outputs of each gate as a function of the input state vector (we use $\underline{x}$ here for brevity, in that if the lower-level gates are determined, the output of gate g^h is a function of $\underline{x}$ only). These gate outputs are obtained by performing a non-linear switching operation on the input sum for each gate; viz., set $g^s(\underline{x}) = +1$ if its input sum attains or exceeds (+1), and set $g^s(\underline{x}) = -1$ if its sum is less than or equal to (-1), for each input state ($\underline{x}$). The sum



$$g^s(\underline{x}) = \begin{cases} +1 \text{ if } \sum_1^n w_h^s x_h + \sum_1^{s-1} v_h^s g^h(\underline{x}) + w_{n+1}^s \geq +1 \\ -1 \text{ if } \sum_1^n w_h^s x_h + \sum_1^{s-1} v_h^s g^h(\underline{x}) + w_{n+1}^s < -1 \end{cases}$$

Figure 1. General Threshold Gate Feed-Forward Net

is not allowed to take on any value in the range from (-1) to (+1); this amounts to a threshold tolerance band.

We will require, in practice, all weights (w_h^s, v_h^s) to be signed integers; a negative weight will correspond to an inverted logic variable. This requirement is computationally convenient, and arises from the physical properties of many actual threshold gate devices (5).

It can be seen that there are a large number of assignable parameters in threshold nets such as shown in fig. (1), as compared (say) to Boolean AND/OR gates (which constitute a small subset of the class of threshold gates). For this reason, it is believed that a general net design algorithm which can be efficiently programmed on a digital computer is almost mandatory to permit effective use of threshold gate nets.

Computational efficiency is believed to be an important requirement for such an algorithm, and for this reason we will require the algorithm to be a first order recursive procedure where each step depends only on the one immediately preceding it, and also that the algorithm shall be "forward", in the sense that no "blind alleys" shall be encountered in its application. These are not trivial restrictions for nonlinear problems.

Naturally, the requirement of programmability will make inefficient a large class of hand-feasible algorithms: viz., those that require heuristic "inspection" in order to determine which computational path to follow.

The algorithm to be presented below is believed to satisfy the above constraints, and will be shown to produce near-minimal gate nets in practice, at least in those cases where comparisons to the literature exist; e.g., Minnick (5).

In two cases, algorithm criteria will be developed to produce minimal two gate nets, if such a net exists for a given switching function. These are the class of logic functions and gates which are functionally invariant to permutations of their direct input variables (symmetric functions), and a certain class of general logic functions which is defined below (section V).

A Fortran program for the algorithm has been written, and the results will be discussed for the purpose of providing a practical estimate of computational efficiency and net minimality.

The case of partially defined switching functions will be separately discussed (Appendix III) as a modification to the general algorithm.

II. PROBLEM STATEMENT

We will use a matric formulation of the linear threshold equations for the weight vector (w^s, v^s) of the $s^{\underline{th}}$ gate in the net of fig. (1). Thus, the input to the gate is the product of the weight vector and the logic variable vector; viz., (w^s, v^s) $(\underline{x}, \underline{g})$, where the vector $\underline{g}$ is composed of the ordered outputs of gates $g^1, g^2, \cdots, g^{s-1}$ for the input state $\underline{x}$. To avoid the confusion of row and column variants of a vector, we shall interpret a premultiplying vector as a row vector and a postmultiplying vector as a column vector (some vectors may appear in both types).

The notation to be used is defined in Appendix I. Capital letters are used to denote matrices (A), whose elements are lower case letters with an ordered pair of subscript indices (a_{ij}), the first pertaining to the row and the second to the column. Sets are also denoted by capital letters which may also be used for matrices, because the intended interpretation will be clear. Furthermore, we will speak of gate $g^s(\underline{x})$ and switching function $f(\underline{x})$, and thereby mean the switching function (or a modified one) represented by $g^s(\underline{x})$ or $f(\underline{x})$.

Lower case letters with either one subscript index or none refer to vectors whose components are indexed by a single subscript, whether exhibited or not $({g^s}_j, g^s)$. Where necessary, row and column vectors (which do not appear in a product for type identification) will be distinguished by using h, i as row indices (e^h), and j, k as column indices (e^j), or by context. The unit vector e has all components equal to unity, and its length and orientation (row or column) will be defined by the context in which it appears.

Partitioned matrices will be frequently used, with the usual straight lines denoting the partitions.

As stated above, we will successively compute each gate g^s in the net, allowing inputs to g^s from the input variables x_h and from the previously determined lower-level gates g^h, with $h < s$. This computation has two phases: first we attempt to find a gate g^s which linearily separates (realizes) $f(\underline{x})$, the given switching function. This phase is defined to be an existence calculation, and if it succeeds we have finished, and $s = r$.* A simple criterion will be exhibited which demonstrates when no such g^s exists; i.e., the equation $g^s(\underline{x}) = f(\underline{x})$ cannot be solved for all $\underline{x}$. In this case, we proceed to a second computational phase, defined to be an adjoining calculation, where we attempt to find a g^s which is such that a minimal number of additional gates will be required to realize $f(\underline{x})$. This is the nonlinear part of the problem.

Our minimality criterion is therefore that we seek a net for $f(\underline{x})$ with a minimal number (r) of gates. It would also be of interest to require that, subject to the above constraint, the sum of the absolute values of the weights for each gate be minimal. There are ways to attempt to meet this requirement (such as "polishing" each gate for minimal weights after computing the net, using linear programming), but the present algorithm turns out to be fairly minimal in this respect also.

We express the threshold equations for gate g^s in a form that is suitable for the existence calculation; i.e., we assume that $s = r$.

* cf. fig. (1): $g^r(\underline{x}) = f(\underline{x})$.

Consequently, we seek a (row) weight vector (w^s, v^s) such that this weighted linear sum of logic variables attains or exceeds (+1) for the true states of $f(\underline{x})$ $(f(\underline{x}) = 1)$, and is less than or equal to (-1) for the false states $(f(\underline{x}) = -1)$. We reverse the signs of the true state inequalities, and write the conditions on g^s as

$$(w^s \mid v^s) \left[\frac{A}{G_{s-1}} \right] \leq -(\; e \;), \qquad (1)$$

which gives $m = 2^n$ inequalities in n+s variables. The parameters are defined as follows: the n+1 by m (rows, columns) matrix A is formed from an input state matrix B by first adjoining a unit row vector (e), and then setting column $a^j = -b^j$, $j \epsilon T$; $a^j = b^j$, $j \epsilon F$; for $j = 0, 1, 2, \cdots, m - 1$. The index sets T, F are defined to contain those indices j such that the $j^{\underline{th}}$ input state of the truth table for $f(\underline{x})$ is a true or false state, respectively; the unit vector will correspond to an identically (+1) input logic variable for the bias (threshold) component w^s_{n+1}; and the above also accounts for the sign reversal of the true state inequalities. The input state matrix B represents the truth table of n variables, and is obtained by writing the binary numbers zero to m - 1 (corresponding to a binary expansion of j) as ordered columns, with the exception that a binary coefficient zero is replaced by (-1):

$$B = \begin{bmatrix} -1 & -1 & -1 & -1 & \cdot & 1 & 1 & 1 \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ -1 & -1 & 1 & 1 & \cdot & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & \cdot & 1 & -1 & 1 \end{bmatrix} \qquad (2)$$

Note that the most significant digit is at the top, so that we have ordered the components of w^s such that w_1^s corresponds to the most significant digit (slowest varying) of the truth table.

If we permute the columns of A so that the true columns ($f(\underline{x}) = 1$) are at the left, and the false columns ($f(\underline{x}) = -1$) are at the right, then we may write A in terms of the true (T) and false (F) state matrices of B:

$$A = \left[\begin{array}{c|c} -T & F \\ \hline -e & e \end{array}\right], \tag{3}$$

which exhibits the sign reversal operation more clearly. We shall not actually make this permutation in practice.

The matrix G_{s-1} in inequality 1 provides the modified outputs of the lower level gates to g^s, and has these modified gate state vectors as rows, viz.:

$$G_{s-1} = \begin{bmatrix} g_0^1 & g_1^1 & g_2^1 & \cdots & g_{m-1}^1 \\ g_0^2 & & \cdots & & \\ \cdot & & & & \\ \vdots & & & & \\ g_0^{s-1} & & & \cdots & g_{m-1}^{s-1} \end{bmatrix}, \tag{4}$$

where the rows are determined by

$$g^h = \operatorname{sgn}\left\{ (w^h \mid v^h) \left[\frac{A}{G_{h-1}}\right] \right\}, \tag{5}$$

and the "sgn" operator replaces each component of the resultant row vector within the brackets by (+1) or (-1), according to whether that component attains or exceeds (+1), or is less than or equal to (-1), respectively; i.e., it is a sign-taking operator which represents the switching function performed by a threshold gate with this weight vector. G_0, of course, is the null matrix.

It is important to note that the sign reversal of the "true" inequalities in 1 is carried throughout the entire analysis; e.g., the vector components g_j^h represent the actual output of gate g^h for $j \epsilon F$, but represent the inverted (negative) output if $j \epsilon T$; this is what is meant by a modified gate function.

The sgn operator is commutative for (-1, +1) logic states in the sense that the sign reversal of the true state inequalities may be carried past the sgn operator and into the linear inequalities for each gate, and this is why these particular logic states are used. Further note that the quantity in brackets in inequality 5 cannot satisfy inequality 1 for $h < r$.

We will define an allowable weight vector as one which satisfies inequality 1 if the negative of the absolute value of the left hand side is taken; i.e., the corresponding gate may not realize $f(\underline{x})$, but the threshold tolerance constraint mentioned above is satisfied (this is a constraint on the adjoining calculation).

Thus, if no solution exists to inequality 1, we replace the inequality by $\left| w^s A + v^s G_{s-1} \right| \geq e$, perform an adjoining calculation according to rules developed below, compute the modified state vector of the gate to be adjoined to the net, adjoin it as a new bottom row to G_{s-1}, and initiate an existence calculation for g^{s+1}.

We digress to give an example. For n = 2, the input state matrix B is:

$$B = \begin{bmatrix} -1 & -1 & +1 & +1 \\ -1 & +1 & -1 & +1 \end{bmatrix}, \tag{6}$$

and for the two-variable AND function ($x_1 x_2$), the only true state is j = 3, i.e., $\underline{x} = (1, 1)$. Thus we adjoin the unit row vector to B and then reverse the sign of the last column, obtaining

$$A = \begin{bmatrix} -1 & -1 & +1 & -1 \\ -1 & +1 & -1 & -1 \\ +1 & +1 & +1 & -1 \end{bmatrix}. \tag{7}$$

Since this is a one-gate function, the s = 1 existence calculation will succeed.

For switching functions $f(\underline{x})$ which are symmetric in their variables; i.e., functionally invariant to variable permutations, it has been shown (1) that if a one-gate realization exists for a symmetric $f(\underline{x})$, then an equal weight ($w_1^1 = w_2^1 = \cdots = w_n^1$) one-gate realization also exists. While this has apparently not been extended to symmetric functions requiring feed-forward nets of more than one gate, there is interest (2) in threshold gate nets whose gates are symmetric (defined as above). In this case, we set the direct input weights equal to each other (but not w_{n+1}^s or v_h^s), which is equivalent to adding the corresponding rows of B together, replacing them by a single sum vector. It is then apparent that there are only n+1 distinct constraints in inequality 1, one for each permutation set of 0, 1, 2, $\cdots$, n variables true ($x_h = 1$). Thus we may define a special case for such symmetric nets, using an s+1 variable weight vector (w_x^s, w_b^s, v_1^s, v_2^s, $\cdots$, v_{s-1}^s), and using

$$B = [-n, \ -n+2, \ -n+4, \ \cdots, \ n-2, \ n] \ , \qquad (8)$$

where w_x^s is the common weight for all direct inputs, and w_b^s is the bias value.

Aside from this change, the inequalities and definitions are as before.

For an example of the symmetric case, note that the AND function which gave equation 7 is symmetric, and has a symmetric A matrix given by

$$A = \begin{bmatrix} -2 & 0 & -2 \\ +1 & +1 & -1 \end{bmatrix} \qquad (9)$$

III. EXISTENCE CALCULATION

Since we will compute the net gates beginning with g^1, we first consider a linear programming method for solving inequality 1, assuming that $r = s = 1$. This method will also suffice for any existence calculation ($s \geq 1$). Familiarity with linear programming in general, and the Simplex method in particular, will be assumed (3, 4). However, the complete algorithm rules will be found summarized in Appendix II in a manner which does not require this knowledge.

Now, it is computationally quite inefficient to attempt to solve inequality 1 directly using the Simplex method. Instead, we begin with a method of Gale (3, p 121), which reduces the number of arithmetic operations required, by a factor which empirically has the order of $2^{2n}/n$ (over a direct solution to 1).

A theorem of linear inequalities (3, p 46) states that $wA \leq -e$ has no solution if and only if there exists a semipositive (non zero) vector y such that $Ay = 0$, $ey = 1$. We choose the latter set of equations to be the primal in a Simplex calculation, and then show that a simple criterion exists which indicates when no feasible y exists and which then exhibits a feasible solution (w) as the dual variables in the Simplex tableau.

For the single gate case, this method is merely a more efficient way for applying Simplex to inequality 1 than that first proposed by Minnick (5); a measure of this gain in efficiency is implicitly given by the fact that the method is feasible for hand calculation for up to four logic variables.

In order to obtain a first feasible solution to initiate Simplex, we adjoin one artificial vector (4, p 61), and define the following to be the primal problem:

Find $y_a = (y_o \mid y) \geq 0$

such that y_o is minimal, and

$$\underline{A}y_a \equiv \left[\begin{array}{c|c} 1 & e \\ \hline 0 & \\ 0 & \\ \cdot & A \\ \cdot & \\ \cdot & \\ 0 & \end{array}\right] \begin{pmatrix} \underline{y_o} \\ \\ y \\ \\ \end{pmatrix} = \begin{pmatrix} \underline{1} \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{pmatrix}. \tag{10}$$

Note that min (y_o) will be zero if $Ay = 0$ has a semipositive solution, and unity if it does not; the initial artificial vector solution being $y_o = 1$, $y = 0$.

The dual problem to equation 10 is:

Find $(w_o \mid w)$

such that w_o is maximal, and

$$(w_o \mid w) \left[\begin{array}{c|c} 1 & e \\ \hline 0 & \\ 0 & \\ \cdot & A \\ \cdot & \\ \cdot & \\ 0 & \end{array}\right] \leq (100 \cdots 0) \tag{11}$$

We know that max (w_o) = min (y_o), so that if $Ay = 0$ has no semipositive solution, then $wA \leq -w_o\,(e) = -e$ has a solution w, which is inequality 1. Refer to Gass (4, p 71) for a discussion of duality in linear programming.

Expressing equation 10 in a Simplex tableau (3, Ch. 4), and adjoining a unit matrix (in order to carry along the dual problem), we obtain the tableau in fig. (2a). Here, Λ is the n+2 by m primal tableau matrix, y_a is the solution column, and Γ is the n+2 by n+1 dual tableau matrix (inverse basis). Note, however, that we have explicitly exhibited the top tableau row (λ^o, γ^o), and these are to be considered as part of Λ and Γ. The bottom row c is the relative cost vector (3, pp 107-110). Some of the vectors explicitly shown in fig. (2a) are omitted from the working tableau (fig. 2b), since they will never be altered in the computation. Note that we have numbered the top tableau row zero so that the row numbering of A (in $\underline{A}$) will not change (cf. equation 10).

Now, the left hand unit vector in fig. (2a) will not be altered, so that the first column of $\underline{A}$ (which multiplies y_o) will always be in the primal basis (defined to be those columns of $\underline{A}$ which correspond to unit vectors in Λ). Since the cost function is y_o = minimum, this means that the relative cost row c is equal to the top row of the tableau (except for the omitted vectors), by applying the usual Simplex rule for adjoining the cost row in relative (basic) form (3, pps 109-110). Consequently, we also omit c from the working tableau of fig. (2b). It is consistent in terms of Simplex to consider the top primal row of the tableau as a cost row (it will not be pivoted on).

The dual portion of c contains the dual variables in our case, and we thus observe that the top dual row (γ^o) may be considered to be a weight vector (w) (4, p 72).

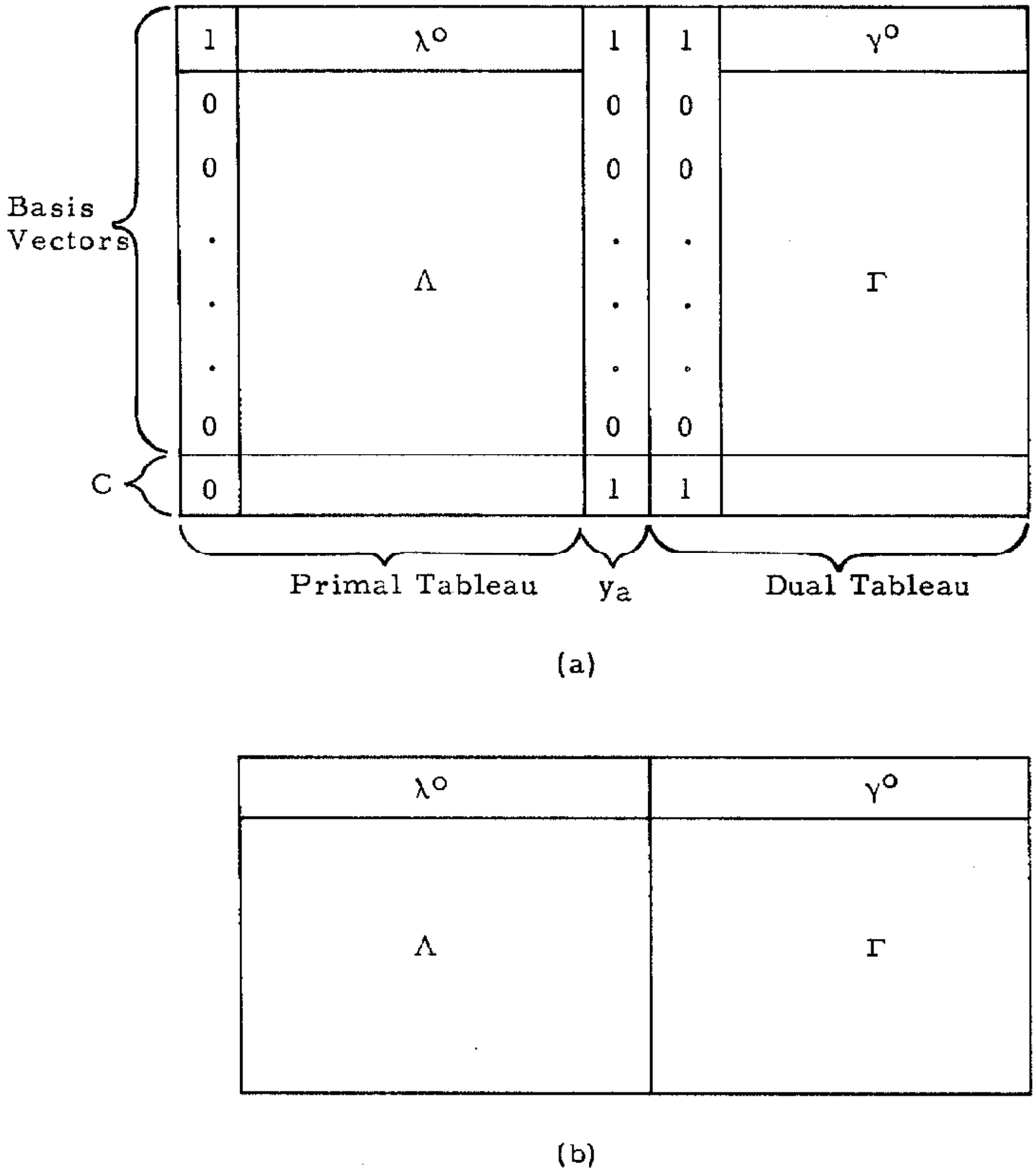


Figure 2

Simplex Tableau Format—(a), unmodified; (b), working tableau

We will henceforth assume that fig. (2b) is meant when referring to a tableau, unless specifically noted.

If we index the tableau matrices with an iteration number $q = 0, 1, 2, \cdots$, then the first tableau is $\Lambda(0) = \underline{A}$, $\Gamma(0) = U$, where U is the n+2 unit matrix with its first column omitted, and is carried along in the tableau to generate the inverse basis, for reasons quite analogous to the Gauss-Jordan reduction for matrix inversion. The first column of $\underline{A}$ is also omitted from the tableau.

For an example, the AND function, whose A matrix is given by equation 7, has a first tableau of

$$\begin{array}{|cccc|ccc|}\hline 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & -1 & 1 & 0 & 0 \\ -1 & 1 & -1 & -1 & 0 & 1 & 0 \\ 1 & 1 & 1 & -1 & 0 & 0 & 1 \\ \hline\end{array} \tag{12}$$

In the symmetric case, the top primal row is as before, and the two-row A matrix obtained from equation 8 is used as the bottom two primal rows. The dual matrix is a 3 by 3 unit matrix with the first column omitted. This gives, for the symmetric AND matrix of equation 9:

$$\begin{array}{|ccc|cc|}\hline 1 & 1 & 1 & 0 & 0 \\ -2 & 0 & -2 & 1 & 0 \\ 1 & 1 & -1 & 0 & 1 \\ \hline\end{array} \tag{13}$$

There are two parts in the computational process for applying Simplex to the tableau, both involving a sequence of iterations, each of which is performed by adding multiples of a given pivot row (λ^i, γ^i) to all other rows (λ^h, γ^h), $h \neq i$, $i \neq 0$, in order to generate a unit vector in a given pivot column in the primal tableau (the pivot row is normalized so that the pivot element becomes unity, by dividing all tableau elements in the pivot row by the pivot element).

First, we must generate a basic tableau, which is defined to be any tableau which contains all of the n+1 unit column vectors e^2, e^3, $\cdots$, e^{n+2} (in any order and position) in the primal portion of the tableau. Note that the first unit vector, e^1, is already in the tableau (fig. 2a).

After once obtaining a basic tableau, the second part of the computation will involve successively pivoting from one basic tableau to another, in a sequence defined below.

The first computational part is equivalent, in our case, to selecting any sequence of n+1 non-zero pivot elements λ_{ij}, one from each row of the primal (except the top row), such that the unit vectors mentioned above will be generated by performing the indicated iterations. Note that by choosing one pivot from each row, each iteration will leave the previously generated unit vectors undisturbed.

Because the top row need not be pivoted on, we see that the solution column y_a remains as in fig. (2a), thus allowing the use of negative pivots if desired. This relaxation of the normal Simplex rule that only positive pivots are allowable results from the degeneracy of our problem, and we will use this fact later (although we will normally use only positive pivots).

Briefly, this Simplex rule arises from the requirement that y_a be semipositive; it can be seen by inspection of fig. (2a) that if the solution column y_a contained a positive number in the pivot row, then a negative pivot would generate negative solution vector components. This cannot happen in our case because the only non-zero y_a component in fig. (2a) is y_o (+1), in the top tableau row.

We now require that the top tableau row never be pivoted on at any stage of the computation (including the adjoining calculation), thus proving that the use of the working tableau of fig. (2a) is legitimate, as stated above.

In order to guarantee that such a basic tableau exists for all A matrices, we need

Theorem 1:

The rank of A is $p = n+1$, for any single valued logic function $f(\underline{x})$.

proof: Since A is $n+1$ by $m = 2^n$, its rank cannot be greater than $n+1$ for $n \geq 1$. To show that its rank cannot be less than $n+1$, we exhibit $n+1$ independent columns of A. It is sufficient for this to consider the matrix B with the unit row vector e adjoined, since A is obtained from this matrix by the reversal of column signs only. Take the $n+1$ columns numbered zero and 2^h, for $h = 0, 1, 2, \cdots, n-1$, and group them into a matrix C, viz.:

$$C = \begin{bmatrix} -1 & -1 & -1 & -1 & & 1 \\ & \cdot & \cdot & \cdot & \cdot \quad \ddots & \\ -1 & -1 & -1 & 1 & \cdot & -1 \\ -1 & -1 & 1 & -1 & \cdot & -1 \\ -1 & 1 & -1 & -1 & \cdot & -1 \\ 1 & 1 & 1 & 1 & & 1 \end{bmatrix}, \tag{14}$$

so that C contains each column of B with a single (+1) component, and column zero. We need only prove that C is nonsingular, and an easy way to do this is to make a nonsingular transformation on C by subtracting column zero from each of the others in turn, obtaining a matrix $\underline{C}$:

$$\underline{C} = \begin{bmatrix} -1 & 0 & 0 & 0 & & 2 \\ & \cdot & \cdot & \cdot & \cdot \quad \ddots & \\ -1 & 0 & 0 & 2 & \cdot & 0 \\ -1 & 0 & 2 & 0 & \cdot & 0 \\ -1 & 2 & 0 & 0 & \cdot & 0 \\ 1 & 0 & 0 & 0 & & 0 \end{bmatrix}, \tag{15}$$

which is clearly nonsingular since its columns are independent.

Because the Simplex procedure for obtaining n+1 unit vectors in a primal (basic) tableau succeeds if there are n+1 independent primal columns in A, we are thus assured that this part of the calculation will succeed for all logic functions.

Furthermore, if we solve the equations $wC = -e$ by hand, then we can obtain a basic tableau for B (representing $f(\underline{x}) \equiv -1$), as shown in fig. (3). Observe that Λ can be easily constructed by writing a skew-transposed binary matrix (truth table) in (0, 1) state format, then writing a row above it such that the sum of all elements in each primal column is unity, and then adjoining a zero top row. This tableau may be easily converted into a first basic tableau for any A matrix by reversing the sign of each primal column corresponding to a true state of $f(\underline{x})$, then setting the top element in these columns equal to two and restoring any modified unit vectors by pivoting (this is proved in theorem 5 below). This allows us to completely bypass the first part of the computation, and fig. (3) will also be used below in discussing the pivot constraint necessary to maintain integral and allowable weight vectors in the γ^o row of the tableau.

The second part of the existence calculation, after attaining a basic tableau, is a normal Simplex process, and is described by

Theorem 2:

A sequence of primal pivot elements $\lambda_{ij} > 0$, $i \neq 0$, such that $\lambda_{oj} > 0$, starting with a basic tableau, will either lead to a basic tableau with $\lambda_{oj} \leq 0$, $j = 0, 1, 2, \cdots, m-1$, or to a basic tableau where at least one primal column satisfies $\lambda_{oj} > 0$, $\lambda_{ij} \leq 0$, $i = 1, 2, \cdots, n+1$. In the former case, row γ^o is a feasible weight vector w, and in the latter case no feasible w exists.

proof: We have already noted that the primal portion of the top tableau row is the relative cost vector, and that the dual portion of this

$$\begin{array}{c|cccccccccccc|cccccc|}
 & \multicolumn{12}{c}{\longleftarrow 2^{n} \longrightarrow} & \multicolumn{6}{c}{\longleftarrow n+1 \longrightarrow} \\
\hline
\uparrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & . & 0 & 0 & 0 & . & . & 0 & 0 & -1 \\
 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & -2 & 0 & . & 2-n & 1-n & -1/2 & . & . & -1/2 & -1/2 & 1-n/2 \\
\hdashline
 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & . & 0 & 1 & 0 & . & . & 0 & 1/2 & 1/2 \\
 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & . & 1 & 1 & 0 & . & & 1/2 & 0 & 1/2 \\
n+2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & . & 1 & 1 & 0 & & . & 0 & 0 & 1/2 \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & . & 1 & 1 & 0 & . & . & 0 & 0 & 1/2 \\
 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
\downarrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & . & 1 & 1 & 1/2 & . & . & 0 & 0 & 1/2 \\
\hline
\end{array}$$

Figure 3. Basic Tableau for B

row contains the dual variables. Since we have a minimization problem (y_o = minimum), the Simplex pivot rules require that we pivot on any positive element (not in the top row) in any primal column with a positive relative cost coefficient ($\lambda_{oj} > 0$), noting that the zero solution column (y_a in fig. (2a), except for its top element) removes any distinction between all such positive pivot elements in any pivot column (4, p. 59). Our problem is degenerate, so that we cannot use the normal Simplex convergence proof, and we must invoke the usual empirical observation that the algorithm will converge in practice, if not in theory (3, pps. 127-128). If, for any basic tableau, all the cost coefficients (top primal row) are seminegative, then by the Simplex optimality condition (3, p. 109), we have reached an optimum solution to the primal problem, and the dual portion of the cost row will constitute a feasible solution w to the dual problem, viz., a weight vector which realizes $f(\underline{x})$ in one gate. If, at any stage, we select a column with $\lambda_{oj} > 0$, such that $\lambda_{ij} \leq 0$, $i \neq 0$, then the Simplex rules require us to pivot on λ_{oj} (3, p. 108). This will clearly generate a feasible solution y_a to the primal problem (with $y_o = 0$), and the dual variables will vanish, thus implying that no feasible w exists. Finally, it is clear that each pivot operation merely moves one unit primal column vector to another primal column, thus leaving the tableau basic.

It is a property of the Simplex algorithm that the total number of iterations required to reach an optimal solution will be reduced if we select pivot columns which satisfy theorem 2, and also have the largest top element (largest relative cost).*

The above process constitutes the first existence calculation, and it will also be used for subsequent existence calculations ($s > 1$) by merely increasing the size of the tableau. Incidentally, this is a fairly simple method for determining if a given switching function can be realized with a single gate (ref. 8 presents an alternative method for comparison). It is practical for hand computation up to about $n = 4$, as mentioned above.

We define a primal column with $\lambda_{oj} > 0$, $\lambda_{ij} \leq 0$, $i \neq 0$, to be a terminal column (implying the existence of at least one basic feasible solution y), and we will normally search the non-terminal primal columns for pivots in an existence calculation. Furthermore, we will attempt to use pivot elements equal to unity, for reasons given below.

Ultimately, if all possible pivot columns are terminal, we cannot proceed further without pivoting on the top primal row. We define such a tableau to be a terminal tableau, and instead of continuing the existence calculation, we will initiate an adjoining calculation (to be described below). The attaining of a terminal tableau, if an existence calculation fails, is related to convergence of the Simplex algorithm itself, since the only alternative is cycling. In any case, we theoretically require only one terminal column to initiate an adjoining calculation; continuing

* Computational details and alternatives are discussed in Appendix II.

the existence calculation until a terminal tableau is reached merely is desirable in practice, since it simplifies the adjoining calculation (cf. theorem 8 below).

We digress in order to illustrate the above points. The two-variable basic tableau for B is:

$$\begin{array}{cccc|ccc} & & & & w_1 & w_2 & w_3 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ \cline{5-7} 1 & 0 & 0 & -1 & -1/2 & -1/2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1 & 1 & 1/2 & 0 & 1/2 \\ \hline \end{array} \tag{16}$$

For the AND function, we modify the last primal column by the rule mentioned above (theorem 5), and obtain a basic tableau for A:

$$\begin{array}{cccc|ccc} \hline 0 & 0 & 0 & 2 & 0 & 0 & -1 \\ \cline{5-7} 1 & 0 & 0 & \textcircled{1} & -1/2 & -1/2 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1 & -1 & 1/2 & 0 & 1/2 \\ \hline \end{array} \tag{17}$$

We initiate an existence calculation by taking the circled pivot, obtaining:

$$\begin{array}{cccc|ccc} & & & & w_1 & w_2 & w_3 \\ \hline -2 & 0 & 0 & 0 & 1 & 1 & -1 \\ \cline{5-7} 1 & 0 & 0 & 1 & -1/2 & -1/2 & 0 \\ 1 & 1 & 0 & 0 & -1/2 & 0 & 1/2 \\ 1 & 0 & 1 & 0 & 0 & -1/2 & 1/2 \\ \hline \end{array} \tag{18}$$

Note that the sum of all basic tableau elements in each primal column is unity, and is zero in each dual column. This is proved below (theorem 4), and is useful in checking the work.

Since row $\lambda^{o} \leq 0$ in tableau 18, we have found a feasible w; viz., $w = (1, 1, -1)$, so that the linear input to the gate is $x_1 + x_2 - 1$, which gives the two-variable AND function in (-1, +1) notation. To convert this to (0, 1) notation, leave all weights unaltered except the bias weight (w_{n+1}), which is modified by adding unity to it, subtracting the algebraic sum of the other weights, and dividing the result by two. Thus, the above weight vector is also valid for (0, 1) notation.

For an example of a terminal tableau, the first basic tableau obtained for the two variable exclusive OR function (by the process above) is terminal:

$$\begin{array}{|cccc|ccc|}
\multicolumn{4}{c}{} & \multicolumn{1}{c}{w_1} & \multicolumn{1}{c}{w_2} & \multicolumn{1}{c}{w_3} \\ \hline
0 & 0 & 0 & 4 & 1 & 1 & 1 \\ \cline{5-7}
1 & 0 & 0 & -1 & -1/2 & -1/2 & 0 \\
0 & 1 & 0 & -1 & 0 & -1/2 & -1/2 \\
0 & 0 & 1 & -1 & -1/2 & 0 & -1/2 \\ \hline
\end{array} \qquad (19)$$

In obtaining tableau 19, two modified unit vectors were restored to their former positions by one pivot operation each (on the -1 element in each modified unit vector).

IV. TABLEAU RELATIONSHIPS

If the existence calculation for g^s $(s \geq 1)$ fails, we must find and adjoin a proper gate g^s to the tableau. Before discussing this, it is necessary to develop some relationships among the tableau elements of fig. (2). We reemphasize that the top tableau row is never used as a pivot row.

First we prove:

Theorem 3:

For any basic tableau, $\lambda^o - e = \gamma^o A$ and $\lambda^i = \gamma^i A$, $i \neq 0$, where λ^i, γ^i are the rows of the primal and dual tableaus, respectively, A is the logic state matrix, and e is the unit vector of length m.

proof: We know that the Γ matrix is the inverse of the current primal basis (as identified by the unit vectors in Λ), viz:

$$\left[\begin{array}{c|c} 1 & \\ 0 & \\ 0 & \\ \cdot & \Lambda \\ \cdot & \\ \cdot & \\ 0 & \end{array}\right] = \left[\begin{array}{c|c} 1 & \\ 0 & \\ 0 & \\ \cdot & \Gamma \\ \cdot & \\ \cdot & \\ 0 & \end{array}\right] \left[\begin{array}{c|cccccc} 1 & 1 & 1 & \cdot & \cdot & \cdot & 1 \\ \cline{2-7} 0 & & & & & & \\ 0 & & & & & & \\ \cdot & & & A & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ 0 & & & & & & \end{array}\right] \tag{20}$$

We have used the complete tableau of fig. (2a) for this; a discussion of the inverse basis may be found in Gass (4, p. 27, p. 75).

Expressing equation 20 by rows, we obtain

$$\begin{aligned} \lambda^o &= \gamma^o A + e, \\ \lambda^i &= \gamma^i A, \ i \neq 0, \end{aligned} \qquad (21)$$

which give the desired relations.

Note that the first of equations 21 implies that subtracting unity from each element of the top row of the primal working tableau will give the linear input (for each state $\underline{x}$) for the threshold gate whose weight vector is given by the top row of the dual tableau, in modified form where the actual input has been multiplied by (-1) for the true states of $f(\underline{x})$. We shall define the gate function given by γ^o $(g(\underline{x}) = \text{sgn}\,(\lambda^o - e))$ as the tableau gate, even though $g(\underline{x})$ has been modified as described above.

Using theorem 3, we can prove another useful result:

Theorem 4:

For any basic tableau, the sum of all elements in each column is unity for the primal matrix, and zero for the dual matrix.

proof: From theorem 3, $\lambda^i = \gamma^i A$, $i \neq 0$. Consequently, if we add all other rows to the top row, obtaining a new top row $\underline{\lambda}^o$, $\underline{\gamma}^o$, then $\underline{\lambda}^o - e = \underline{\gamma}^o A$. Now, there are $n + 1$ unit column vectors λ^j, $j \epsilon E$, in Λ, where we define the column index set E such that λ^j is a unit vector for $j \epsilon E$. Hence, $\underline{\lambda_{oj}} = 1$, $j \epsilon E$. Furthermore, these $n + 1$ columns must correspond to $n+1$ independent columns of A, by the pivoting process. Let these columns of A be grouped into a square matrix Am. Then the tableau weight vector $\underline{\gamma}^o$ satisfies $\underline{\gamma}^o$Am = 0, since the components of the linear sum vector on the right are precisely $\underline{\lambda_{oj}} - 1 = 0$, $j\epsilon E$, by theorem 3. But Am is nonsingular, and hence $\underline{\gamma}^o = 0$ is the only solution. Therefore $\underline{\gamma}^o A = 0$ and thus the primal top row must be $\underline{\lambda}^o = e$, again by theorem 3. Consequently, by the way that $(\underline{\lambda}^o, \underline{\gamma}^o)$ was constructed, and for all i, j,

$$\sum_{i=0}^{n+1} \lambda_{ij} = \underline{\lambda_{oj}} = 1, \qquad (22)$$

and

$$\sum_{i=0}^{n+1} \gamma_{ij} = \underline{\gamma_{oj}} = 0, \qquad (23)$$

which is the statement of the theorem.

This result allows us to take a basic tableau for B (or for some A) with the same number of variables, and easily modify it so that it is a basic tableau for a given A matrix:

Theorem 5

Given a basic tableau for the B matrix of n variables, a basic tableau for any A matrix of n variables may be obtained by modifying each column λ^j, $j\epsilon T$, corresponding to a true column of A, by the following reversal rule: replace λ_{oj} by $2 - \lambda_{oj}$, and λ_{ij} by $-\lambda_{ij}$, $i \neq 0$, for each $j\epsilon T$, and perform any pivot operations required to restore the previous primal unit vectors.

proof: The state matrix A is obtained by adjoining the unit row vector to B and reversing the sign of each column which corresponds to a true state of the switching function. Since we never pivot on the top tableau row, this is precisely equivalent to reversing the signs of the corresponding elements λ_{ij}, $i \neq 0$, in any tableau obtained from A. This is valid because pivot multiples are functions only of the ratios of elements in the pivot column, and hence are invariant to the reversal of the signs of all elements in a column. Some pivots may have changed sign, but this is allowable in our case. However, we do not reverse the sign of any element in the top row of $\underline{A}$ (equation 10), and consequently we must find the new top row by other means. If we first assume that the unit vectors in the basic tableau for B do not correspond to true states of $f(\underline{x})$, then the above sign reversals leave the tableau in basic form, so that the proof of theorem 4 remains valid. Thus, for the primal columns λ^j, $j \epsilon F$, $\lambda_{oj}(A) = \lambda_{oj}(B)$, and for λ^j, $j \epsilon T$, $\lambda_{oj}(A) = 1 - \sum_{i \neq 0} \lambda_{ij}(A) = 1 + \sum_{i \neq 0} \lambda_{ij}(B) = 2 - \lambda_{oj}(B)$. Finally, since $\gamma^i(A) = \gamma^i(B)$, $i \neq 0$, theorem 4 implies that $\gamma^o(A) = \gamma^o(B)$.

The proof of theorem 4 remains valid with minor modifications even if some unit vectors correspond to true states in applying the present theorem, and consequently we need only pivot on these modified unit vectors to restore the tableau to basic form (with the same basis).

Clearly, we can use a basic tableau for a different A matrix in using theorem 5 by applying the reversal rule to the columns corresponding to the conflicting logic states of the two switching functions (both functions must have the same number of variables). Tableau 17 is an example of this theorem (page 24).

Theorem 5 can be easily extended to any s for the multiple gate case.

We will discuss below the selection of pivot rules which (after an unsuccessful existence calculation) will generate the gate to be adjoined (to the net being computed) as the tableau gate. In order to adjoin this gate to the tableau, we need:

Theorem 6:

The tableau gate g^s is adjoined to its basic tableau in proper form to be pivoted into the basis by the following rule: the primal portion of the new bottom row is given by $g_j^s = 2 - \lambda_{oj}$, $j \epsilon P \triangleq \left\{ j \,|\, \lambda_{oj} > 0 \right\}$, and $g_j^s = -\lambda_{oj}$, $j \epsilon N + Z \triangleq \left\{ j \,|\, \lambda_{oj} \leq 0 \right\}$. The dual portion of the row is $g_j^s = -\gamma_{oj}$. A unit column vector is adjoined on the right of the dual tableau to augment the inverse basis, with the unit element being in the new bottom row.

Before proving this theorem, we digress to more fully clarify its meaning. First, we define the index sets N, P, and Z as those sets of column indices j such that, for a given basic tableau, $\lambda_{oj} < 0$, $\lambda_{oj} > 0$ and $\lambda_{oj} = 0$, respectively. Note that we have used a plus sign for the

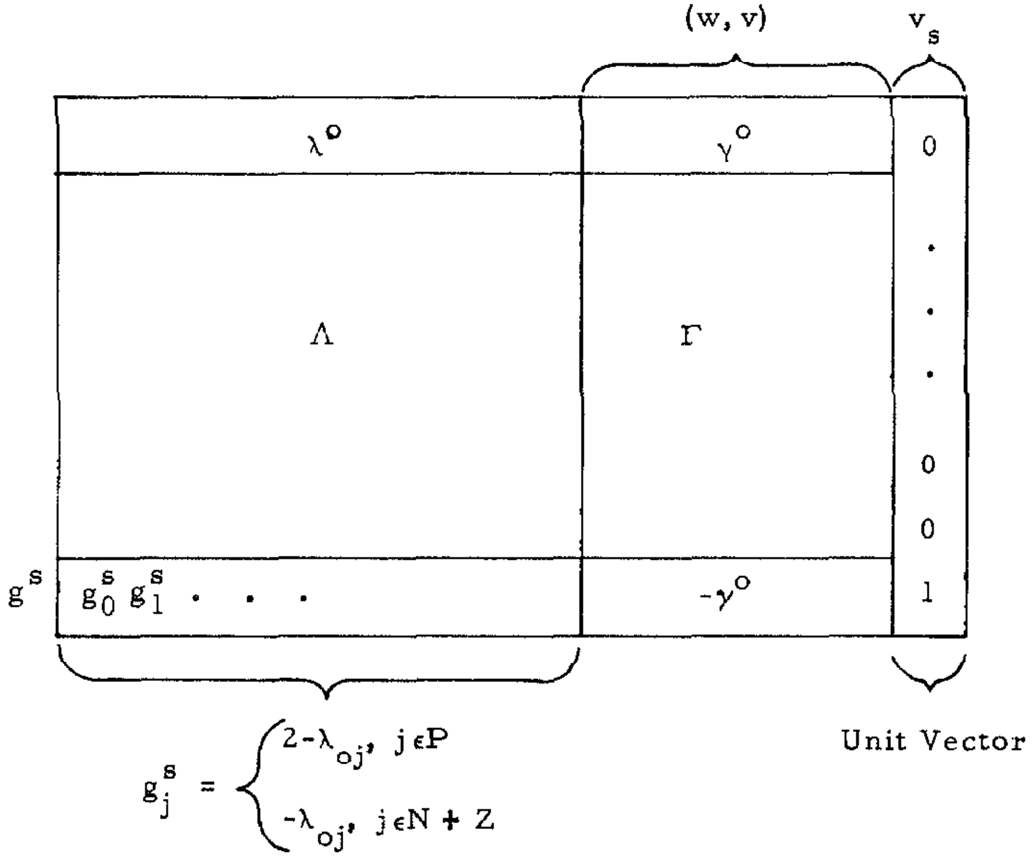

(w, v)
v_s
λ^o
γ^o
0
Λ
Γ
0
0
g^s
$g_0^s\ g_1^s\ \cdot\ \cdot\ \cdot$
$-\gamma^o$
1
$g_j^s = \begin{cases} 2-\lambda_{oj},\ j\epsilon P \\ -\lambda_{oj},\ j\epsilon N + Z \end{cases}$
Unit Vector

Figure 4. Adjoining Tableau Format

set-theoretic union operation, since no confusion will arise here. Also note that Z contains E as a subset.

Application of theorem 6 gives the adjoined tableau shown in fig. (4), where the upper right hand corner element will contain the output weights of g^s to the subsequent higher-level gates, and (w, v) is presently (w^s, v^s). For example, applying the theorem to tableau 19 gives:

$$\begin{array}{c|cccc|ccc|c|}
 & & & & & w_1^1 & w_2^1 & w_3^1 & v_1 \\ \hline
 & 0 & 0 & 0 & 4 & 1 & 1 & 1 & 0 \\ \cline{6-8}
 & 1 & 0 & 0 & -1 & -1/2 & -1/2 & 0 & 0 \\
 & 0 & 1 & 0 & -1 & 0 & -1/2 & -1/2 & 0 \\
 & 0 & 0 & 1 & -1 & -1/2 & 0 & -1/2 & 0 \\ \hline
g' & 0 & 0 & 0 & -2 & -1 & -1 & -1 & 1 \\ \hline
\end{array} \qquad (24)$$

After we adjoin the gate to the tableau, we must select some nonzero pivot element in the primal portion of the new row (-2 in the above example) in order to pivot out the new dual unit vector, generating one more unit vector in the primal, and thus restoring the tableau to basic form.

<u>proof of theorem 6</u>: If we had adjoined g^1 to $\underline{A}$ at the beginning of the computation and had obtained the first basic tableau by the longer procedure (bypassing the use of the B tableau), we would have merely written the modified gate function $g^1(\underline{x}) = \pm 1$ as an additional primal tableau row, and we would have augmented the dual unit matrix with a bottom row of zeros and with the new unit vector shown in fig. (4). In fact, if we wished to adjoin rows $g^1, g^2, \cdots, g^s$ to $\underline{A}$ and repeat the entire

computation, we could treat them as rows of A, as long as the rank of A was then $n + s + 1$, since the above theorems do not distinguish between rows of $\underline{A}$ arising from direct logic variables and those arising from gate functions, except for theorem 1. We replace theorem 1 (for $s > 1$) by the requirement that the rank of the adjoined tableau be $n + s + 1$, and we show below that this requirement is a necessary condition for net minimality. Consequently, with this restriction, the above theorems are valid for any $s \geq 1$.

Hence, if we so adjoin g^s at the beginning, and repeat precisely the pivot operations that led to the adjoining tableau, the g^s row would be in the desired adjoining form, and we need only prove that the above rule gives this row form directly, without actually repeating the computations. Note that row g^s is not pivoted upon in this conceptual process, as is consistent with the fact that its unit vector is still in the dual tableau in fig. (4).

Now, the primal portion of the g^s row we seek is therefore clearly equal to the modified gate function $g^s(\underline{x}) = \pm 1$, plus some linear combination of all other rows (except the top rows) of all previous tableaus, which were chosen (after obtaining the first basic tableau) such that $g_j^s = 0$, $j \epsilon E(q)$ at each stage q. By the nature of the pivot operation, the rank of the primal tableau does not change, and only a nonsingular transformation has been made on the equations $Ay = 0$, $ey = 1$, such that they have been replaced by an equivalent set $\Lambda y = 0$, $\lambda^o y = 1$ (this fact will be useful below).

Consequently, the primal rows of the adjoining tableau have the same rank as all previous primal tableaus, and also span the same subspace. Thus, if there exists a unique linear combination of the primal adjoining tableau rows which restores the primal unit vectors when added to $g^s(\underline{x}) = \pm 1$, then this must give the proper g^s row to be adjoined to the tableau (including the dual portion). For this tableau, theorem 3 implies that $g^s(\underline{x}^j) = -1$, $j \epsilon E$. We seek a set of scalars α_i such that $g^s(\underline{x}^j) + \sum_{i \neq 0} \alpha_i \lambda_{ij} = 0$, $j \epsilon E$. Obviously, the solution is that all α_i equal unity, and it is clearly unique. Applying theorem 4 (omitting the g^s row from the sums), we see that the result of adding all primal and dual rows (except the top row) to the vector $(g^s, 0)$, gives the following relative g^s row:

Primal:

$$\text{For } j \epsilon P:^{*} \quad g_j^s = (+1) + \sum_{i \neq 0} \lambda_{ij} = 2 - \lambda_{oj},$$

$$\text{For } j \epsilon N + Z: \quad g_j^s = (-1) + \sum_{i \neq 0} \lambda_{ij} = -\lambda_{oj}, \tag{25}$$

Dual:

$$g_j^s = (0) + \sum_{i \neq 0} \gamma_{ij} = -\gamma_{oj},$$

which is the statement of the theorem. Note that the new dual unit column vector in fig. (4) remains unaltered by this process, as required.

* allowability of the tableau weight vector implies that $\lambda_{oj} \geq 2$, $j \epsilon P$.

Aside from other conditions on the tableau gate to be adjoined, we must also insure that its weight vector is allowable. From theorem 3, this is equivalent to requiring that $\lambda_{oj} \geq 2$, $j \epsilon P$, for our threshold tolerance band of (-1) to (+1). We now show that the pivot constraint $\lambda_{ij} = \pm 1$, for <u>all</u> phases of the computation, is sufficient for this:

<u>Theorem 7</u>:

The tableau gate weight vector will always be <u>integral</u> and <u>allowable</u> if the primal basic tableau pivot constraint $\lambda_{ij} = \pm 1$ is always obeyed in all pivot operations. If, further, the adjoining tableau has at least one $\lambda_{oj} = -2$ and/or $\lambda_{oj} = 4$, the tableau gate can be adjoined and the tableau restored to basic form without invalidating the above statement.

<u>proof</u>: We will prove that if any tableau exists for a given A such that Λ, 2Γ, and γ^o are integral, and λ^o is even-integral, then the pivot constraint above will maintain these tableau characteristics. Clearly the first basic tableau obtained by using fig. (3), as modified by theorem 5, does possess these characteristics.

Now, if we pivot on $\lambda_{ij} = \pm 1$, then 2Γ and Λ will remain integral since all pivot multiples are integral. Furthermore, the pivot multiple for the top row must be even-integral, and hence λ^o must remain even-integral, and γ^o must remain integral. If λ^o is even-integral, then $0 < \lambda_{oj} < 2$ is impossible, and hence the tableau weight vector must be allowable. Given this, theorem 6 implies that the adjoined gate row will be even-integral in the primal, and integral in the dual portion of the adjoined tableau. Thus, we can pivot on any $g_j^s = \pm 2$ in the primal to restore the basic character to the tableau, without destroying any of the above tableau characteristics. Since these pivots arise from $\lambda_{oj} = -2$ or $\lambda_{oj} = 4$ in the adjoining tableau, this completes the proof of the theorem.

It has not been possible to find conditions such that pivot elements with absolute value unity exist in the primal tableau, and which also satisfy the other pivot requirements.* In fact, a few cases have occurred in actual existence calculations where none of the pivot columns satisfying theorem 2 had a unity element. In these cases, it has proven sufficient to take one (-1) pivot to continue the existence calculation, preferably in a column with a negative top element (this stems from a reversal of the Simplex selection rules for negative pivots).

Conversely, theorem 7 implies that there is a relationship, in this algorithm, between allowable weight vectors and integral weight vectors. In practice, violating this pivot constraint nearly always leads to unallowable weight vectors.

Finally, this pivot constraint serves to significantly ease the search for suitable pivots in the adjoining calculation to be described below.

For these reasons, we will terminate a computational phase rather than select a suitable pivot which does not also have an absolute magnitude of unity.

* These "non-integral" cases are treated in Appendix II.

We close this section by giving another interpretation of the rule $\lambda_{oj} \geq 2$, $j \epsilon P$, for tableau weight vector allowability. If we apply the sense-reversal rule of theorem 5 to the P columns of a tableau satisfying this constraint, then these elements become $2 - \lambda_{oj} \leq 0$, and thus the entire top primal row is non positive. Theorem 2 then implies that γ^o is a feasible weight vector for this altered logic function, and consequently it must be allowable.

V. ADJOINING CALCULATION

There remains the problem of finding pivot rules which (after an unsuccessful existence calculation for g^s) will generate a tableau gate having the property that a near minimal total number of gates will be required to realize $f(\underline{x})$.

The computational criterion that will be adopted is to remove the maximum number of basic feasible solutions (bfs)y that is possible for each gate adjoined; i.e., the adjoined gate gives us a new constraint on y of the form $g^s y = 0$, and we wish to choose g^s such that $g^s y \neq 0$ for a maximum number of bfs y in the adjoining tableau. This is clearly a necessary condition for the two-gate case, at least, since the existence of any feasible y implies the existence of a bfs y (3, thm 2.11, p.50).

The following theorem clarifies this, and illustrates a further use of a terminal tableau:

Theorem 8:

Each terminal column λ^j, $j \epsilon P$, in an adjoining tableau, implies the existence of a basic feasible solution y which is removed by adjoining the tableau gate if and only if $\lambda_{oj} > 2$. Furthermore, the removal of at least one such solution is a sufficient condition that the tableau rank increase by unity if the tableau gate is adjoined, and is also a sufficient condition that the algorithm converge to a finite net which realizes $f(\underline{x})$.

proof: There are clearly a finite number of distinct bases in A, so that the last statement of the theorem follows from the preceding discussion. For the first part of the theorem, we shall define a unit vector solution y to be a bfs containing the tableau unit vectors and one terminal P column λ^j. The components of this vector y are zero, except that $y_j = 1$, $y_k = -\lambda_{ij} \geq 0$, where λ^k is the (i + 1)st unit vector $(\lambda^k = e^{i+1})$. It can easily be seen that this vector satisfies $\Lambda y = 0$, $y \geq 0$, and can be normalized to $ey = 1$.* This solution is equivalent to the bfs y obtained by pivoting on λ_{oj} (in the normal Simplex process).

Now, if we adjoin the tableau gate g^s in relative form, $g^s_k = 0$, $k \epsilon E$, and $g^s_j = 2 - \lambda_{oj}$. Thus, if $\lambda_{oj} > 2$, the unit vector solution associated with λ^j is removed by adjoining g^s, and if $\lambda_{oj} = 2$, then $g^s_j = 0$, so that the associated unit vector solution is not removed by g^s.

If any feasible solution y is removed by g^s, then it is clear that row g^s cannot be linearly dependent on the rows of Λ, because $g^s y = 0$ would then follow, contrary to the requirement that $g^s y \neq 0$. Thus the tableau rank will increase by unity for each gate adjoined, as is required for the subsequent existence calculations, if at least one bfs y is removed.

Note that a top primal row element equaling two is impossible in an integral terminal tableau, since there must be then exactly one other element equal to (-1) with the rest zero (by theorem 4). Applying theorem 5 to this column turns it into a unit vector, implying that two primal columns are dependent, and thus that two columns of A are dependent. This is not possible, due to the presence of the bias vector in A. Consequently, adjoining the tableau gate to an integral, terminal tableau removes all exhibited bfs y.

* e.g., for tableau 19 (page 25), y = (1, 1, 1, 1).

Theorem 8 does not tell us whether possible bfs y containing N vectors are removed by g^s, and furthermore, the requirement of tableau terminality for the adjoining calculation is cumbersome in practice even though we begin an adjoining calculation with a terminal tableau reached by the preceding existence calculation. The insufficiency of theorem 8, in general, is due to the fact that we are requiring the algorithm to be iterative, in the sense that each tableau is completely determined by the one immediately preceding it. For example, if we search for a terminal tableau with a maximal number of P columns whose top elements are greater than two, then adjoining the tableau gate removes the maximum number of bfs y which can be exhibited by a single iteration on the adjoining tableau (viz., by pivoting on the top primal row). The theorem gives us a theoretically necessary method for finding a two-gate net for $f(\underline{x})$ (if one exists), in that we can conceptually search the tableau by an exhaustive sequence of iterations to find all bfs y, and all allowable weight vectors, and then compare these two sets in order to determine whether any of the gates removes all bfs. This process is obviously a computationally poor one, and we shall instead require that the adjoined gate remove a maximal number of possibly feasible bases.

If we define /N/ to be the number of indices in the set N, we have

Theorem 9:

The adjoining tableau requirement that /N/ be minimal, and that $\lambda_{oj} > 2$, $j \epsilon P$, is such that adjoining the tableau gate removes a maximal number of possibly feasible bases.

proof: If we adjoin such a gate g^s by theorem 6, and then pivot on a primal element $g^s_j < 0$, chosen so that $-\lambda_{oj}/g^s_j$ is a maximum, then the λ^o row is altered to a form where $\lambda_{oj} > 0$, $j \epsilon N$; $\lambda_{oj} \leq 0$, $j \epsilon P + Z$. Since the λ^o row expresses the constraint $\lambda^o y = 1$, and since $y \geq 0$, there are consequently no feasible bases containing vectors from P + Z columns only. This set has a maximal number of vectors, and therefore a maximal number of possibly feasible bases have been removed, including any that may be represented by terminal P columns.

This minimal /N/ criterion can be proved to give minimal two-gate nets for a certain class of switching functions:

Theorem 10:

If an adjoining tableau exists for g^1 with /N/ = 0, and $\lambda_{oj} > 2$, $j \epsilon P$, then a two-gate net realizing $f(\underline{x})$ can be found in one more iteration.

proof: Having such a tableau, adjoin the tableau gate g^1 and pivot it into the basis in the manner used in the proof of theorem 9. Then the entire top primal row becomes non-positive, and thus, by theorem 2, the new tableau gate realizes $f(\underline{x})$.

Examination of the above pivot operation reveals that this is the class of two-gate nets with parallel direct input weight vectors w^1, w^2 (including the bias weight). The existence of such realizations seems to be neither rare nor common, in practice, for two-gate switching functions, particularly if we note that theorem 10 also applies to the last two gates in any net. Tableau 24 (p. 32) is an example of a /N/ = 0 case.

We may give another interpretation of the minimal /N/ criterion by considering the system of inequalities 1 that we were attemping to satisfy in the existence calculation for g^1. Since it was proved impossible to satisfy $w^1a^k \leq -1$ for all k, it might be felt that the requirements on g^2; viz., $w^2a^k + v_1^2 g_k^1 \leq -1$, would be relaxed if the weight vector w^1 was such that $w^1a^k \leq -1$ for a maximal number of indices k. The minimal /N/ criterion indicates that it is equally advantageous to attempt to satisfy either $w^1a^k < -1$ or $w^1a^k = +1$. This allows somewhat greater freedom to g^1 in relaxing the requirements on g^2. The information used to obtain this improvement is that obtainable in one iteration after adjoining a gate; i. e., in the single iteration used to restore the tableau to basic form. Note that this is again consistent with our first-order iterative procedure.

Before discussing pivot rules for the adjoining calculation, we shall introduce a flagging process by which we can discover (and flag) certain primal columns which cannot be contained in any bfs y. This arises from the following simple observation: as mentioned above, the primal rows λ^i, $i \neq 0$, represent the equations $\lambda^i y = 0$. Since y must be semipositive, it is clear that if any row λ^i becomes semipositive, or seminegative, in any tableau, then all y_j components corresponding to non-zero elements in that row must be identically zero; i. e., cannot appear in any bfs. Thus we can flag the primal columns λ^j corresponding to these $y_j = 0$, and ignore them in counting /N/ in all tableaus.* We

* This is the only cumulative (non-iterative) part of the algorithm.

also ignore previously flagged columns in inspecting the primal rows for subsequent column flagging. Consequently, theorems 9 and 10 may be improved by this process.

This concept can be carried further. If any semipositive or negative row can be found in any tableau at stage s, then it can be found by some linear combination of the primal rows λ^i, $i \neq 0$, in any single tableau of stage s. The determination of such a linear combination is a linear programming problem which could be added to the computation if desired, (say), as part of the adjoining calculation (the method consists of changing some of the components of the e vector in equation 10 to zero). However, in practice, it turns out that such possible seminegative or positive rows are usually generated automatically by the previous existence calculations; e. g., as a by-product of attempting to attain a seminegative (optimal) top primal row, semipositive primal rows are generated if possible. A theoretical proof of this seems impossible without modifying the algorithm (e.g., by dropping the integral requirement), but we can interpret the meaning of a terminal tableau in the present context. Assuming that no columns have been previously flagged, let us attempt to construct a semipositive linear combination of the terminal tableau primal rows (omitting the top row). The presence of the unit column vectors in the primal clearly implies that the row multiples of such a linear combination must be semipositive. The presence of the terminal P columns then requires that the row multiple be zero for any row containing one or more $\lambda_{ij} < 0$, $j \epsilon P$. Thus, if we define a span-terminal tableau to be a terminal tableau with at least one $\lambda_{ij} < 0$, for every $i \neq 0$, and where $j \epsilon P$, then we have shown that a span-terminal tableau, which is unflagged, cannot be flagged, in that no semipositive row can be formed from it (and, similarly, no seminegative row). In practice, all unflagged tableaus have been unflaggable, usually because the terminal tableaus are span-terminal (except for rare cases constituting less than 1% of the total number of cases tried).

Let us close this subject by noting the symmetry between the existence, adjoining, and flagging calculations. For an existence calculation we seek a seminegative top row, for an adjoining calculation we seek a semipositive top row (with, however, all positive elements greater than two), and for flagging we seek either semipositive or seminegative primal rows (excepting the top row).

We must now discuss pivot rules for an adjoining calculation. In view of the above, we first consider reversing the existence calculation pivot rules; viz., pivot on $\lambda_{ij} = 1$, such that $\lambda_{oj} < 0$. Just as the existence calculation pivot rules will give us a seminegative top row, if one exists, these rules will give us a semipositive top row, if one exists. However, we must also avoid generating twos in the top row, both to satisfy theorem 9, and to avoid generating unadjoinable tableau gates where the top primal row consists of twos and zeros only. The first tableau for A, as obtained from the basic tableau for B by theorem 5, is such a "null" gate, and these null gates are characterized by the fact that they are not actually threshold gates at all. Namely, applying the sgn operator to the linear inputs of null gates has no effect; i.e., no nonlinear operation is performed. In order not to unduly complicate the algorithm, we shall avoid such null gates by not allowing any pivot operation to be made which generates twos in any unflagged elements of the top row.

Thus, a suitable pivot has $\lambda_{ij} = 1$, $\lambda_{oj} < 0$, and generates no unflagged twos. As in the existence calculation, we reduce the number of iterations required by selecting suitable pivots with minimal values of λ_{oj}^{*}. We terminate the adjoining calculation either if there are no unflagged columns in N, at some point, or if there are no suitable pivots. The tableau gate is then adjoined by theorem 6, pivoted into the basis, and an existence calculation undertaken.

Another possible set of adjoining calculation pivot rules would be the following: select a $\lambda_{ij} = \pm 1$ pivot which effects a minimal (algebraic) change in /N/ (i.e., negative if possible), and does not generate twos in the top primal row. Again, flagged columns are ignored, and the computation terminates as before. This is a more complex method than the former one, and moreover, gives poorer results in practice (more gates per net). Therefore, it will not be further developed here.

Several different adjoining calculation pivot rules have been tried, the main value of which is to give one a choice of different threshold nets for the required switching function. Of these, one should be briefly mentioned, since it is an obvious extension of the flagging process developed above. Specifically, one could attempt to find a gate which would allow a maximal number of primal columns to be subsequently flagged, instead of seeking a gate which removes a maximal number of bfs. In view of the flagging discussion above, it would be particularly advantageous to try to flag columns which have appeared in span-terminal sets of columns; viz., sets of terminal columns which cause a tableau to be

*Appendix II contains another iteration reduction method.

span-terminal. A complex procedure was developed and programmed for this method, but the resultant nets were quite unsatisfactory (non-minimal).

The best set of algorithm rules found are summarized in Appendix II, together with some programming suggestions.

Now, these algorithm rules may also be used for the symmetric case by using the B matrix given by equation 8, and making two pivot operations to obtain a first basic tableau. The existence and adjoining calculations can proceed as before with this smaller tableau. However, for this symmetric net case, we can prove the theoretical necessity of a different adjoining tableau criterion in obtaining minimal two-gate symmetric nets:

Theorem 11

For the symmetric net algorithm, a necessary condition that a two gate net be found for a given $f(\underline{x})$ is that the adjoining tableau for g^1 satisfy the following conditions: for all triads of primal indices $j < k < l$, such that $j \epsilon T$ (F), $k \epsilon F$ (T), $l \epsilon T$ (F), either $g^1 (\underline{x}^j) = g^1(\underline{x}^k)$ and/or $g^1 (\underline{x}^k) = g^1(\underline{x}^l)$.

We digress first to clarify this. The primal tableau for g^1 contains three rows, and we tentatively adjoin the (modified) tableau gate g^1 as a fourth row by setting $g^1 = \text{sgn}\ (\lambda^o - e) = \pm 1$; i.e., $g^1(\underline{x}^j)$ is the j^{th} primal component of this new row. The rule requires that we examine each ordered set of three primal columns where the first and last have the same logic state sense (corresponding to $f(\underline{x})$), and the middle has the opposite. If, for any such triad, $g^1(\underline{x})$ also has an alternating sense ($\pm 1, \mp 1, \pm 1$), then a two-gate net will not be found using this g^1.

<u>proof of theorem 11</u>:

Let us investigate basic solutions y (not necessarily semipositive) to $By = 0$, $ey = 0$, where B is given by equation 8, viz., the $j^{\underline{th}}$ element b_j is $-n+2j$, $j = 0, 1, 2, \cdots, n$. It is easily seen that all sets of less than three vectors $(b_j, 1)$ are independent. All sets of three such vectors are dependent, giving basic solutions y with nonzero components

$$\begin{aligned} y_j &= 1 - k > 0 \quad , \\ y_k &= j - 1 < 0 \quad , \qquad (26) \\ y_1 &= k - j > 0 \quad , \end{aligned}$$

where, without loss of generality, we take $j < k < l$. Clearly, relations 26 will imply the existence of a bfs $y \geq 0$ using any triad corresponding to a triad of alternating logic states as described in the theorem statement, and which can be normalized to $ey = 1$. Furthermore, corresponding $g^1(\underline{x})$ sequences (1, 1, 1) or (-1, -1, -1) will remove these solutions, so that the only interesting cases are sequences such as (-1, -1, 1), (-1, 1, 1), (-1, 1, -1) and their negatives. Assume, without loss of generality, that $j, l \epsilon F$, $k \epsilon T$. Then we obtain a normalizable bfs $y \geq 0$ such that $y_j = 1-k$, $y_k = 1-j$, $y_1 = k-j$, where we have reversed y_k because its corresponding column in A is reversed. Now compute g^1y for the above three sequences:

$$\begin{aligned} -y_j - y_k + y_1 &= 2\ (k - 1) \neq 0 \\ -y_j + y_k + y_1 &= 2\ (k - j) \neq 0 \\ -y_j + y_k - y_1 &= 0 \end{aligned} \tag{27}$$

Hence, the only possibility for non-removal of this bfs y is that these components of $g^1(\underline{x})$ duplicate the bias vector constraint in $Ay = 0$; i.e., no new constraint is added to these bfs. This fact is necessary for the general algorithm, but it is not sufficient to insure removal of a bfs. Consequently, the allowable $g^1(\underline{x})$ sequences for removal of all bfs y are those stated in the theorem. If all bfs y are not removed, we know that the next existence calculation will not succeed and a two-gate net will not be found for $f(\underline{x})$. Hence, the theorem is a necessary criterion.

Theorem 11 can be used in an adjoining calculation as an addition to the previous rules (as with theorem 10), or it can form a basis for selection of pivot rules (as with theorem 9), since it gives us a direct means of computing all unremoved bfs in a g^1 symmetric net tableau without pivoting. However, no reasonably simple such set of adjoining calculation rules have been found which, in practice, gave more nearly minimal nets than the previous rules. Consequently, theorem 11 seems to be best used in conjunction with the previous rules in the manner of theorem 10 (i. e., for a criterion as to whether to adjoin the present tableau gate).

Concerning the general question of adjoining calculation pivot rules, we must consider cycling; viz., a closed, repeating sequence of tableau gates. The rules chosen (Appendix II) are Simplex-derived, and the restriction against generating unflagged twos in the top primal row serves to further decrease the probability of cycling. In practice, cycling has never occurred in the general algorithm, but it is sometimes possible in the symmetric net algorithm. This occurred in the adjoining calculation for the first few gates in symmetric nets containing a relatively large number of gates, where several different sets of lower-level gates all lead to the same total number of net gates; e.g., nets for the symmetric parity functions. The probability of cycling decreases as gates are adjoined to the tableau. In practice, the following additional adjoining calculation rules prevent adjoining calculation cycling: before making a pivot operation, mark the unity element of the primal unit vector leaving the basis (being destroyed) as a non-allowable pivot for the next iteration, and allow no more than four times the number of tableau rows as the maximum number of iterations which do not decrease /N/, in each adjoining calculation.

We now give two multiple-gate sample calculations in order to clarify all the algorithm rules. First, let us realize the three-variable switching function $f(\underline{x}) = \underline{x}_1 \, \underline{x}_2 + \underline{x}_2 \, x_3 + x_1 \, x_2 \, \underline{x}_3$ (in Boolean AND/OR notation), using the general algorithm. Using fig. (3), applying theorem 5 to the "true" primal columns (0, 1, 5, 6), and restoring two unit vectors, we obtain:

$$\begin{array}{cccccccc|cccc}
t & t & & & & t & t & & & & & \\
\hline
0 & 0 & 0 & 0 & 0 & 2 & 4 & -2 & -1 & -1 & 0 & -1 \\
1 & 0 & 0 & 1 & 0 & -1 & -1 & 2 & 1/2 & 1/2 & 1/2 & 1/2 \\
0 & 1 & 0 & -1 & 0 & \textcircled{1} & 0 & -1 & 0 & 0 & -1/2 & -1/2 \\
0 & 0 & 1 & 1 & 0 & 0 & -1 & 1 & 0 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 1/2 & 0 & 0 & 1/2 \\
\hline
\end{array} \tag{28}$$

We initiate an existence calculation, beginning with the circled pivot in tableau 28 (even though there already is a terminal column which indicates that no one gate solution exists), using theorem 4 to check the work.

After two iterations, we reach a terminal tableau which has a span terminal column and is therefore unflaggable:

$$\begin{array}{c|cccccccc|cccc|c}
 & & & & & & & & & w_1^1 & w_2^1 & w_3^1 & w_4^1 & v_1^s \\
\hline
 & 0 & -2 & -2 & 0 & 0 & 0 & 6 & -2 & -1 & -2 & 1 & -1 & 0 \\
 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 1/2 & 1/2 & 0 & 0 & 0 \\
 & 0 & 1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & 1/2 & -1/2 & 0 & 0 \\
 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & 1 & 0 & 1/2 & 0 & 1/2 & 0 \\
 & 0 & 1 & 1 & 0 & 1 & 0 & -2 & 1 & 1/2 & 1/2 & -1/2 & 1/2 & 0 \\
\hline
g^1 & 0 & \textcircled{2} & 2 & 0 & 0 & 0 & -4 & 2 & 1 & 2 & -1 & 1 & 1 \\
\hline
\end{array} \tag{29}$$

We have adjoined the tableau gate by theorem 6 because there are no suitable pivots in tableau 29 for the adjoining calculation; viz., there is no unity primal pivot element in any column with a negative top element which does not generate twos in the top primal row.

This adjoined gate is now pivoted into the basis using the circled pivot element in tableau 29, and another existence calculation undertaken as before. After four iterations (including the above one), we reach an optimal tableau (top primal row seminegative), and hence have finished:

$$\begin{array}{cccccccc|ccccc}
 & & & & & & & & w_1^2 & w_2^2 & w_3^2 & w_4^2 & v_1^2 \\
\hline
-2 & 0 & 0 & -2 & 0 & -2 & 0 & 0 & 1 & 1 & -1 & 1 & 3 \\
\cline{9-13}
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & -1/2 & -1/2 & 1/2 & -1/2 & -1 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & -1/2 & 0 & 0 & 0 & -1/2 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1/2 & 0 & -1/2 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1/2 & 0 & 0 & -1/2 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1/2 & -1/2 \\
\hline
\end{array} \tag{30}$$

Note also that all the primal columns in tableau 30 are flaggable. This minimal two-gate net, in the notation used by Minnick (5) is

$$x_3 * x_1,\ x_2,\ 3\,(x_1,\ 2x_2,\ 1 * x_3),\ 1 \quad , \tag{31}$$

where the asterisk represents a gate with zero threshold whose input is obtained by summing the indicated terms, with minus signs attached to terms on the left of the asterisk.

For a symmetric-net example that we can apply theorem 11 to, we choose the five variable symmetric switching function which is true for either zero, two, or three input variables true.

First, we write down the non-basic tableau for the $\underline{A}$ matrix obtained by using the B matrix of equation 8:

t		t	t			w_x^1	w_b^1
1	1	1	1	1	1	0	0
5	-3	①	-1	3	5	1	0
-1	1	-1	-1	1	1	0	1

(32)

We make one pivot operation each on the bottom two primal rows in order to obtain a basic tableau, choosing pivots with value ±1 or ±2 (to satisfy theorem 7), resulting in:

t		t	t			w_x^1	w_b^1	v_1^s
4	0	0	-2	6	8	1	2	0
-1	0	1	2	-3	-4	-1/2	-3/2	0
-2	1	0	1	-2	-3	-1/2	-1/2	0
$\textcircled{-2}$	0	0	2	-4	-6	-1	-2	1

(33)

We have adjoined the tableau gate to tableau 33 both because it satisfies theorem 11, and because there are no suitable pivots for an adjoining calculation. Specifically, the only triads of ordered, alternating $g^1(\underline{x}) = \text{sgn}(\lambda^o - e)$ elements are in column zero, any one of columns one, two, or three, and either column four or five. None of these triads of columns have an alternating logic state sense (t, f, t or f, t, f) and therefore even though possible bfs among these triads would not be further constrained by the $g^1 y = 0$ equation, no bfs will remain since there are, in fact, no bfs among these triads of primal columns, by theorem 11.

Having adjoined g^1 to tableau 33, we pivot it into the basis using the circled element as the pivot, and undertake another existence calculation. An optimal tableau results after one more iteration:

$$\begin{array}{|cccccc|ccc|}
\multicolumn{6}{c}{} & w_x^2 & w_b^2 & v_1^2 \\
\hline
0 & 0 & -2 & 0 & 0 & -2 & -1 & -1 & 3 \\
\cline{7-9}
0 & 0 & 1 & 1 & -1 & -1 & 0 & -1/2 & -1/2 \\
0 & 1 & 1 & 0 & 1 & 2 & 1/2 & 1 & -3/2 \\
1 & 0 & 1 & 0 & 1 & 2 & 1/2 & 1/2 & -1 \\
\hline
\end{array} \qquad (34)$$

Note that all the primal columns in tableau 34 are flaggable; the bottom two primal rows will flag all primal columns except column three, then the element of the second row which is in the unflagged column is semipositive, and this results in flagging column three also. The resultant net for this function is

$$x,\ 1 * 3\ (* x,\ 2)\ , \qquad (35)$$

where we define x to be the sum of the x_i, $i = 1, 2, \cdots, 5$.

VI. EXPERIMENTAL RESULTS

A Fortran program for the general algorithm rules in Appendix II was written, and run on an IBM 7090 computer. The program accepts switching functions with up to nine logic variables. Among the switching functions realized were 122 four variable functions selected from Minnick's Table I (5), so as to provide a partial test of the degree of minimality of the algorithm. The nets listed in this table were assumed minimal, this assumption being based on the fact that several workers have improved or tested the minimality of the table by heuristic, hand applied methods (5), (7). Not all functions were selected because the intent of the present work is to produce an algorithm, not a table.

These 122 functions were selected from the 237 listed functions by an arbitrary process by which, it was hoped, the most "difficult" functions (those with a minimal number of distinct minimal-gate nets) would be selected, e.g., by selecting a two-gate function sandwiched in the table between a number of three-gate functions. The results bore this assumption out; viz., the results obtained are a conservative measure of the results that would have been obtained if all 237 functions had been selected.

Now, Minnicks table of nets contains two level nets only, and thus is not precisely comparable to general feed-forward nets. However, among the 122 functions selected, a majority (70) were two-gate nets, and thus are feed-forward nets. The forty three-gate functions are not precisely comparable, and this should be kept in mind in judging the results.

Assuming the nets in the table to be minimal, the present algorithm produced minimal nets for 65.5% of the functions realized, 31% of the nets required one gate more than the minimal number, and 3.5% of the nets were two gates over minimal. The nets produced were not identical to the table nets in most cases.

One of the most severe practical (circuit) constraints on threshold gate nets is a limitation on the maximum value allowed for the sum of the absolute values of the input weights for each gate in the net; viz., a fan-in constraint. A comparison of the 122 computed nets to the corresponding table nets showed that the maximum number of weights per net gate in the computed nets was 10% lower (averaged) than that of the table nets, these table nets requiring an average of ten weights (maximum sum in net); an average of nine being required for the computed nets. The bias weight was not counted.

There seems to be little similarity between the present algorithm and the algorithm developed by Minnick (5) for two-level nets, beyond the fact that they are both based on the Simplex algorithm (as mentioned previously), and experimental comparisons of the two algorithms are also difficult. Minnick does not give results for his algorithm directly (without heuristic hand reduction of his computed nets), and theoretical comparisons are also not very instructive, except for the fact of computational efficiency, as previously mentioned (section III). This latter statement stems from the fact that Minnick attempts to solve equation 1 directly, a procedure which is less efficient (requires more iterations on a larger tableau for solution).

As a measure of the computational efficiency of the present algorithm, the 7090 object program required an average of about 3/4 second to compute each net for the 122 four variable functions realized (with an average net containing between two and three gates). Nine variable nets appear to require from one-half to five minutes of computation time per gate.

There were three cases among the 122 functions realized where a (-1) pivot element had to be selected once in the process of one of the existence calculations, viz., when a tableau was neither optimal nor terminal, and no $\lambda_{ij} = 1$ pivot existed such that $\lambda_{oj} > 0$. The only apparent effect of this was to increase the total required computation time in some of these cases.

Finally, the symmetric algorithm was tested by hand on the symmetric odd parity functions (2) of 2, 3, 4, 7, 8, and 10 variables, resulting in symmetric nets with 2, 2, 3, 4, 4, and 5 gates, respectively. These are minimal symmetric nets, according to Kautz (2), except for the 7 and 10 variable nets which are each one gate over minimal. For an estimate of computational efficiency, about ten minutes were required to compute the five gate net for the ten variable parity function.

VII. CONCLUSIONS

An efficient algorithm has been presented which produces near-minimal threshold gate nets for given switching functions. It is of interest to consider possible modifications and extensions of this algorithm to account for further net constraints, such as gate fan in/out limits; threshold tolerance (reliability) requirements; net level restrictions; net topology constraints; nets where a cost is attached to the use of an inverted switching variable; nets with restricted types of threshold gates, such as allowing only $p < n+s$ switching variables as inputs to a gate, or nets of majority gates; and multiple output nets.

We shall briefly discuss these new algorithm constraints with the following qualification: the outlines given below of proposed methods are intended to be suggestive of research areas, and are not necessarily complete descriptions. Further study will be required to determine the best methods.

Many of these new constraints may be included by restating the original threshold inequalities I in a more conventional (but computationally inefficient) format:

Find $z = (w^s | \underline{w}^s | v^s | \underline{v}^s | x^s) \geq 0$

such that the cost

$$cz$$

is minimal, and such that

$$(w^s|\underline{w}^s|v^s|\underline{v}^s|x^s)\left[\begin{array}{c|c|cc|c}
A & & \multicolumn{2}{c|}{0} & \\
\cline{1-1}
-A & & & & \\
\cline{1-1}\cline{3-4}
G_{s-1} & e & E & 0 & d^3 \\
\cline{1-1}\cline{3-4}
-G_{s-1} & & 0 & E & \\
\hline
\multicolumn{4}{c|}{E} & 0
\end{array}\right] \qquad (36)$$

$$= \quad (-d^1 \quad |q| \quad d^2|\underline{d}^2|\ 0)$$

Here, c is the program cost vector; the underlined weight vector components correspond to inverted switching variables given by -A and $-G_{s-1}$; x^s is a required slack vector; e, E are respectively the unit vector and unit matrix of suitable size; vector d^1 is a threshold tolerance vector; parameter q is a fan in limit; vectors d^2, $\underline{d}^2$ are residual fan out limit vectors for the normal and inverted gate outputs, respectively; and vector d^3 is a weight nulling vector whose components are one or zero.

The proposed algorithm modifications and extensions will now be treated in order, using the L. P. (linear program 36), and other methods applied directly to the original algorithm. We shall not develop the details of applying the Simplex method to the L. P., nor of the required adjoining calculation if the L. P. has no solution.

Fan In Limit

The e vector column in the L. P. matrix expresses the constraint that the sum of the weight vector components be less than or equal to q, the maximum allowable fan in for gate g^s. Note that this fan in constraint cannot be added to the original algorithm in this manner because the original weight vector is not semipositive. However, an altered fan in constraint can be added to the original algorithm by restricting the maximum absolute value of the modified input sum to the gate.

Fan Out Limit

If u is the fan out limit on the sum of the output weights of each gate, then we set $d^2_h = u - \sum_{k=h+1}^{s-1} v^k_h \geq 0$, for h = 1, 2, ..., s-1, and similarly for $\underline{d^2_h}$, $\underline{v^k_h}$ (assuming that u applies individually to the normal and inverted gate outputs in the L. P.). This amounts to subtracting the previously fixed output weights from the fan out limit for each gate, to obtain residual fan out limits.

Threshold Tolerances

Reliability analysis of some types of threshold gate circuits, such as transistor circuits, leads to a required threshold tolerance band for each input state. In this case, we merely replace the e vector in inequality 1by the actual threshold tolerance vector d^1, so that this is easily included either in the original algorithm or in the L. P. For an undefined input state of the switching function, set the corresponding d^1 component equal to minus infinity. Note that this latter case is treated in detail for the original algorithm in appendix III.

Net Level Restrictions

The level of an r gate feed forward net will be less than r if some gates g^h have $v^h_{h-1} = \underline{v}^h_{h-1} = 0$. With the L. P. formulation, we can thus at least tend to restrict the net level by attaching high cost vector c components to v^s_{s-1} and $\underline{v}^s_{s-1}$, then decreasing these costs if required in order to give the L. P. a solution. Such L. P. parameter variations are the subject of the theory of parametric linear programming (4, p. 109 ff), and this theory is of significant potential use in generally extending the original algorithm.

For multiple gate nets, the maximum possible level restriction is to allow two level nets only, and we might briefly discuss this in terms of the original algorithm. One general method of obtaining two level nets is to pivot out all the tableau rows corresponding to previously adjoined gates after an unsuccessful existence calculation, by selecting one (±1/2) pivot element in each adjoined gate column of the dual tableau, thus causing the v^s tableau vector to identically vanish. The resultant non-basic primal rows could be used as cost rows for a flagging calculation. Alternatively, the adjoining calculation could consist of searching for a terminal tableau with a maximal number of bfs y in the basic portion of the primal which also satisfy the non-basic primal row constraints, since $\Lambda y = 0$ whether or not Λ is basic. This process amounts to attempting to adjoin tableau gates which remove a maximal number of remaining bfs, and requires further study; e.g., the number of possible primal columns in a bfs increases by one for each adjoined gate, even though the adjoined gate rows are non-basic. It is conceivable that research in this area could lead to a method for realizing theoretically minimal two level threshold gate nets.

Net Topology Constraints

It is possible for gate connection (wiring) constraints to exist which do not allow certain gates to be connected to certain higher level gates. One such example would be an etched circuit board construction of the net in fig. (1) which does not allow crossings of the gate output lines; i. e., a planar output net constraint. Another example would be a limit on the maximum physical distance between gates (wire length limit). The effect of these types of topological constraints is to require that certain weights $v_h^s = \underline{v}_h^s = 0$, in a determinable manner due to the successive way that the gates are calculated. We include this in the L. P. formulation by setting a component of vector d^3 equal to unity if the corresponding weight is to vanish, and zero otherwise. In the original algorithm, one can make a pivot operation on a (±1/2) dual pivot in each dual column which corresponds to a vanishing weight, and then ignore the resultant non-basic primal rows in the calculation.

Inversion Restrictions

Certain types of threshold gates, such as transistor circuits, often require the addition of inverter circuits if the inverse of a net switching variable is required, and the cost of such an inverter can approach the cost of a gate itself. In the L. P. formulation, the normal and inverted forms of switching variables have separate weights. Consequently, we could account for inversion cost by attaching higher cost vector components to the $\underline{w}^s$, $\underline{v}^s$ vectors than those attached to w^s, v^s. However, it would be more accurate to impose a fixed step increase in cost for each $\underline{w}_i^s$ or $\underline{v}_i^s$ which becomes non-zero, implying the use of parametric linear programming methods.

Restricted Threshold Gates

If any set of at most $p < n+s$ switching variables are allowed as inputs to gate g^s, then we could require that the d^3 vector in the L. P. have at most p zero elements. Parametric linear programming may be useful in selecting the proper d^3 components.

A majority gate in (-1, +1) logic state notation may be defined as a gate with a zero bias weight. This is easily imposed on either the L. P. or the original algorithm by removing the bias weight(s) and the bias unit vector row(s) from the set of constraints.

Multiple Output Nets

In terms of the L. P., a multiple output net which realizes switching functions $f_i(\underline{x})$ would lead to a constraint matrix having a block diagonal of constraint matricies for each $f_i(\underline{x})$. The off-diagonal blocks would correspond to intercoupling of the gate state vectors in the diagonal blocks.

For the original algorithm, we may propose the following type of method: starting with i = 1, apply the reversal rule of theorem 5 to the primal columns corresponding to conflicting logic states of $f_i(\underline{x})$ and $f_{i-1}(\underline{x})$, using the last tableau reached in the calculation for $f_{i-1}(\underline{x})$. Define $f_o(\underline{x}) = -1$; i. e., use the B tableau. Carrying a separate flag vector for each $f_i(\underline{x})$, undertake an existence calculation. If an optimal tableau results, adjoin the optimal tableau gate, pivot it into the basis, remove $f_i\ (\underline{x})$ from the set of functions to be realized, and repeat the method for $f_{i+1}\ (\underline{x})$. If a terminal tableau results, repeat with $f_{i+1}\ (\underline{x})$ without adjoining a gate. After all functions have been considered, undertake an adjoining calculation for the last unrealized function, but modify the $|N|$ count as follows: add unity to the conventional $|N|$ count for each primal column in N which corresponds to an input state for which one of the other unrealized $f_i\ (\underline{x})$ agrees with the tableau (last) switching function, once for each such agreement. Similarly, add unity for each disagreement corresponding to a P column. This process is an attempt to remove the maximum total number of possibly feasible bases for the $f_i\ (\underline{x})$ set, and will probably require modification after more detailed analysis. After completing the adjoining calculation, adjoin and pivot in the tableau gate, and undertake another series of existence calculations as before. The resultant net will have the form of fig. (1) where, for each i, there exists some $h \leq r$ such that $f_i\ (\underline{x}) = g^h\ (\underline{x})$; i.e., the unmodified output of the $h^{\underline{th}}$ gate is the $i^{\underline{th}}$ function.

VIII. REFERENCES

1. Shannon, C., Trans. AIEE (1938), 7, pp. 713-723.

2. Kautz, IRE Trans. on Electronic Computers (1961), EC-10, pp. 371-378.

3. Gale, Theory of Linear Economic Models (1960), McGraw-Hill.

4. Gass, Linear Programming (1958), McGraw-Hill.

5. Minnick, IRE Trans. on Electronic Computers (1961), EC-10, pp. 6-16.

6. McNaughton, IRE Trans. on Electronic Computers (1961), EC-10, pp. 1-5.

7. Klock, et al, Logic of Controlled Threshold Devices, ASTIA report no. EDC-6-62-1 (1962), p. 60.

8. Coates and Lewis, J. Franklin Institute (1961), 272, no. 5.

APPENDIX I

List of Symbols

A	Logic state matrix.
$\underline{A}$	Simplex primal program matrix.
B	Input state matrix (truth table).
c	Primal program cost vector.
d	Arbitrary vector.
e	The unit vector (1, 1,..., 1).
e^i	Unit row vector with unit for $i^{\underline{th}}$ component, zeros elsewhere.
e^j	Unit column vector with unity for $j^{\underline{th}}$ component, zeros elsewhere.
E	Set of primal tableau column indices identifying basis: $E \underline{\underline{\Delta}} \left\{ j \mid \lambda^j = \text{a unit column vector} . \right\}$
$f(\underline{x})$	Given bivalued switching function of n variables. A vector $f(\underline{x}^j) = (+1,\ j \epsilon T;\ -1,\ j \epsilon F)$.
F	False state matrix containing all columns of B corresponding to false states of f $(\underline{x})$. Set of primal tableau column indices such that indexed column corresponds to a false state of f $(\underline{x})$: $F \overset{\Delta}{=} \left\{ j \mid f(\underline{x}^j) = -1 \right\}$.
G_{s-1}	Matrix of modified logical output vectors of gates g^1 to g^{s-1}.
g^s	Gate s in net. Modified output vector of gate s; $g^s_j = \pm 1$ corresponding to the logical output of g^s for input state $\underline{x}^j$, reversed in sign for $j \epsilon T$.
h, i	Row indicies. Denotes row vector as superscript, unless noted.
j, k	Column indices. Denotes column vector as superscript, unless noted.
m	Number of columns in A or B.

n Number of logic variables x_h in f $(\underline{x})$.

N Set of primal tableau column indices such that first component of column is negative: $N \triangleq \{j | \lambda oj < 0\}$.

/N/ Number of indices in set N.

P Set of primal tableau column indices such that first component of column is positive: $P \triangleq \{j | \lambda oj > 0\}$.

r Number of gates in computed net realizing f $(\underline{x})$.

sgn Sign taking operator; sgn (x) = (+1, x $\geq$+1;-1, x $\leq$ -1).

T True state matrix containing all true columns of B. Set of indices such that indexed primal column corresponds to a true state of f $(\underline{x})$: $T \triangleq \{j | f(\underline{x}^j) = 1\}$.

U Unit matrix with first column omitted.

v^s Vector of input weights to g^s from lower level gates; v_h^s is weight associated with input to s^th gate from output of h^th gate.

w^s Vector of input weights to gate g^s from direct input logic variables; w_h^s is weight associated with input to g^s from x_h (w_{n+1}^s is bias weight).

w_b^s Symmetric algorithm bias weight.

w_x^s Symmetric algorithm common input weight for all direct inputs x_h.

$\underline{x}$ Direct input state vector. State x^{j-1} is j^th vector in ordering (-1, -1 ..., -1, -1), (-1, -1, ..., -1, +1),..., (1, 1,..., 1, -1), (1, 1,..., 1, 1).

y Vector of primal program variables.

y_a Artificial form of y; y_a = (1, y).

Z Set of primal tableau column indices such that indexed column has zero first component: $Z \triangleq \{j | \lambda_{oj} = 0\}$.

Λ Primal tableau matrix with elements λij.

Γ Dual tableau matrix with elements γ ij.

APPENDIX II

Summary of Algorithm Rules

1. Given a switching function f($\underline{x}$) of n variables, construct the basic B tableau of fig. (3) for this value of n. Begin phase 2.

2. Apply the reversal rule of theorem 5 to all primal columns with indices j such that f ($\underline{x}^j$) = 1 (true state columns). If any primal unit column vectors are modified, make a pivot operation on the (-1) element in each such column to restore its unit vector. Set a gate counter s equal to unity. Make a first flagging inspection by flagging each primal column which has a non-zero element in any semipositive primal row (except the top row). Begin phase 3.

3. Undertake an existence calculation by making a sequence of pivot operations on unity primal elements which are not in the top row, and which have maximally positive top elements in the pivot columns. After each such iteration make a flagging inspection; however, search for either semipositive or seminegative primal rows, and ignore primal row elements which are in previously flagged columns. This generalized flagging inspection is to be made after all subsequent pivot operations, including those in phases 4 and 5. Repeat phase 3 until no primal column has a positive top element and another element equal to unity. When this occurs, inspect the top primal row to see if it is seminegative. If so, the calculation is finished, and the top dual row will contain the weight vector (w, v) of the last (output) gate in an s-gate net realizing f($\underline{x}$). If the top primal row is not seminegative, inspect the primal

columns with positive top elements to determine whether all other elements are seminegative in these columns. If so, the tableau is terminal, and phase 4 is begun. If not, make a special pivot operation with any (-1) primal element in a column with a negative top element, and repeat phase 3. If none exist, select a (±1) primal pivot, preferably in a pivot column with a non-zero top element, make this special pivot operation, and repeat phase 3.

4. Undertake an adjoining calculation by first determining whether negative elements exist in the top primal row, and which are in unflagged columns. If none exist, begin phase 5. Otherwise, select a (+1) primal pivot in a column with the maximally (absolute) negative top element possible, such that the indicated pivot operation will not generate twos in top primal row elements which are in unflagged columns. Make a generalized flagging inspection after performing the pivot operation. Repeat phase 4 until no suitable pivot exists, at which point phase 5 is begun.

5. Adjoin the tableau gate by first writing down the top dual row as the weight vector (w, v) for gate g^s in the net. Adjoin a new bottom row and a new dual column to the tableau by theorem 6. Select a pivot element in the primal portion of the new bottom row which is equal to (+2) if possible (for empirical reasons); otherwise select a (-2) pivot. If no such (±2) pivot exists, perform a special pivot operation on any (±1) primal pivot which will generate at least one (±2) element in the new bottom row (see below). Having a (±2) pivot in the new primal row, divide the entire row by the selected pivot, including the new unit vector unity element, and

perform the indicated pivot operation. Increase gate counter s by unity and begin phase 3.

Remarks: For automatic computation using a digital computer which internally carries integers in right-normalized binary, the skew-transposed binary matrix in fig. (3) may be easily constructed by successively extracting the bits of the binary representation of $j = 0, 1, 2, \ldots, m-1$, and inserting them into the elements of primal column j. The entire computation may be carried out with integers if twice the dual tableau is used (the gate weight vectors are then equal to the top dual row divided by two).

If a medium or large computer is available, an empirical improvement in average net minimality and a reduction in the average total number of iterations required per net may be obtained by programming a somewhat more complex set of pivot rules. If the computer command structure and memory size are adequate, the following modifications are recommended: in place of the requirement than an existence calculation pivot satisfy ($\lambda ij = 1; \lambda oj = \max. > 0$, substitute: ($\lambda ij = 1; \lambda oj > 0$; $\Delta /N+Z/ = \max.$), where Δ is to be read as "algebraic change in". In the adjoining calculation, replacing the requirement ($\lambda ij = 1; \lambda oj = \min. < 0$; $\lambda ok \neq 2$, for all k), substitute: ($\lambda ij = 1, \lambda oj < 0$; $\lambda ok \neq 2$, for all k; $\Delta /N/ = \min.$). These rules require trial partial pivot operations to be made, similar to those previously required in the adjoining calculation to insure that twos will not be generated in the top primal row. Specifically, that allowable existence calculation pivot ($\lambda ij = 1; \lambda oj > 0$) is chosen which generates a maximal number of zero or negative top primal row elements, and that allowable adjoining calculation pivot ($\lambda ij = 1$; λoj; <0; λ ok $\neq$ 2, for all k) is chosen which generates a minimal number of negative top primal row elements. Note, however, that /N+Z/ or /N/ could actually decrease or increase in a given iteration of the existence or adjoining calculations, respectively, since we are measuring the algebraic change in the size of these sets.

Finally, if the algorithm is programmed on a large computer, it is possible to realize switching functions with a relatively large number of variables; e.g., $n = 9$. If such a function requires more than a few gates; e.g., six or more, it has been found in practice that the computer program can, in rare cases*, generate a tableau gate which cannot be pivoted into the basis without violating theorem 7; viz., no (±2) pivot element exists in the new primal row. In these cases, it was always found possible to adjoin the tableau gate and make a special pivot operation before pivoting the gate into the basis. Namely, make trial partial pivot operations on (±1) primal pivots until one is found which will generate a (±2) element in the new primal row. This generated (±2) element can then be used as a pivot to generate the new primal unit column vector, and the next existence calculation then proceeds as usual.

*Excessive computation time would have been required to empirically determine how rare these cases are; consequently, no measure of this will be given.

APPENDIX III

Partially Defined Switching Functions

Theorem 5 allows us to obtain a first basic tableau for A from the basic B tableau in fig. (3) only if each input state is defined by $f(\underline{x})$ to be either true or false. If some input states are undefined, replace theorem 5 by:

Theorem 12

A first basic tableau for A may be obtained by applying the following procedure to the basic B tableau: if $f(\underline{x}^j) = 1$, apply the reversal rule of theorem 5 to λ^j. If $f(x^j) = -1$, leave λ^j undisturbed. If $f(\underline{x}^j)$ is undefined, remove column λ^j from the tableau. After forming this reduced tableau, restore the modified unit vectors as in theorem 5, and inspect the remaining primal columns for possible missing (removed) unit vectors e^{k+1}. Perform a sequence of pivot operations, using pivot elements $\lambda kj \neq 0$, until the reduced primal tableau is restored to basic form; ie., contains one unit column vector for each row (except the top row). If a reduced primal row vanishes at any stage, make a pivot operation on this row with any non-zero pivot in the dual tableau, and then remove the pivot row and the (dual) pivot column from the tableau.

proof: An undefined input state produces no corresponding constraint in the set of inequalities 1, and thus no corresponding primal column in the tableau. Consequently, we may merely remove primal columns from the basic B tableau which correspond to undefined states of $f(\underline{x})$. However, the reduced tableau so obtained should be made basic in order to apply the algorithm rules to it. As discussed in section III, a tableau may be made basic by performing a sequence of pivot operations which generate one unit primal column vector for each tableau row except the top row. This process will fail only if there are no non-zero pivots in one or more primal rows at some stage; ie., if some primal rows vanish. If a reduced primal row vanishes, then there must exist a linear dependence among the rows of the corresponding reduced A matrix, because the pivoting process can only fail if the primal tableau rank is less than n+1.

Now, the existence of such a linear dependence implies that one or more of the weights w_i are superfluous, in that they can be identically set to zero without decreasing the number of threshold gates attainable from the reduced A matrix. Specifically, for every linear input sum $d = wAr$, where Ar is the reduced A matrix, there exists a weight vector $\underline{w}$ with the superfluous $w_i = 0$, such that $d = \underline{w}Ar$. Consequently, we can account for a vanishing primal row by removing a row a^i from Ar, thus setting $w_i = 0$, without loss of generality. In terms of the tableau, this may be performed by making a pivot operation in the dual portion of a zero primal row, and then removing both the pivot row and (dual) pivot column from the tableau (this is a reversal of the procedure originally used to obtain the B tableau). Furthermore, at least one such non-zero dual pivot element must exist, since all dual matrices are non-singular.

After each reduced primal row has been either pivoted in or out of the primal basis, it can be seen that the resultant reduced tableau will satisfy all former theorems relating to full tableaus, with the exception that theorem 1 has been modified to state that the reduced primal tableau has a rank equal to its number of rows (excepting the top row).

Now, if we also impose the (±1) pivot constraint of theorem 7 in order to maintain integral and allowable* weight vectors, we may be sometimes forced to make special pivot operations in order to obtain a first basic tableau; viz. when a primal row is non-zero, but contains no (±1) pivot, or where a primal row vanishes and no (±1/2) pivot exists in its dual portion. This (±1/2) dual pivot choice comes from a simple extension of theorem 7, and it might also be noted that it is permissible to leave a zero primal row in the tableau if desired (no dual pivot operation), since the algorithm will automatically ignore it. However, if no (±1) pivot exists in a non-zero primal row, we must attempt to generate one by searching for any (±1) reduced primal pivot which will generate a (±1) element in the desired row, then using this generated pivot to form the required unit vector.

It must be stated that the use of theorem 7 can theoretically restrict the number of distinct gates attainable in a reduced tableau. The practical consequences of using both theorems 7 and 12 together are not precisely known. However, the method was experimentally tested on

*Note that if this pivot constraint is imposed, the weight vectors will also be allowable for the undefined states, since the proof of theorem 7 is not invalidated by the procedure of theorem 12.

about one-hundred four-variable switching functions, with zero to about eight arbitrarily chosen undefined states, and no failures of the method occurred. No special iterations were required to generate (±1) pivots in order to obtain basic tableaus*, and no reduced primal rows vanished.

Alternate methods can be used for partially defined switching functions. For example,we could arbitrarily group the undefined states into F to begin the calculation, and then allow unrestricted application of the reversal rule of theorem 5 to the undefined columns as required; e.g., we can convert undefined P columns to N columns, and vice-versa, in order to facilitate the existence and adjoining calculations on the full tableau. However, this does not take advantage of the increase in computational efficiency made possible by reducing the size of the tableau, and the method of theorem 12 seems to be adequate in practice.

To illustrate theorem 12, consider the basic B tableau for n = 3:

$$\left[\begin{array}{cccc:cccc|cccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
1 & 0 & 0 & -1 & 0 & -1 & -1 & -2 & -1/2 & -1/2 & -1/2 & -1/2 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1/2 & 1/2 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \enclose{circle}{1/2} & 0 & 0 & 1/2
\end{array}\right] \tag{37}$$

First assume that only the first four primal columns are defined. Then the reduced primal tableau will have a zero bottom row, and we can pivot on the circled dual element, remove the bottom tableau row and

*However, it is simple to deliberately construct artificial cases where such iterations are required.

first dual column, and thus obtain a basic reduced tableau with $w_1 = 0$. We shall not actually specify the defined states of $f(\underline{x})$ here, because the intent is to illustrate the procedure followed when a reduced primal row vanishes.

Now let us consider another case in detail. Assume that primal columns zero to seven in tableau 37 are to correspond, respectively, to $f(\underline{x})$ states (+1, +1, ±1, -1, ±1, ±1, +1, -1), where a (±1) state is undefined. Group the defined columns (0, 1, 3, 6, 7) into a first reduced tableau, apply the reversal rule to the true state columns, and restore the two modified unit vectors:

$$\left[\begin{array}{ccccc|cccc}
0 & 0 & 0 & 4 & -2 & -1 & -1 & 0 & -1 \\
1 & 0 & 1 & -1 & 2 & 1/2 & 1/2 & 1/2 & 1/2 \\
0 & 1 & -1 & 0 & -1 & 0 & 0 & -1/2 & -1/2 \\
0 & 0 & 1 & -1 & 1 & 0 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & -1 & \textcircled{1} & 1/2 & 0 & 0 & 1/2
\end{array}\right] \tag{38}$$

We note that one primal unit vector is missing from tableau 38, and we generate it by using the circled pivot element. This gives:

$$\left[\begin{array}{ccccc|cccc}
0 & 0 & 0 & 2 & 0 & 0 & -1 & 0 & 0 \\
1 & 0 & 1 & \textcircled{1} & 0 & -1/2 & 1/2 & 1/2 & -1/2 \\
0 & 1 & -1 & -1 & 0 & 1/2 & 0 & -1/2 & 0 \\
0 & 0 & 1 & 0 & 0 & -1/2 & 1/2 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & 1/2 & 0 & 0 & 1/2
\end{array}\right] \tag{39}$$

Tableau 39 is basic, and a single existence calculation pivot operation (using the circled pivot) will give an optimal tableau. The weight vector of the resultant single gate net will be $w^1 = (1, -2, -1, 1)$.