ESD-TR-69-244

Technical Note 1969-48

B. E. White

Efficient Realization of Boolean Functions by Pruning NAND(NOR) Trees

2 September 1969

Prepared under Electronic Systems Division Contract AF 19(628)-5167 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Lexington, Massachusetts

~~AD693565~~

AD693563

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

EFFICIENT REALIZATION OF BOOLEAN FUNCTIONS BY PRUNING NAND(NOR) TREES

B. E. WHITE

Group 66

TECHNICAL NOTE 1969-48

2 SEPTEMBER 1969

LEXINGTON MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Department of the Navy under Air Force Contract AF 19(628)-5167.

ABSTRACT

A combinatorial tree structure composed entirely of NAND(NOR) blocks is pruned in a non-exhaustive fashion to yield minimal or near-minimal networks. It is assumed that complemented variables are not available and that there are no fan-in or fan-out limitations. The cost of a network is taken as being primarily determined by the number of logic blocks with the number of inputs and logic levels as secondary factors. The pruning algorithm lends itself to both hand methods and machine computation, although the synthesis procedure has not been programmed.

Of the 68 nondegenerate functional equivalence classes of 3 variables, the minimum number of blocks results in 63 cases; only one more block in excess of the minimum is required in each of the other 5 cases. For 18 randomly selected Boolean functions of 4 variables, the tree solutions yield an average of 8.2 blocks per function with the following distribution:

number of logic blocks	5	6	7	8	9	10	11
number of tree solutions	1	0	2	9	4	1	1 .

It is shown that the linear function $f(n) = x_0 \oplus \ldots \oplus x_n$ of n variables or its complement can be realized with 3n-2 NAND(NOR) blocks, for $n \geq 3$.

Accepted for the Air Force
Franklin C. Hudson
Chief, Lincoln Laboratory Office

This work is based on the author's Masters thesis, "Implementing Combinatorial Logic With the NAND Tree," September 1962, for the Department of Electrical Engineering at Massachusetts Institute of Technology, Cambridge, Massachusetts.

I. INTRODUCTION

The synthesis of combinatorial NAND(NOR) logic is a subject that has generated considerable interest in the last ten years, primarily because of its application in the design of transistor digital computers. Much of the basic work in this area was done by G. A. Maley [1] and his colleagues. Among the published papers in the field, [2, 3, 4 and 5] are representative; additional references are listed in [5] and a Russian survey article[6]. Another interesting paper[7] submitted recently has yet to appear formally but is available. Hand methods of synthesis are emphasized in [1,2], while [4, 5] discuss algorithms that lend themselves to computer implementation, as well. Under a fan limit equal to the number (three) of variables, Hellerman [3] used a computer to exhaustively find all minimal networks for every functional equivalence class. In [7] Hellerman's solutions are obtained with no fan restriction by a machine algorithm that is non-exhaustive but apparently too formidable for a hand method. Gimpel[4] solves the problem of finding the minimum three-level network with complemented variables unavailable using a method similar to but more complex than the well-known Quine-McCluskey algorithm for two-level AND/OR networks. Dietmeyer and Su[5] concentrate on algorithms for fan-limited blocks and assume complemented variables are available.

Here the author is concerned with the efficient synthesis of minimal or near-minimal NAND(NOR) networks with complemented variables not available, where the number of logic blocks is considered the principal cost factor with the number of inputs and levels of secondary importance. In contrast to other approaches in which irredundant networks are constructed algebraically from given Boolean functions, the present synthesis procedure is devised to destroy redundancies in canonic networks that implement the desired functions from the outset. As a hand method for, say, up to five or six variables, this approach has the features of allowing the designer to see the network converge to an irredundant form and to perhaps make improvements in the realization by intuition. As an algorithm for computer execution, the procedure appears relatively straight-forward to program and, being non-exhaustive, suggests the potential for rapidly obtaining good solutions for functions of four or more variables. At this time precise information on typical computation speeds cannot be reported because this version of the algorithm has not been programmed.

The canonic network used in the synthesis is a combinatorial tree structure composed only of NAND(NOR) blocks. The tree can be thought of as a network implementation of a Karnaugh map or logical truth table. The basic tree structure was originally suggested by J. Earle [1 - p. 154] as a means of rapidly obtaining a multiple-output network. However, only single-output networks are considered here.

The combinatorial tree is discussed in III following a brief review of NAND(NOR) logic in II. The principles used in removing tree redundancies are presented in IV. Some results of the algorithm outlined in V are given in VI. A few conclusions are included in VII.

II. NAND(NOR) LOGIC

Although the reader is assumed to be familiar with Boolean algebra and NAND(NOR) logical design, a brief review of some basic principles is given here. This section highlights the duality between NAND and NOR logic in order to motivate subsequent discussion that is valid whether a network is composed entirely of NAND blocks or only of NOR blocks.

A NAND(NOR) block output is the logical complement of the logical product (sum) of the block inputs. The logical operations $\wedge$, $\vee$ and $-$ can be realized solely with NAND(NOR) blocks as shown in Fig. 1 for the binary variables x_0 and x_1. Therefore, a NAND(NOR) block is called a universal logical connective in the sense that any Boolean function can be realized using only NAND(NOR) blocks.

Let a Boolean function of $n \geq 1$ binary variables be expressed in sum-of-products or product-of-sums form as

$$f(n) = f(x_0, \ldots, x_k, \ldots, x_{n-1}) = \bigvee_{i \epsilon F} m_i = \bigwedge_{i \epsilon G} M_i, \tag{1a}$$

with the i^{th} product or sum given by

$$m_i = \bigwedge_{k=0}^{n-1} [b_k^i x_k + (1 - b_k^i) \bar{x}_k]$$
$$M_i = \bigvee_{k=0}^{n-1} [b_k^i x_k + (1 - b_k^i) \bar{x}_k] , \tag{1b}$$

where the n-tuple

$$\beta_i = b_{n-1}^i \ldots b_k^i \ldots b_0^i \tag{2a}$$

is the binary equivalent of the decimal number

$$i = \sum_{k=0}^{n-1} b_k^i 2^k , \tag{2b}$$

and F and G are subsets of the universal set

$$U = \{0, \ldots, i, \ldots, 2^n - 1\} . \tag{3}$$

The complement of f(n) can be written as

$$\overline{f(n)} = \bigvee_{i \in \overline{F}} m_i = \bigwedge_{i \in \overline{G}} M_i \; . \tag{4}$$

The dual of f(n) is defined as

$$f_D(n) = \bigwedge_{i \in F} M_i = \bigvee_{i \in G} m_i , \tag{5}$$

which from (1) is seen to be accomplished by the interchange of $\bigvee$ and $\bigwedge$ or, equivalently, the substitution of NOR(NAND) blocks for NAND(NOR) blocks in Fig. 1. The relationship between F and G in (1a) and (5) is easily established as

$$G = \{2^n - 1 - i\} \ni i \in \overline{F} , \tag{6}$$

since from (1), (2) and (4)

$$\begin{aligned} f(n) &= \overline{\bigvee_{i \in \overline{F}} m_i} = \bigwedge_{i \in \overline{F}} \overline{m}_i \\ &= \bigwedge_{i \in \overline{F}} \left\{ \bigvee_{k=0}^{n-1} [b_k^i \overline{x}_k + (1 - b_k^i) x_k] \right\} \\ &= \bigwedge_{i \in \overline{F}} M_{2^n - 1 - i} \; . \end{aligned}$$

The function associated with a given NAND(NOR) block can be completely specified by (from) the set of subscripts C = $\{i\}$ on the m's (M's) in the sum-of-product (product-of-sums) expression equivalent to the logical product (sum) of the block inputs. The set $\{i\}$ ($\{2^n - 1 - i\}$) specifies the combinations of binary variables according to (1b) and (2) for which f(n) must assume the

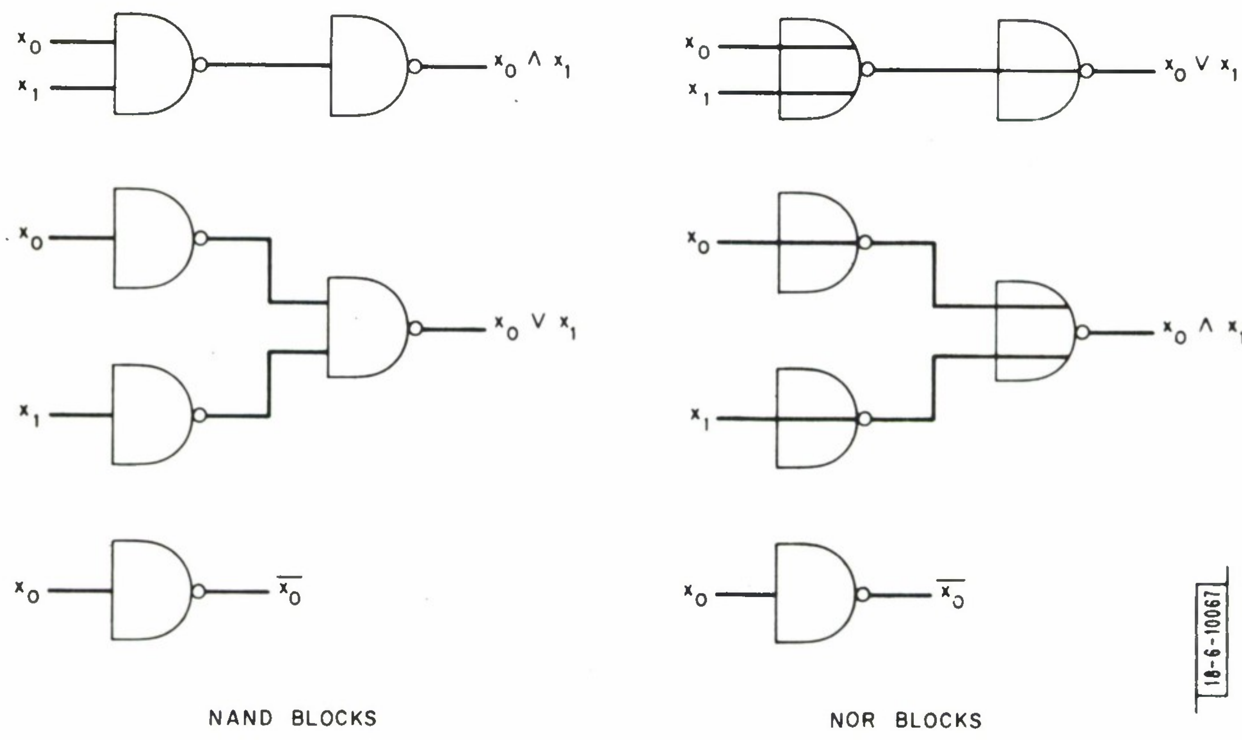


Fig. 1. Universal logical connectives.

logical value 1(0) if the given NAND(NOR) block feeds a common output NAND(NOR) block. Thus, C can be called a functional cover, and the union over the set O of covers of NAND(NOR) blocks feeding the output NAND(NOR) determines the 1(0) network outputs as $F = \bigcup_{O} \{i\} \left(\overline{F} = \bigcup_{O} \{2^n - 1 - i\}\right)$.

It is implicitly assumed that all NAND(NOR) blocks include a constant input of 1(0) so that in the absence of binary inputs, the NAND(NOR) block output is 0(1). When a binary input to a NAND(NOR) block is 1(0), that input might as well be absent since it cannot affect the block output; when a binary input is 0(1), it forces the block output to 1(0) regardless of the logical values of other block inputs. A straightforward application of these elementary ideas can be used to obtain the following useful property of NAND(NOR) logic.

Property 1. Given a set S of NAND(NOR) blocks, none of the binary inputs common to S need appear as inputs to any other NAND(NOR) block whose output is connected to every block in S via some logic path and yet does not lead to a path termination without connecting with some block in S. Special cases of Property 1 are:

a) Binary inputs common to a NAND(NOR) block and all NAND(NOR) blocks it feeds can be removed at that block.

b) Binary inputs identical to those at the output NAND(NOR) block can be removed everywhere else they appear in the NAND(NOR) network.

III. THE n-TREE

In III and IV the duality between NAND and NOR logic is exploited so that the explicit mention of NAND(NOR) blocks is unnecessary. The results are essentially the same whether a NAND or a NOR network is utilized. An attempt is made to employ concise terminology that conveys an image consistent with the tree structure of the network.

The n-tree is a combinatorial network of nodes, branches and roots. Using (3), each node is distinguished by a label $i \in U$. Referring to (2), node i has among its inputs the set of branches

$$X_i = \{x_k\} \ni b_k^i = 1 \text{ in } \beta_i . \tag{7}$$

Every node has a single implicit output. The remainder of the inputs to node i come from the outputs of other nodes and are called roots. If $\bigwedge$ is taken as componentwise multiplication for n-tuples, then the set of roots at node i is

$$R_i = \{j\} \ni \beta_j \wedge \beta_i = \beta_i , \quad j \neq i, \tag{8}$$

where $j \in R_i$ means that node j feeds node i. From (7) and (8), the complete set of n-tree nodes including X_i as branches is

$$T_i = \{i\} \cup R_i . \tag{9}$$

It is sometimes convenient to label the tree level

$$\ell_i = \text{number of 1's in } \beta_i \tag{10}$$

of node i. With this notation the roots at node i can be grouped as $n - \ell_i$ mutually disjoint sets

$$R_i = \bigcup_{\ell = \ell_i + 1}^{n} R_\ell^i , \tag{11}$$

where R_ℓ^i is the set of nodes on level ℓ that feed node i. From (7), the number of nodes on level ℓ of the n-tree is $\binom{n}{\ell}$, so the total number of n-tree nodes is

$$2^n = \sum_{\ell=0}^{n} \binom{n}{\ell}. \tag{12}$$

The number of branches at node i is ℓ_i. From (8), the number of roots at node i from level $\ell > \ell_i$ is

$$\binom{n-\ell_i}{\ell-\ell_i},$$

so the total number of roots at node i is

$$2^{n-\ell_i} - 1 = \sum_{\ell=\ell_i+1}^{n} \binom{n-\ell_i}{\ell-\ell_i}. \tag{13}$$

To help clarify the notation, the tree structure for $1 \le n \le 3$ is shown in Fig. 2. Node i is indicated by i enclosed in a square. The inputs to node i are given immediately above the square with the roots from the same level grouped together. Note that the number of branches and roots at a node increases and decreases, respectively, as the level of a node increases.

Summarizing the structure of the n-tree in one sentence, node $i \in U$ has as branches a distinct set X_i selected from $x_0, \ldots, x_{n-1}$ and is fed by all higher-level nodes with branches that include X_i as a subset. This statement easily leads to the following simple property of the n-tree.

Property 2. Given any node in the n-tree, all roots at that node are included in the roots at every n-tree node fed by the given node, i.e., if node j feeds node i then $R_j \subset R_i$.

An arbitrary function to be synthesized is uniquely specified by

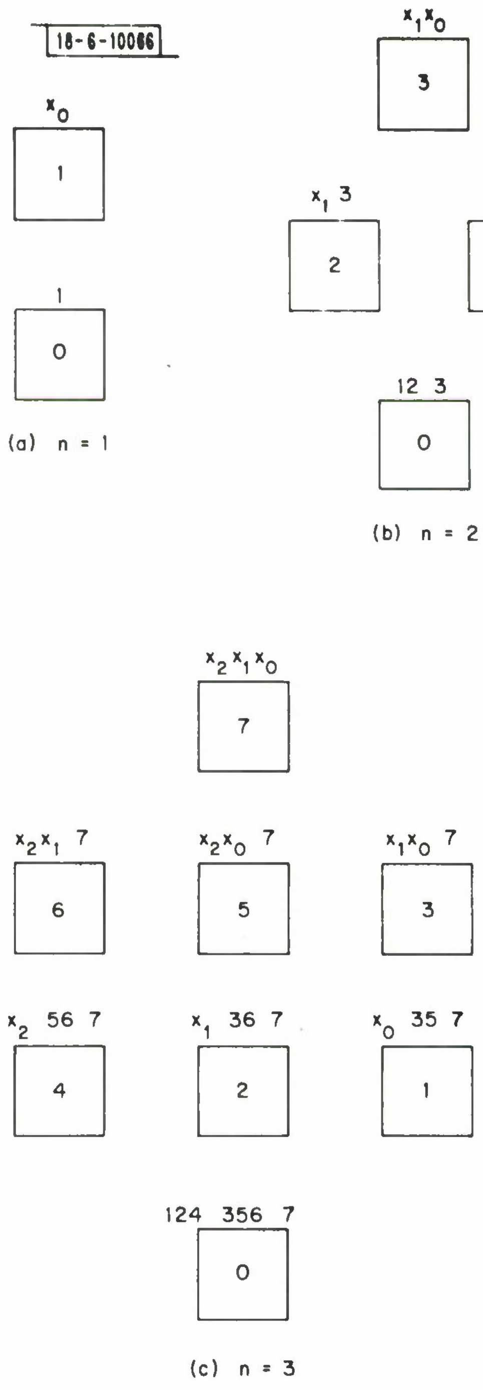


Fig. 2. The n-tree structure ($1 \leq n \leq 3$).

$$F = \bigcup_{\ell=0}^{n} F_\ell \subset U, \qquad (14a)$$

where F_ℓ is the set of n-tree nodes in F on level ℓ; for convenience the remaining nodes on level ℓ are designated $\overline{F}_\ell$, so

$$\overline{F} = \bigcup_{\ell=0}^{n} \overline{F}_\ell \subset U. \qquad (14b)$$

If C_j is the functional cover for node j, then from II, (8) and (9) the appropriate cover for node i fed by node j is the recursive definition

$$C_i = T_i \bigcap_{j \in S_i} \overline{C}_j, \qquad (15)$$

where $S_i \subseteq^{*} R_i$ is the set of roots saved at node i. From (15) if $S_i = \emptyset$, then $C_i = T_i$. From (9) and (15), if $S_i = R_i$, then

$$C_i = (\{i\} \cup R_i) \bigcap_{j \in R_i} \overline{C}_j = \{i\},$$

which is shown by induction as the level ℓ is decreased from n, i.e., for $\ell = n$, $i = 2^n - 1$ and $C_i = \{i\}$ since $R_i = \emptyset$; assuming that $C_j = \{j\}$ for node j on level $\ell_j > \ell_i$,

$$C_i = (\{i\} \cup R_i) \bigcap_{j \in R_i} \overline{\{j\}}$$

$$= (\{i\} \bigcap_{j \in R_i} \overline{\{j\}}) \cup (R_i \bigcap_{j \in R_i} \overline{\{j\}})$$

$$= \{i\} \cup \emptyset = \{i\},$$

since $j \neq i, \forall j \in R_i$. With the addition of a common output node external to the n-tree called the tree base, and using (14), the following property of the

*Inclusion is indicated here in anticipation of the removal of redundant roots; actually, with the tree grafts defined in IV, S_i may contain roots disjoint from R_i.

n-tree is established.

Property 3. The n-tree realizes F iff all nodes in F and no nodes in $\overline{F}$ feed the base.

Introduced here is the notion of certain embedded tree structures utilized in the sequel. An m-tree is the network of nodes, branches and roots of the n-tree consisting only of the set of 2^m nodes (and their branches and roots)

$$T_i^j(m) = \{i+k\} \ni \beta_k = \beta_k \wedge \beta_{j-i}, \; j \geq i, \tag{16a}$$

where i, j ϵ U, $\beta_j \wedge \beta_i = \beta_i$, and $0 \leq m \leq n$. If $m = n$, the m-tree and n-tree are identical; if $m < n$ the m-tree is a subtree structure embedded in the n-tree. Note that in (9) and (15),

$$T_i = T_i^{2^n-1}(n - \ell_i) .$$

An m-knot is a set

$$K_i^j(m) = T_i^j(m) \ni T_i^j(m) \subseteq F \; (\overline{F}), \tag{16b}$$

i.e., the nodes of the m-tree corresponding to an m-knot are either all in F or all in $\overline{F}$.

An m-nest is a set

$$N_i^j(m) = T_i^j(m) \ni k \; \epsilon \; F \text{ iff } \ell_k \text{ even(odd)}, \forall k \; \epsilon \; T_i^j(m), \tag{16c}$$

i.e., in the m-tree corresponding to an m-nest, all the nodes on even levels are in F($\overline{F}$) and all the nodes on odd levels are in $\overline{F}$(F).

IV. PRUNING TECHNIQUES

With Property 3 as a beginning, the objective of an efficient synthesis is to prune the n-tree by the orderly removal of logically redundant nodes, branches and roots. A node that is saved to permit the pruning of at least one other node is called a stump.

In addition to direct pruning, the network can be reduced indirectly by other techniques. In particular, a tree graft refers to the attachment of a new root at a given n-tree node for the purpose of supplanting other roots at that node. A tree growth refers to the addition of a node different from the base but external to the n-tree which facilitates further reductions. Unless the situation warrants the more specific terms, pruning is used generically to include tree grafts and growths.

The set of nodes feeding the base is designated

$$O = O_t \cup O_g, \tag{17a}$$

where O_t and O_g is the set of n-tree and growth nodes, respectively, that feed the base; by Property 3,

$$O_t \subseteq F. \tag{17b}$$

If C is the functional cover associated with a given node,

$$F = \bigcup_{O} C = C_g \bigcup_{i \in O_t} C_i, \tag{17c}$$

where C_g represents the total cover of the growth nodes in O_g, and C_i is defined by (15).

Offered here are some rules of redundancy in theorem form that suggest effective pruning methods. Implicit in the statement of some of the following results is the assumption that in pruning the n-tree the logical output function of the network remains unchanged.

Since C_i depends on the covers of nodes at higher tree levels that feed node i, it is efficacious to prune the n-tree from the top level-by-level. With this order of pruning, the proofs of most redundancy rules, viz., Theorems 1, 2, 4 and 5 are also simplified.

Theorem 1. At every node $i \in \overline{F}$ (that does not feed a node of a tree growth), any set $\subseteq R_i$ of roots can be pruned.

Proof: By Property 2, $R_j \subset R_i$ when n-tree node j feeds n-tree node i. By Property 3 nodes in $\overline{F}$ do not feed the base. Thus, barring any node in $\overline{F}$ that feeds a node of a tree growth*, the roots R_j appear everywhere node j feeds. By Property 1a any subset of R_j can be pruned at node j.

Theorem 2. At every node $i \in F$ (that does not feed a node of a tree growth), any set $\subseteq R_i \cap F$ of roots can be pruned.

Proof: Suppose a root $j \in R_i \cap F$ appears at the base. Then by Property 1b, that root can be pruned anywhere else it appears in the network. If neither node i nor node j feeds the base, then by Property 2, root j can be pruned at node i, provided i does not feed a node of a tree growth (see footnote of Theorem 1). It remains to check the case when i but not j appears at the base.

While j appears at the base $C_j \subset F$, for if $C_j \supset F$, the network would not realize F; $C_j \neq F$, since $i \in F$ and $i \notin C_j$. At the base j is pruned only if C_j is covered by other nodes feeding the base, again, because F would not be realized otherwise. With j pruned at the base and node i feeding the base, if root j does not appear at node i, then root j has already been pruned legitimately, by definition; if root j appears at node i, then from (15)

$$C_i = T_i \cap \overline{C_j} \bigcap_{j \neq r \in S_i} \overline{C}_r \quad ,$$

*As is seen in Theorem 5, the node of a tree growth may have only one root, say root j, which cannot be a member of R_j, from (8); actually, only nodes in $\overline{F}$ feed tree growth nodes.

that becomes

$$(C_j \cup \overline{C}_j)\, T_i \underset{j \neq r \in S_i}{\cap} \overline{C}_r = T_i \underset{j \neq r \in S_i}{\cap} \overline{C}_r$$

with the removal of root j at node i, i.e., there is an addition of

$$C_j \cap T_i \underset{j \neq r \in S_i}{\cap} \overline{C}_r \subset C_j \subset F$$

to the cover supplied by node i, which does not alter the function realized by the network since C_j is already covered at the output.

Corollary 1. No more than a three-level realization of F is always possible.

Proof: If all the roots at every node in $\overline{F}$ are pruned by Theorem 1, and if all the roots in F at every node in F are pruned by Theorem 2, then all nodes in $\overline{F}$ can only feed nodes in F, and nodes in F can only feed the base. Tree growths are not involved in such realizations which, obviously, are of three logic levels or less.

Theorem 3. If node j feeds node i and $C_k \subset C_j$ for $k \in R_j$, then root k can be pruned at node i.

Proof: By Property 2, if $k \in R_j$, then $k \in R_i$. If $C_k \subset C_j$, then $\overline{C}_k \supset \overline{C}_j$. From (15),

$$C_i = T_i \cap \overline{C}_j \cap \overline{C}_k \underset{j,k \neq r \in S_i}{\cap} \overline{C}_r$$

$$= T_i \cap \overline{C}_j \underset{j,k \neq r \in S_i}{\cap} \overline{C}_r$$

is independent of root k at node i, since $\overline{C}_j \cap \overline{C}_k = \overline{C}_j$.

Corollary 2. If node j feeds node i and $S_j = \phi$, then the roots R_j can be pruned at node i.

Proof: From (15), with all roots R_j pruned at node j, $C_j = T_j = \{j\} \cup R_j$. Since $C_k \subseteq T_k = \{k\} \cup R_k$, and $R_k \subset R_j, \forall k \in R_j$, $C_j \supset C_k$; by Theorem 3, the roots R_j can be pruned at node i.

Lemma 1. Given any knot $K_i^j(m)$, if the roots $K = K_i^j(m) \cap \overline{\{i\}}$ are pruned at node i, then $C_i \supseteq K_i^j(m)$.

Proof: From (16ab), all but node i of the nodes in $K_i^j(m)$ feed node i, i.e., $K \subseteq R_i$. [The equality holds only when $j = 2^n - 1$.] A node $r \notin K$ that feeds node i has no roots from nodes in $K_i^j(m)$, i.e., $R_r \cap K_i^j(m) = \phi$. Since node r feeds node i, $i \notin R_r$. If node i + k (k > 0) in K fed node r, then so would node j, since node j feeds node i + k and because $R_{i+k} \subset R_r$, by Property 2. Consequently, from (8),

$$\beta_j \wedge \beta_r = \beta_r \text{ and } \beta_r \wedge \beta_i = \beta_i,$$

which along with (16a) and the fact that

$$\beta_j \wedge \beta_i = \beta_i$$

implies that

$$\beta_{j-i} \wedge \beta_{r-i} = \beta_{r-i},$$

or that $r \in T_i^j(m)$. But by assumption $r \notin T_i^j(m)$, so $R_r \cap K = \phi$ is proved by contradiction. It follows that $C_r \subseteq \{r\} \cup R_r$ is disjoint from $K_i^j(m)$ for each node $r \notin K_i^j(m)$ that feeds node i. Thus, if the roots K are pruned at node i, from (15),

$$C_i = T_i \cap \bigcap_{r \in S_i} \overline{C}_r \supseteq T_i \cap \bigcap_{r \in S_i} \overline{(\{r\} \cup R_r)}$$

$$\supseteq T_i \cap K_i^j(m) = K_i^j(m).$$

Lemma 2. Given any knot $K_i^j(m) \subset F(\overline{F})$ and a knot $K_s^r(m') \subset \overline{F}(F) \ni$ $r \in R_i$ and $s \notin R_i$, $T_s \cap K_i^j(m) = \emptyset$.

Proof: From (2), (8) and (9), the β for every element in T_s has 1's in all components where β_s has 1's. For T_s and $K_i^j(m)$ to have at least one element in common, the β for some element in $K_i^j(m)$ must have 1's in all components where β_s has 1's. Since β_j has more 1's than the β for any other element in $K_i^j(m)$ from (16ab), it is assumed that $j \in R_s$, which must hold if $T_s \cap K_i^j(m) \neq \emptyset$. If $\bigvee$ is taken as componentwise logical addition for n-tuples, β_r and β_j both have 1's in all components where $\beta_t = \beta_i \bigvee \beta_s$ has 1's ($\beta_t \neq \beta_i$, β_s, since $i \neq s$) because $r \in R_i$, $r \in R_s$, $j \in R_i$ and $j \in R_s$ (by assumption). Consequently, presuming that $t \neq r$, j, nodes r and j must feed a common node t that is in both $K_s^r(m')$ and $K_i^j(m)$, i.e., r, $j \in R_t$ with $t \in K_s^r(m')$, $K_i^j(m)$. Since these knots are disjoint by definition, $j \notin R_s$. If $t = r$ or if $t = j$, then node j must feed node r or vice-versa i.e., $j \in R_r$ or $r \in R_j$, respectively. From the proof of Lemma 1 this is impossible since $r \notin K_i^j(m)$ and $r \notin K_s^r(m')$. Again, by contradiction, $j \notin R_s$.

Theorem 4. Given any knot $K_i^j(m)$, all nodes in $K = K_i^j(m) \cap \overline{\{i\}}$ but stumps can be pruned provided node i is saved.

Proof: Lemma 1 permits node i to cover for all nodes in $K_i^j(m)$ independent of nodes in K. Lemma 2 guarantees Lemma 1 even if a tree graft, viz., root $s \in \overline{F}(F)$ supplanting root $r \in \overline{F}(F)$ at node $i \in F(\overline{F})$, is applied at node i. Consequently, all nodes in K but stumps can be pruned if node i becomes a stump.

Lemma 3. Given any nest $N_i^j(m)$ with $i \in F(\overline{F})$, if the nodes in $N = N_i^j(m) \cap \overline{\{i\}}$ are saved and if at node $i + k \in N_i^j(m)$ all roots in N but those from nodes on level $\ell_{i+k}+1$ are pruned, then $C_i \supseteq N_i^j(m) \cap F(\overline{F})$ and $C_i \cap N_i^j(m) \cap \overline{F}(F) = \emptyset$.

Proof: By an argument similar to that in the proof of Lemma 1, using (16ac), for a node $r \notin N$ which feeds any node in $N_i^j(m)$, C_r is disjoint from $N_i^j(m)$. With $K_s^r(m') \cap N_i^j(m) = \emptyset$ a modified Lemma 2 guarantees that C_s is disjoint from $N_i^j(m)$ if root s of $K_s^r(m')$ supplants root r by means of a tree graft. Thus, no elements of $N_i^j(m)$ can be inhibited from the cover of any node in $N_i^j(m)$ by roots from nodes not in N, i.e.,

$$\bigcap_{r \in S_{i+k} \cap \overline{N}} \overline{C}_r \supset N_i^j(m), \forall\, i+k \in N_i^j(m). \tag{18a}$$

However, nodes in N collectively inhibit the elements $\{i+k\} \subseteq N$ for ℓ_k odd from C_i and guarantee the inclusion of $\{i+k\} \subseteq N$ for ℓ_k even in C_i, if all roots in N at node $i+k \in N_i^j(m)$ but those from nodes on the next higher level are pruned, where ℓ_k is defined by (2), (10) and (16a). This is now shown by induction on decreasing tree level from ℓ_j.

For $\ell = \ell_j$, since $C_j \supseteq \{j\}$ and $R_j \cap N = \phi$, it follows that

$$C_j \cap N_i^j(m) = \{j\} \bigcup_{\substack{\Delta\ell \\ \text{even}}} (R_\ell^j \cap N), \tag{18b}$$

where $\Delta\ell = \ell - \ell_{i+k} > 0$ and R_ℓ^{i+k} is defined with (11). Assume that for $\ell = \ell_{i+k}$,

$$C_{i+k} \cap N_i^j(m) = \{i+k\} \bigcup_{\substack{\Delta\ell \\ \text{even}}} (R_\ell^{i+k} \cap N). \tag{18c}$$

With all roots in N at node $s \in N_i^j(m) \cap \{\overline{j}\}$ pruned except those roots in N from nodes on level $\ell_s + 1$ (see (18e)), from (15),

$$C_s \cap N_i^j(m) = T_s \bigcap_{r \in S_s \cap \overline{N}} \overline{C}_r \bigcap_{r \in S_s \cap N} \overline{C}_r \cap N_i^j(m), \tag{18d}$$

where

$$S_s \cap N = \{i+k\} \subset N \cap R_s \ni \ell_{i+k} = \ell_s + 1. \tag{18e}$$

Using (18a), (18d) becomes

$$C_s \cap N_i^j(m) = \underset{i+k \in S_s \cap N}{T_s} \cap \overline{C}_{i+k} \cap N_i^j(m). \qquad (18f)$$

Since

$$\overline{C}_{i+k} \cap N_i^j(m) = (\overline{C}_{i+k} \cup \overline{N_i^j(m)}) \cap N_i^j(m),$$

using (18c), (18f) can be rewritten as

$$\begin{aligned} C_s \cap N_i^j(m) &= \underset{i+k \in S_s \cap N}{T_s} \cap \overline{[\ \{i+k\} \underset{\substack{\Delta \ell \\ \text{even}}}{\cup} (R_\ell^{\,i+k} \cap N)\,]} \cap N_i^j(m) \\ &= T_s \cap \overline{[\ \underset{i+k \in S_s \cap N}{\cup} \{i+k\} \underset{\substack{\Delta \ell \\ \text{even}}}{\cup} (R_\ell^{\,i+k} \cap N)]} \cap N_i^j(m) \\ &= T_s \cap \overline{[\ \underset{\substack{\Delta \ell \\ \text{odd}}}{\cup} (R_\ell^{\,s} \cap N)]} \cap N_i^j(m) \\ &= T_s \cap [\ \{s\} \underset{\substack{\Delta \ell \\ \text{even}}}{\cup} (R_\ell^{\,s} \cap N)] \\ &= \{s\} \underset{\substack{\Delta \ell \\ \text{even}}}{\cup} (R_\ell^{\,s} \cap N), \end{aligned} \qquad (18g)$$

with the application of De Morgan's Theorem and Property 2.

Since (18bg) have the same form as (18c), this nearly completes the proof. It merely remains to note from (16c) and (18g) that when $s = i \in F(\overline{F})$, $C_i \supseteq N_i^j(m) \cap F(\overline{F})$ and $C_i \cap N_i^j(m) \cap \overline{F}(F) = \emptyset$.

Lemma 4. Given any nest $N_i^j(m)$ with $i \in F(\overline{F})$ and a variable $x_k \in X_j \cap \overline{X}_i$, $C_{i+2^k} \supseteq N_{i+2^k}^j(m-1) \cap \overline{F}(F)$, $C_{i+2^k} \cap N_{i+2^k}^j(m-1) \cap F(\overline{F}) = \emptyset$, $C_i \supseteq N_i^{j-2^k}(m-1) \cap F(\overline{F})$ and $C_i \cap N_i^{j-2^k}(m-1) \cap \overline{F}(F) = \emptyset$ all can hold if node $i+2^k$ has no roots in $N_{i+2^k}^j(m-1)$ but has tree graft roots from all nodes in $N_i^{j-2^k}(m-1)$ on n-tree level ℓ_i+1, and if each node in $N_i^{j-2^k}(m-1)$ has roots from all nodes in $N_i^{j-2^k}(m-1)$ on the next higher n-tree level but no other nodes in $N_i^{j-2^k}(m-1)$ and no roots in $N_{i+2^k}^j(m-1)$.

Proof: As can be verified using (16ac), the nest $N_i^j(m)$ is composed of two distinct nests, $N_{i+2^k}^j(m-1)$ and $N_i^{j-2^k}(m-1)$, that are uniquely determined by x_k, which can be any branch that appears at node j but not node i. The nodes $N_{i+2^k}^j(m-1)$ and $N_i^{j-2^k}(m-1)$ have $\{x_k\} \cup X_i$ and X_i as common branches, respectively. From (2), (7) and (8), T_{2^k} is the set of all n-tree nodes that have x_k as a branch. Let $v(\overline{v})$ be the logical value of an input to a node such that the output of that node is not influenced (is forced to v) by that input.

If $x_k = v$, all x_k branches in the n-tree can be ignored. This is equivalent to removing all x_k branches and subtracting 2^k from all nodes and roots in T_{2^k} of the n-tree. But in this event an original node $r \in T_{2^k}$ becomes identical to n-tree node $r-2^k$ with roots in T_{2^k} removed. Furthermore, since n-tree node $r-2^k$ now has all the branches (as well as roots) that appear at the new node $r-2^k$, and because the original node r feeds n-tree node $r-2^k$, the output of n-tree node $r-2^k$ is forced to v by the new node $r-2^k$. This follows from Property 1a where all the explicit inputs to the new node $r-2^k$ are redundant with respect to n-tree node $r-2^k$. Therefore, only the influence of the implicit input v remains at the new node which thereby effectively supplies $\overline{v}$ as an input to n-tree node $r-2^k$. Hence, all n-tree nodes and roots not in T_{2^k} can also be removed.

If $x_k = \overline{v}$, all roots in T_{2^k} have logical value v and can be ignored. This is equivalent to removing all nodes and roots in T_{2^k} from the n-tree.

Let the roots of nodes in $N_i^{j-2^k}(m-1)$ be constrained as stated in Lemma 4. Similarly, let each node in $N_{i+2^k}^j(m-1)$ have roots from all nodes in $N_{i+2^k}^j(m-1)$ on the next higher n-tree level but no other nodes in $N_{i+2^k}^j(m-1)$. Then by Lemma 3, all the conditions on C_{i+2^k} and C_i stated in Lemma 4 hold. The conditions on C_{i+2^k} are unaffected if node $r \neq i+2^k$ in $N_{i+2^k}^j(m-1)$ is replaced by node $r-2^k \neq i$ in $N_i^{j-2^k}(m-1)$. This can be verified using the previous discussion as follows. If $x_k = v$, each root $s \notin N_{i+2^k}^j(m-1)$ at node r corresponds to the pair of equivalent roots s and $s-2^k \notin N_i^{j-2^k}(m-1)$ at node $r-2^k$; root $s \in N_{i+2^k}^j(m-1)$ at node r corresponds to root $s-2^k \in N_i^{j-2^k}(m-1)$ at node $r-2^k$. The absence of roots in $N_{i+2^k}^j(m-1)$ at node $r-2^k$ prevents the redundancy of node $r-2^k$ when $x_k = v$. If $x_k = \overline{v}$, the output of node $i+2^k$ is forced to v regardless of the other inputs at node $i+2^k$. Consequently, the conditions on C_{i+2^k} remain satisfied if the roots in $N_{i+2^k}^j(m-1)$ on level $\ell_{i+2^k}+1$ are supplanted by the roots in $N_i^{j-2^k}(m-1)$ on level ℓ_i+1 by means of tree grafts.

Lemma 5. Given the conditions of Lemma 4, (a) $C_i(C_{i+2^k})$ also includes all elements in $N_i^j(m) \cap F$ only covered by stumps in $N_i^{j-2^k}(m-1)$ $(N_{i+2^k}^j(m-1)) \cap F$ if none of these elements are included in any of the covers of nodes in $N_i^j(m) \cap \overline{F}$ which feed a node in $N_i^{j-2^k}(m-1) \cap F$ (node $i+2^k \in F$), and (b) for a node $s \in N_i^j(m) \cap O$ that node $i+2^k(i) \in \overline{F}$ feeds, C_s includes all elements in F only covered by stumps in $N_{i+2^k}^j(m-1)(N_i^{j-2^k}(m-1)) \cap F$ if none of these elements are included in any of the covers of nodes in $\overline{F}$ which also feed node s, and if no nodes in $N_i^j(m) \cap F$ feed node s.

Proof: In (a), consider a stump on level ℓ in $N_i^{j-2^k}(m-1) \cap F$ which covers a set of elements in $N_i^j(m) \cap F$ not covered by nodes disjoint from $N_i^j(m)$. By definition of the roots in Lemma 4, these elements are inhibited from the cover of each node in $N_i^{j-2^k}(m-1)$ on level $\ell - 1$, regardless of whether unspecified roots at these nodes are present. Continuing to level $\ell - 2$ the cover of each node in $N_i^{j-2^k}(m-1)$ includes the elements in question because neither the cover of a node in $\overline{F} \cap N_i^j(m)$ nor in $F \cap N_i^j(m)$ feeding such a node covers any of the elements, by assumption. The inhibition and covering of these elements continues to alternate as the level is decreased. By induction on decreasing the level to ℓ_i, C_i includes these elements. The remainder of the lemma can be shown in a similar fashion. Note that from (17a), s may be a node of a tree growth.

Lemma 6. Given the conditions of Lemmas 4 and 5, all roots at nodes in $N_i^{j-2^k}(m-1) \cap \{\overline{i}\}$ and at node $i+2^k$ (at nodes in $N_i^{j-2^k}(m-1)$) but those roots specified in Lemma 4 as being present can be pruned when $i \in F(\overline{F})$ provided any inhibiting required at nodes in $N_i^{j-2^k}(m-1)$ $(N^j_{i+2^k}(m-1)) \cap F$ is accomplished at nodes $i(i+2^k)$ and s,* and if nodes in $N_i^j(m) \cap F$ do not feed node s.

Proof: If $i \in F(\overline{F})$, any inhibiting required at nodes in $N_i^{j-2^k}(m-1)$ $(N^j_{i+2^k}(m-1)) \cap F$, including that mentioned explicitly in Lemma 5a, can just as well be accomplished at node $i(i+2^k)$ and node s,* by Property 2; if s is a node of a tree growth, the appropriate roots must be created at node s since Property 2 does not apply for tree growths. All but the specified roots at node $i+2^k(i) \in \overline{F}$ can be pruned because this can only decrease the number of elements in C_s without deleting elements that are guaranteed by Lemma 5b. The last clause of Lemma 6 is necessary when s is not a node of a tree growth to insure that C_s includes all elements in F covered only by nodes in $N^j_{i+2^k}(m-1)$ $(N_i^{j-2^k}(m-1)) \cap F$.

Theorem 5. Given any nest $N_i^j(m)$ with $i \in F(\overline{F})$ and a variable $x_k \in X_j \cap \overline{X_i}$, all nodes in $N^j_{i+2^k}(m-1)$ but node $i+2^k$ can be pruned if the roots of node $i+2^k$ and the nodes of $N_i^{j-2^k}(m-1)$ are as stated in Lemma 6 [except that if $i \in F$, node $i+2^k$ feeds node i], if all roots in $N_i^{j-2^k}(m-1) \cap F$ [but root i, if $i \in F$] are pruned from the base, and if the covers of nodes in $N_i^j(m) \cap \overline{F}$ are constrained as stated in Lemma 5 and applied as in Lemma 6 [should node s of Lemma 5b not exist, a tree growth is created which feeds only the base and is fed by $x_k(\overline{x_k})$ and node $i+2^k(i)$].

Proof: If $i \in F$, node i is fed by node $i+2^k$ to prevent $N^j_{i+2^k}(m-1) \cap \overline{F}$ from appearing in the network output, as it would otherwise since then nodes

*If $i \in F(\overline{F})$, any inhibiting required at nodes in $N^j_{i+2^k}(m-1)$ $(N_i^{j-2^k}(m-1)) \cap F$ is accomplished at node s.

$i+2^k$ and i are equivalent when $x_k = v$ from the proof of Lemma 4; if $i \in \overline{F}$, node i need not be fed by node $i+2^k$ because node i feeds only nodes not in T_{2^k} whose outputs are redundant anyway when $x_k = v$. For the same reason roots in $N_i^{j-2^k}(m-1) \cap F$ are pruned from the base. Lemmas 4, 5 and 6 permit <u>nodes</u> $i+2^k$ and i to properly cover all elements in $N_i^j(m)$ and those in $N_i^j(m) \cap F$ only covered by stumps in $N_i^j(m) \cap F$ without requiring the presence of nodes in $N_{i+2^k}^j(m-1) \cap \overline{\{i+2^k\}}$. Thus, the latter set of nodes can be pruned. The tree growth provision of the theorem merely offers a means of supplying the network output with a required cover when node s does not exist; note that another extra node is required to generate $\overline{x_k}$ from x_k when $i \in \overline{F}$.

<u>Theorem 6.</u> If $0 \in \overline{F}$, all nodes (not used in tree grafts or growths) in any knot $K_0^j(m) \subset \overline{F}$ can be pruned.

Proof: No node $k \in K_0^j(m)$ can feed the base, from (16ab) and Property 3. Barring stumps in $K_0^j(m)$ used in tree grafts or growths (see Theorems 4 and 5), node k cannot feed any node not in $K_0^j(m)$. From (8), every n-tree node r feeds node 0, i.e., $r \in R_0 \forall r \neq 0$. From the proof of Lemma 1, an n-tree node $r \notin K_0^j(m)$ that feeds node 0 has no roots in $K_0^j(m)$. Thus, an assumption that k feeds r is disproved by contradiction. Since node k feeds nowhere but other nodes in $K_0^j(m)$, it can be pruned.

V. A SYNTHESIS PROCEDURE

The <u>cost</u> of the n-tree synthesis of F is taken as the triple

$$\$ = (\eta, \rho, \lambda), \tag{18}$$

where η, ρ and λ are integers specifying the total number of nodes, inputs and levels, respectively, in the final network realization. In the pruning method suggested here an attempt is made to minimize η before ρ or λ, a bias which reflects an assumed greater cost for nodes than inputs or levels. Invoked in the following procedure are Theorems 1, 2, 3 and 5 for pruning inputs and Theorems 4, 5, and 6 for pruning nodes.

<u>Step 1 (Th. 4)</u>

(a) Setting $\ell = n$ and $j = 2^n-1$, using (16ab), find all the knots $\{K_i^{2^n-1}(m)\}$ of maximum size and select one of these arbitrarily. If $m \geq 1$, all nodes in $K_i^{2^n-1}(m) \cap \{\bar{i}\}$ of the chosen knot are pruned and stump i is saved. If $m = 0$, node 2^n-1 is saved. Decrease ℓ to n-1; if $\ell = 0$, go to Step 3.

(b) Given any level $0 < \ell < n$ and a set $\{j\}$ of remaining nodes on level ℓ that are not stumps, prune all nodes in $\{j\} \ni j \in K_i^j(m)$ for some stump i. Next, using (14), for every node $j \in F_\ell(\overline{F}_\ell)$ that feeds only one remaining node $i \in F_{\ell-1}(\overline{F}_{\ell-1})$, prune node j and save stump i; then prune any other node in $\{j\}$ that forms a one-variable knot with any one of these new stumps. Each node $j \in F_\ell(\overline{F}_\ell)$ that feeds no remaining node $i \in F_{\ell-1}(\overline{F}_{\ell-1})$ is saved. Using (16ab), find all the knots $\{K_i^j(m)\}$ of maximum size among remaining nodes for every other node in $\{j\}$. For a given j, if $m \geq 2$ prune node j if $j \in K_i^j(m)$ for some stump i; otherwise, select one of these knots arbitrarily, prune node j and save stump i. Finally, if $m = 1$, prune node j and save stump i that is not included in any of the chosen knots for $m \geq 2$, if possible; also, the node $i \in F_{\ell-1}(\overline{F}_{\ell-1})$ fed by the most such nodes $j \in F_\ell(\overline{F}_\ell)$ is preferred.

(c) Decrease ℓ by one and repeat (b) until $\ell = 0$.

When there is more than one knot $K_i^j(m)$ of maximum size for a given j, a refined choice rather than an arbitrary selection of i sometimes leads to a cheaper realization. The simplest refinement is to select an i which is not part of any knot $K_r^i(m)$ for $m \geq 1$, since in this event node i must be saved anyway; if no such i exists, then an arbitrary selection is made. Extensions of this principle below tree level ℓ_i are straightforward but may become quite involved when there are many options. Instead of attempting to abstractly describe a more general refinement, one is illustrated in subsequent examples. As an alternate to complex refinement, Step 1 can be iterated over all possible choices in order to prune the most n-tree nodes. The appropriate mixture of refinement and repetition is more a question of algorithm efficiency than network minimality.

<u>Step 2 (Th. 5)</u>

(a) If $n < 3$, omit Step 2. Otherwise, using (16ac), among only the saved nodes find all the nests $\{N_i^j(m)\}$ with $m \geq 2$ that are not properly included as subsets of larger nests. Arbitrarily select one of these nests of maximum size. If there are no such nests, then go to Step 3.

(b) Given $N_i^j(m)$ with node j on level ℓ and $i \in F(\overline{F})$, determine if any of the m nodes in $N_i^j(m)$ on level ℓ-m+1 (if node i) feeds a saved node $s \in F$; if such a node s exists, all nodes in $N^j_{i+2^k}(m-1) \cap \{\overline{i+2^k}\}$ are pruned, where $x_k \in X_j \cap \overline{X}_i$ and k is selected in accordance with any of the nodes in $N_i^j(m)$ on level ℓ-m+1 which feed node s (is selected arbitrarily). If such a node s does not exist and m = 2, then nodes $N^j_{i+2^k}(m-1) \cap \{\overline{i+2^k}\}$ are still saved. If s does not exist, $m \geq 3(4)$ and $i \in F(\overline{F})$, then all nodes in $N^j_{i+2^k}(m-1) \cap \{\overline{i+2^k}\}$ are pruned, and a tree growth is created consisting of a node feeding the base and fed by $x_k(\overline{x}_k)$ and node $i+2^k(i)$, where k is selected arbitrarily. In the remaining case for m = 3 and $i \in \overline{F}$, if i = 0 or if node i is not a stump, then $N_i^j(3)$ is reduced to three two-variable nests $\{N^j_r(2)\}$, any one of which is treated exactly as in the previous case for m = 2, $r \in F$ and node s existing; if $i \neq 0$ and node i is a stump, then the three nodes in $N^j_{i+2^k}(2) \cap \{\overline{i+2^k}\}$ are pruned, and a tree growth fed by $\overline{x}_k$ and node i is created.

(c) Repeat (b) for the next larger nest found in (a) not already considered if that nest is disjoint from the nodes $N_i^{j-2^k}(m-1)$, $i+2^k$ and s of any other nest where nodes were pruned. When all nests found in (a) have been considered, go to Step 3.

As in Step 1, when there is more than one nest $N_i^j(m)$ of maximum size, a refined choice rather than an arbitrary selection may lead to a cheaper realization. The simplest refinement of Step 2 is to select a nest for which a saved node s exists; if there is no such nest, then an arbitrary selection is made. Since the expected number of options in Step 2 is less than in Step 1, further refinement is less preferred than iterating Step 2 over all possible choices in order to prune the most saved n-tree nodes.

The various cases of Step 2b result from the fact that $2^{m-1}-1$ nodes can be pruned from $N_i^j(m)$ but a tree growth requires one (two) additional nodes when $i \in F(\overline{F})$. Also, in the special case when node s does not exist, $m = 3$, $i \in \overline{F}$ and either $i = 0$ or node i is not a stump, the direct application of Theorem 5 sacrifices the opportunity for pruning node i by Theorem 6. This is illustrated in a subsequent example. In short, Theorem 5 is applied only if there is a net gain in the number of nodes pruned.

When $m \geq 4$, a cheaper network may result if all nodes in $N_i^j(m)$ but stumps are replaced by a linear realization of $3m-2$ nodes in the m variables of $X_j \cap \overline{X}_i$. With such a replacement the output node of the linear network is identified with node i when roots in $N_i^j(m)$ are pruned. A general construction technique for such linear networks is described in Appendix A.

<u>Step 3 (Th. 6)</u>

If $0 \in F$, omit Step 3. If $0 \in \overline{F}$ and node 0 does not feed a node of a tree growth, then node 0 is pruned; otherwise, node 0 is saved.

Step 4a (Th. 5)

If nodes are not pruned in Step 2, then go to Step 5. Otherwise, for each nest $N_i^j(m)$ where nodes $N^j_{i+2^k}(m-1) \cap \overline{\{i+2^k\}}$ are pruned, perform the following operations. Prune all roots but those from nodes in $N_i^{j-2^k}(m-1)$ on the next higher level at nodes in $N_i^{j-2^k}(m-1) \cap \{\overline{i}\}$. Prune all roots in $N_i^j(m)$ but those from nodes in $N_i^{j-2^{k^1}}(m-1)$ on level ℓ_i+1 at node i, except root $i+2^k$ if $i \in F$. At node $i+2^k$ prune all roots in $N^j_{i+2^k}(m-1)$ and add roots from all nodes in $N_i^{j-2^k}(m-1)$ on level ℓ_i+1.

Step 4b (Ths. 1,2 and 3)

If $i \in F(\overline{F})$, all roots in $\overline{N_i^j(m)}$ are pruned at node $i+2^k(i)$. At node $i(i+2^k)$ all remaining roots in F are pruned. Using (17a), if $s \in O_t$, all roots in F are pruned at node s. This leaves only roots in $\overline{F}$ remaining at node $i(i+2^k)$ and node s.

For each node $r \in N_i^j(m) \cap F$ but node $i(i+2^k)$, find the union of all knots $\{K_r^t(m')\}$ and form the set $C_{\overline{r}} = \{K_r^t(m')\} \cap \{\overline{r}\}$. If $C_{\overline{r}} = \phi$, define $E_r = \phi$. Otherwise, for each $r \ni C_{\overline{r}} \neq \phi$, determine the elements $E_r \subseteq C_{\overline{r}} \ni$ the knot $K_u^e(m'')$ does not exist for a saved node $u \in \overline{N_i^j(m) \cap F}$, where $e \in C_{\overline{r}}$.

If $i \in F$, using (11) and (15), for every root $w \in R^i_{\ell_i+1} \cap N_i^j(m)$ [there are exactly m such roots], at node i prune each root $z \in C_{\overline{w}}$ if node z is pruned and $z \in C_w \ni C_z \subset C_w$, where C_z is the union of all knots $\{K^t_z(m')\}$ [if z is not a node of another nest where nodes were pruned in Step 2]. Set $\ell = \ell_i+1$ and select any remaining root $w \in R^i_\ell$ at node i. If node w was pruned, root w is replaced by root w' by means of a tree graft, where node w' is the stump saved to permit the pruning of node w. For every $r \ni E_r \cap R_w \neq \phi$ and $r \in N_i^{j-2^k}(m-1)$, node r feeds node w by means of a tree graft. [From the proof of Lemma 1, $r \notin R_w$.] Roots R_w [or $R_{w'}$, if node w was pruned] are pruned at both nodes w [or w'] and i. When all remaining roots in R^i_ℓ are considered, ℓ is increased by one and this process is repeated through $\ell = n$.

If $i \in \overline{F}$, using (11) and (15), for every root $w \in R^{i+2^k}_{\ell_{i+2^k}+1} \cap N^j_i(m)$ [there are exactly $m-1$ such roots], at node $i+2^k$ prune each root $z \in C_{\overline{w}}$ if node z is pruned and $z \in C_w \ni C_z \subset C_w$. Set $\ell = \ell_{i+2^k+1}$ and select any remaining root $w \in R^{i+2^k}_{\ell}$ at node $i+2^k$. Again, if node w was pruned, root w is replaced by root w' from the appropriate stump. For every $r \ni E_r \cap R_w \neq \emptyset$ and $r \in N^j_{i+2^k}(m-1)$, node $r-2^k$ feeds node w by means of a tree graft. Roots R_w [or $R_{w'}$] are pruned at both nodes w[or w'] and $i+2^k$. When all remaining roots in $R^{i+2^k}_{\ell}$ are considered, ℓ is increased by one and this process is repeated through $\ell = n$.

If $s \in O_t$ and if $i \in F(\overline{F})$, at node s prune each root $w \in C_{\overline{i+2^k}}(C_{\overline{i}})$ if node w is pruned and $w \in C_{i+2^k}(C_i) \ni C_w \subset C_{i+2^k}(C_i)$. Set $\ell = \ell_s+1$ and select any remaining root $w \in R^s_{\ell}$ but $i+2^k(i)$. If node w was pruned, root w is replaced by root w' from the appropriate stump. For every $r \ni (\{r\} \cup E_r) \cap R_w \neq \emptyset$ and $r \in N^j_{i+2^k}(m-1)$ $(N^{j-2^k}_i(m-1))$, node $r-2^k(r)$ feeds node w [or w'] by means of a tree graft. Roots R_w [or $R_{w'}$] are pruned at node w [or w']. At node s prune each root $z \in C_{\overline{w}}$ [or $C_{\overline{w'}}$] if node z is pruned and $z \in C_w$ [or $C_{w'}$] $\ni C_z \subset C_w$ [or $C_{w'}$]. When all remaining roots in R^s_{ℓ} are considered, ℓ is increased by one and this process is repeated through $\ell = n$.

Using (17a), if $s \in O_g$ and if $i \in F(\overline{F})$, perform the following operations for each node $z \in N^j_{i+2^k}(m-1)$ $(N^{j-2^k}_i(m-1)$ on level ℓ_{i+2^k+1} (ℓ_i+1) [Only this level need be considered by Property 2.] to determine the inhibiting roots required at node s. Given a node z, define the set $S_z = R_z$ initially. Remove all elements in F and $N^j_i(m)$ from S_z. If node $w \in R^z_{\ell_z+1} \cap N^j_i(m)$ was a stump prior to Step 2, form the set $C_{\overline{w}}$ and eliminate elements in $C_{\overline{w}}$ from S_z. Set $\ell = \ell_z+1$ and select any remaining element $w \in R^z_{\ell}$ at node z. If node w was pruned, element w is replaced by element w' from the appropriate stump. Eliminate elements R_w from S_z. When all remaining elements in R^z_{ℓ} are considered, increase ℓ by one and repeat this process through $\ell = n$.

The final set S_z are nodes that feed node s.

Step 5 (Ths. 1, 2 and 3)

Except for saved nodes $N_i^{j-2^k}(m-1)$, $i+2^k$ and s associated with any nest $N_i^j(m)$ where nodes $N^j_{i+2^k}(m-1) \cap \{\overline{i+2^k}\}$ were pruned in Step 2, perform the following operations for each saved n-tree node z with all the roots R_z still remaining. If $z \in \overline{F}$, prune all roots R_z at node z. If $z \in F$, prune all roots in F at node z. Also, using (11), set $\ell = \ell_z+1$ and select any remaining root $w \in R^z_\ell$. If node w was pruned replace root w by root w' by means of a tree graft, where node w' is the stump saved to permit the pruning of node w. [If w' is one of the nodes in $N^j_{i+2^k}(m-1) \cap \{\overline{i+2^k}\}$ that was pruned in any nest $N^j_i(m)$, then node $w'-2^k$ is used as the stump and root $w'-2^k$ replaces root w at node z.] If node w [or w'] has no roots, prune roots R_w at node z; otherwise prune all roots in C_w [or $C_{w'}$] but w [or w']. When all remaining roots in R^z_ℓ are considered, ℓ is increased by one and this process is repeated through $\ell = n$.

Step 6

(a) The base is fed by remaining nodes in F except nodes in $N_i^{j-2^k}(m-1) \cap \{\overline{i}\}$ of each nest $N_i^j(m)$ where nodes $N^j_{i+2^k}(m-1) \cap \{\overline{i+2^k}\}$ were pruned. At this point it may be possible to further prune the resulting network by somewhat ad hoc techniques.

(b) If node 0 remains and has a single root w, and if node w feeds no other remaining node then both nodes w and 0 are pruned and the remaining inputs at node w replace root 0 at every remaining node where 0 appears. If the base has a single root w and is not fed by a node of a tree growth, and if node w has a single input, then both node w and the base are pruned and the input to node w becomes the network output. [This follows from the fact that a node with a single input merely implements the logical complementation of that input.] If a branch now appears at the base, this branch is pruned at every other remaining node. [This follows from Property 1b.]

(c) If node 0 remains and is fed by nodes with a common input, then that input feeds the base and is pruned elsewhere in the network. [This is easily verified by showing network equivalence for both logical values (v and $\overline{v}$) of the common input].

(d) Branches at nodes on level three can sometimes be pruned by Property 1a utilizing existing complemented variables (or other third-level nodes) where required at nodes on level two; rarely, even a node on level three can be pruned in this fashion. Also, occasionally the network can be simplified by merely creating complemented variables to replace multi-branch nodes on level three.

VI. SPECIFIC RESULTS

Let F be represented as a binary number

$$\alpha = a_{2n-1} \ldots a_i \ldots a_0; \; a_i = \begin{cases} 1 \\ 0 \end{cases} \text{if } i \in \begin{matrix} F \\ \overline{F} \end{matrix} \qquad (19)$$

but written in octal notation for convenience, e.g., for $n = 3$, $F = \{1, 2, 4, 7\}$ is represented as $\alpha = 10010110$ but is written as $(\alpha)_8 = 226$. This example function illustrates the special case of Step 2b of the synthesis procedure where the node s does not exist for the nest $N_0^7(3)$ and $0 \in \overline{F}$. The resulting network realizing 226 is shown in Fig. A-1 as f(3). The complementary function 151 is synthesized as $\overline{f(3)}$ in Fig. A-1. Both networks are obtained using Theorem 5 in Steps 2 and 4 of V and are the cheapest possible realizations, i.e., from (18), \$226 = (7, 20, 4) and \$151 = (7, 16, 5) are minimal in η, ρ and λ.

There are sixty-eight nondegenerate functional equivalence classes of three variables ($n = 3$), where two functions are defined to be equivalent if one becomes identical to the other with any permutation of the true variables. Hellerman[3] lists all possible minimal circuits for $n = 3$ with η minimized first and ρ minimized subject to η being minimum. The synthesis procedure yields the minimum value of η in sixty-three cases; only one more node in excess of the minimum is required in each of the other five cases. In fifty-five of the sixty-eight cases, the synthesis procedure results in a network identical to one of Hellerman's circuits. [He lists more than one minimal circuit for some of the classes.] A cost comparison of the other thirteen cases are listed in Table 1.

Except for the two functions 226 and 151, all three-variable solutions of the synthesis procedure are of three levels or less ($\lambda \leq 3$). The first seven entries of Table 1 show tree solutions requiring the same number of nodes but one more input and one less level than Hellerman's circuit; the eighth entry requires two more inputs. The last five entries correspond to tree solutions that require one more node than the minimum. For the last

Table 1. Cost Comparison of Synthesis Procedure Solutions and Hellerman's Circuits

F	Tree Solution	Hellerman's Circuit
212	(4,7,3)	(4,6,4)
13	(5,8,3)	(5,7,4)
33	(5,9,3)	(5,8,4)
274	(5,11,3)	(5,10,4)
275	(6,12,3)	(6,11,4)
255	(6,11,3)	(6,10,4)
153	(7,15,3)	(7,14,4)
75	(6,12,3)	(6,10,4)
232	(6,13,3)	(5,11,4)
251	(7,13,3)	(6,12,4)
351	(8,16,3)	(7,15,4)
55	(7,13,3)	(6,11,4)
236	(7,17,3)	(6,15,4)

Table 2. Randomly Chosen 4-Variable Examples

F(octal)	$\$ = (\eta, \rho, \lambda)$	Comment
161472	(7,20,4)	$N_8^{14}(2)$, k=2, s=4
151432	(8,21,3)	2; 10,5
45565	(9,23,3)	1; 12,10; 7(x_1, x_0)
134160	(5,12,3)	-; 10,3
136644	(7,18,3)	1; 6
36402	(9,23,3)	4,2; 9,6
121153	(10,25,3)	8,4,2; 10; 7
141732	(8,20,3)	-; 12,10,5
36607	(9,23,3)	4; 9,3;14(x_1)
131457	(8,19,3)	4; 10,6
153651	(11,29,3)	4,2,1; 13,11
10506	(8,20,3)	4; 10,9,3
77624	(8,23,3)	1; 6;-;15
175044	(8,18,3)	8,1; 6
17110	(8,27,3)	-;5;14
175665	(8,18,3)	1;10,6
104535	(8,19,3)	1;12,10,5
22536	(9,29,4)	$N_8^{13}(2)$, k=2, s=4

two entries, Hellerman's circuit results if Step 2 is applied before Step 1 in the synthesis procedure.

The tree solutions are all minimal for $n \leq 2$. For $n \geq 4$, there are no known tabulations of minimal NAND(NOR) circuits with complemented variables unavailable. With no standard of comparison, it is impossible to be precise in evaluating the quality of any synthesis procedure with respect to the relative cost of the resulting solutions. However, reasonable guidelines such as the following one offered by the author can be achieved with experience in the logical design of NAND(NOR) networks.

Conjecture. Given an arbitrary Boolean function $f(n)$ with $n \geq 3$ and complemented variables unavailable, there exists a network of no more than $3n-2$ NAND(NOR) blocks that realizes $f(n)$.

For the 4-variable examples discussed next, this conjecture suggests an upper bound of $\eta \leq 10$ for the minimal realization.

In Table 2 are listed the costs of the tree solutions for some 4-variable functions. Each bit in α of (19) for every F was selected using a source of random digits [8] in an appropriate manner. Theorem 5 was invoked in Steps 2 and 4 of V in only two cases, viz., 161472 and 22536, which yielded a 4-level solution by permitting the pruning of node 14 and 13 of the two-variable nest $N_8^{14}(2)$ and $N_8^{13}(2)$, respectively; $x_k = x_2$ and $s = 4$ was chosen in each case. All the third-level nodes in the tree solutions for the other cases are indicated in the comment column, e.g., for 45565, node 1 from tree level 1 ($\ell = 1$), nodes 12 and 10 from $\ell = 2$ and node 7 from $\ell = 3$ are present with no remaining roots; $7(x_1x_0)$ means that branch x_2 was pruned at node 7. Note that the cheaper solutions, such as that for 134160 with $\eta = 5$ and no third-level nodes from $\ell = 1$, tend to utilize fewer nodes with a single branch, i.e., fewer complemented variables each requiring a single-input node. The synthesis procedure is now followed in detail for two interesting examples in Table 2.

Example 1 (F: 153651) Step 1: Node 15 is pruned and either node 14 or node 7 becomes a stump because a knot $K_i^{15}(m)$ for $m \geq 2$ does not exist. Since both nodes 14 and 7 are in F and feed nodes in F on the next lower tree level, this decision is postponed temporarily. Nodes 13 and 11 are both in $\overline{F}$ and neither feeds a node in $\overline{F}$ on the next lower tree level; consequently, both nodes are saved. Similarly, nodes 5 and 3 in F must be saved; since node 7 feeds nodes 5 and 3, the decision is now made to prune node 7 and designate node 14 as a stump. Nodes 12, 10 and 9 in F each feed a single node in F on the next lower tree level, so all three nodes are pruned and node 8 in F becomes a stump. Node 6 in $\overline{F}$ feeds nodes 4 and 2 in $\overline{F}$ which must be saved, so node 6 is pruned. Finally, node 1 in $\overline{F}$ and node 0 in F must be saved.

Step 2: There are two nests $N_0^5(2)$ and $N_0^3(2)$ of at least two-variables among the saved nodes, but a saved node s does not exist. Since m = 2, this step is complete.

Step 3: This step is by-passed since node 0 is in F.

Step 4: This step is by-passed since nodes were not pruned in Step 2.

Step 5: Prune all roots at nodes 13, 11, 4, 2 and 1 in $\overline{F}$; since $S_i = \emptyset$ for each of these nodes, the cover $C_i = T_i$, from (15). Root 15 in F is pruned at node 14 in F; $C_{14} = \{14, 15\}$. Root 7 in F is pruned at node 5 in F; root 13 in $\overline{F}$ must feed node 5 but by Corollary 2 root 15 is pruned at node 5; $C_5 = \{5, 7\}$. Similarly, roots 7 and 15 are pruned at node 3 and $C_3 = \{3, 7\}$. Roots 9, 10 and 12 in F are pruned at node 8 in F; roots 11 and 13 in $\overline{F}$ must feed node 8 but root 15 is pruned by Corollary 2; root 14 in F is pruned and $C_8 = \{8, 9, 10, 12, 14\}$. Finally, roots 1, 2 and 4 in $\overline{F}$ must feed node 0 in F but all other roots in R_0 but root 8 are pruned by Corollary 2; root 8 in F is pruned and $C_0 = \{0, 8\}$.

Step 6: The base is fed by nodes 0, 8, 3, 5 and 14 in F. From (17), $O_g = \emptyset$, $O_t = \{0, 8, 3, 5, 14\}$ and

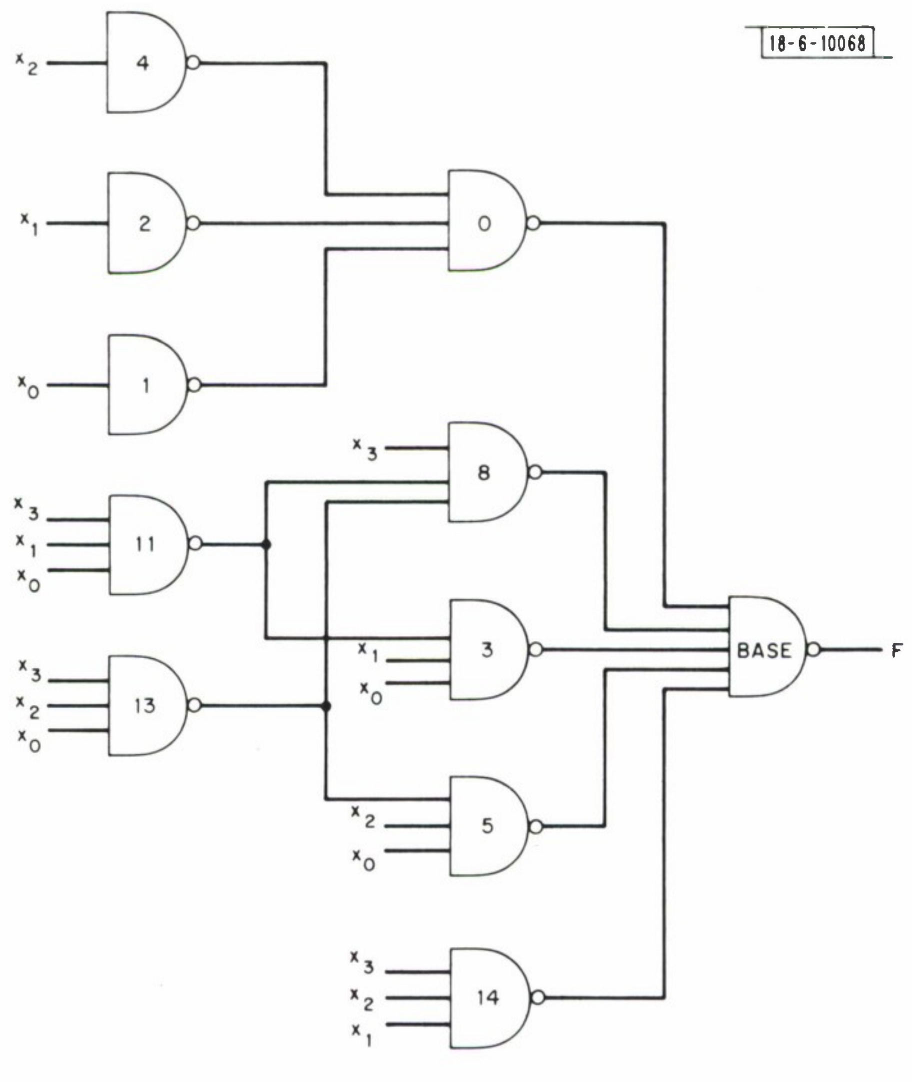
$$\bigcup_O C = \{0, 8, 9, 10, 12, 14, 3, 7, 5, 15\} = F,$$

so the function is realized. Neither node 0 nor the base has a single root and the nodes feeding node 0 have no common input. Third-level roots 13 and 11 are required at node 8, so no branches can be pruned. Thus, no further pruning is possible in this step; \$ = (11, 29, 3).

The tree solution of Example 1 for NAND blocks is shown in Fig. 3a. Since η = 11 exceeds the conjectured upper bound of 10 blocks for 4-variable functions, one hopes that a cheaper but functionally equivalent tree realization exists if a different initial n-tree condition is assumed. Indeed, in this example, if $\overline{F}$ is realized with the n-tree and a node with no other inputs is fed by the base, the total cost is \$ = (10, 29, 4), i.e., one less node is needed at the expense of an additional logic level. For this $\overline{F}$ synthesis the procedure is similar to that for F except that node 14 is pruned in Step 6 as a result of replacing root 14 at node 13 and node 11 by root 3 and 5, respectively, by means of a tree graft. An even cheaper realization is possible here if the same Boolean function is implemented with NOR blocks by synthesizing G with the n-tree, where G is defined by (6) given the F of (1a), i.e., G: 65024 in octal notation. For this G synthesis several tree grafts are performed in Step 5, and in Step 6 when root 1 replaces root 7 at node 14 and root 6 replaces root 7 at node 9 by means of tree grafts, again, it happens that node 7 can be pruned. This time the total cost is only \$ = (8, 23, 3), a reduction of three nodes and six inputs from the first realization! This NOR block solution is shown in Fig. 3b.

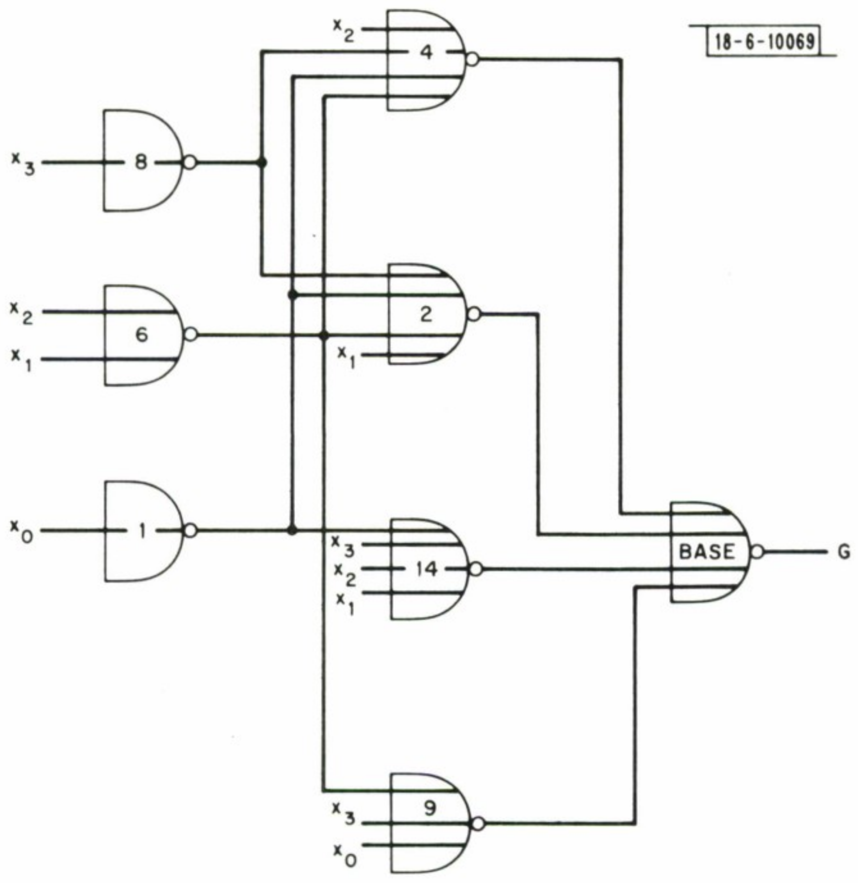
Example 2 (F: 22536) Step 1: Node 15 is pruned and either node 14, node 11 or node 7 becomes a stump. Choosing stump 7 arbitrarily, nodes 14 and 11 are pruned with nodes 12 and 9 becoming stumps. Node 13 in F must be saved since it feeds no node in F on the next lower tree level; similarly, node 5 in $\overline{F}$ must be saved. Nodes 10, 6 and 3 in F are pruned since nodes 8, 4, 2 and 1 in F must be saved. Finally, node 0 in $\overline{F}$ is saved.

Step 2: The largest nest among the saved nodes is $N_0^{13}(3)$. Because a saved node s does not exist and since m = 3 and $0 \epsilon \overline{F}$, $N_0^{13}(3)$ is reduced



(a) NAND BLOCKS

Fig. 3. Tree solution of example 1.



(b) NOR BLOCKS

Fig. 3. Continued.

to three two-variable nests $N_8^{13}(2)$, $N_4^{13}(2)$ and $N_1^{13}(2)$, and $N_8^{13}(2)$ with $x_k = x_2$ and $s = 4$ is selected arbitrarily. Node 13 is pruned.

Step 3: Node 0 in $\overline{F}$ is pruned.

Step 4: All roots are pruned at node 9; $C_9 = \{9, 11, 13, 15\}$. Root 13 in $N_8^{13}(2)$ is pruned at node 8. At node 12 root 13 in $N_{12}^{13}(1)$ is pruned and root 9 in $N_8^9(1)$ is added; $C_{12} = \{12, 14\}$. Roots 14 and 15 in $\overline{N_8^{13}(2)}$ are pruned at node 12. At node 8 in F root 10 in F is pruned. Roots 6 and 13 in F are pruned at node $s = 4$ in F. The set $C_{\overline{13}} = E_{13} = \phi$. For root 9 at node 8, roots 11 and 15 are pruned at node 8 since $C_{\overline{9}} = 11$ and node 11 was pruned and because $C_{15} = \{15\} \subset C_9$. For root 12 at node 8, root 14 is pruned at node 8 since $C_{\overline{12}} = 14$ and node 14 was pruned. All roots R_8 have now been considered; only roots 9 and 12 at node 8 remain; $C_8 = \{8, 10\}$. Similarly, for root $i+2^k = 12$ at node 4, root 14 is pruned at node 4. Root 5 in $\overline{F}$ must feed node 4. Since $(\{13\} \cup E_{13}) \cap R_5 \neq \phi$ and $13 \in N_{12}^{13}(1)$, node 9 feeds node 5. Roots $R_5 = \{7, 13, 15\}$ are pruned at node 5. Root $7 = C_{\overline{5}} \in C_5 = \{5, 7\}$ cannot be pruned at node 4, since node 7 is saved and $C_7 = \{7, 15\}$ is not properly included in C_5, but root 15 is pruned at node 4 by Corollary 2; $C_4 = \{4, 6, 13\}$.

Step 5: Prune root 15 at node 7 in $\overline{F}$. Roots 3, 6 and 10 in F are pruned at node 2 in F; root 7 must feed node 2 but root 15 is pruned by Corollary 2; root 11 and root 14 are replaced by root 9 and 12, respectively, by means of a tree graft at node 2; $C_2 = \{2, 3, 6, 10\}$. At node 1 in F, roots 3 and 13 in F are pruned; roots 5 and 9 must feed node 1; root $7 \in C_5$ is pruned at node 1 because of root 5, and roots 11 and 15 are pruned by Corollary 2 because of root 9; $C_1 = \{1, 3\}$.

Step 6: The base is fed by nodes

$$O_t = \{1, 2, 4, 8\} \subset F = \bigcup_O C = \{1, 3, 2, 6, 10, 4, 13, 8\}; \; O_g = \phi.$$

No further pruning is possible in this step; \$ = (9, 29, 4). The tree solution is shown in Fig. 4.

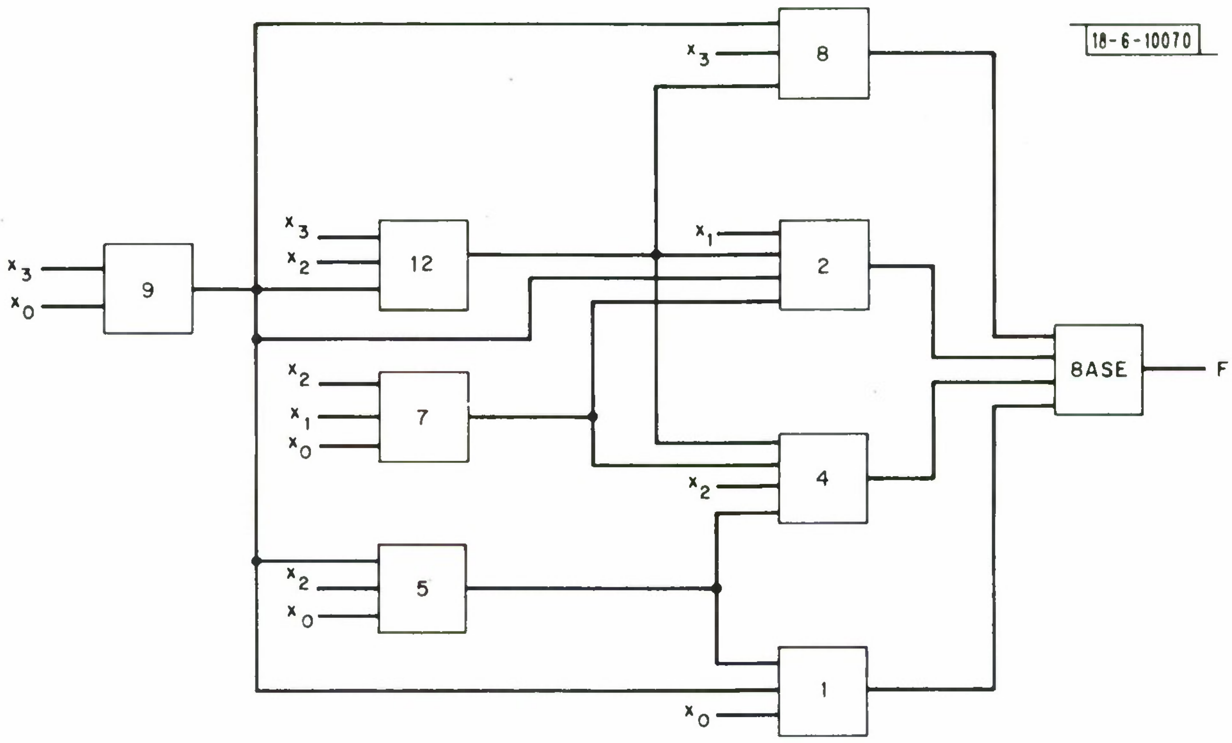


Fig. 4. Tree solution of example 2.

VII. SUMMARY

A synthesis procedure utilizing a few basic theorems for directly pruning a canonic but redundant NAND(NOR) tree network is presented. The utility of regular subtree configurations (knots and nests) for simplying networks is exhibited. Given a Boolean function F, the logic designer has the option of selecting the minimum network resulting from the tree solutions of F, the complementing function $\overline{F}$ and the dual functions G and $\overline{G}$ [with the proper attention to whether NAND or NOR blocks are used and if the output is complemented].

To avoid a significant bias in the choice of example functions, a table of random digits is employed. From the examples it appears that most tree solutions are of three logic levels or less. Three-level networks are obtained faster since major portions of the algorithm are by-passed, but in a broad sense irredundant solutions of more than three levels tend to require fewer logic blocks.

Fan-in and fan-out requirements apparently do not greatly exceed the number of variables. It is conjectured that no more than 3n-2 blocks are needed to realize any n-variable function, for $n \geq 3$.

REFERENCES

1. G. A. Maley and J. Earle, The Logical Design of Transistor Digital Computers, Prentice-Hall Inc., Englewood Cliffs, N. J. (1963).

2. E. J. McCluskey, "Logical Design Theory of NOR Gate Networks With No Complemented Inputs," 1963 Proc. 4th Ann. Symp. on Switching Circuit Theory and Logical Design, 137-148, (1963).

3. L. Hellerman, "A Catalog of Three-Variable OR-INVERT and AND-INVERT Logical Circuits," IEEE Trans. Electronic Computers, EC-12, 198-223, (June 1963).

4. J. F. Gimpel, "The Minimization of TANT Networks," IEEE Trans. Electronic Computers, EC-16, 18-38, (February 1967).

5. D. L. Dietmeyer and Y-H. Su, "Logic Design Automation of Fan-In Limited NAND Networks," IEEE Trans. Electronic Computers, C-18, 11-22, (January 1969).

6. S. V. Novikov, "Analysis of Algorithms for the Synthesis of Logic Networks from OR-NOT Elements," Automation and Computing Technology (Avtomatika i Vychislitel'naya Tekhnika), 18-24, (February 1969).

7. C. R. Baugh, T. Ibaraki, T. K. Liu and S. Muroga, "Optimum Network Design Using NOR and NOR-AND Gates by Integer Programming," IEEE Computer Group Repository, R-69-61, (10 January 1969).

8. A Million Random Digits with 100,000 Normal Deviates, The RAND Corporation, Free Press, Glencoe, Ill. (1955).

APPENDIX A

Linear networks of two and three variables are defined as shown in Fig. A-1. If the nodes are all NAND(NOR) blocks, then $f(2) = x_0 \oplus x_1 (\overline{x_0 \oplus x_1})$ and $f(3) = x_0 \oplus x_1 \oplus x_2$, where $\oplus$ is logical addition modulo two. These networks are the cheapest possible realizations of the given functions.

Since the complement of the function realized by any NAND(NOR) network can be obtained when the output block feeds another NAND(NOR) block, as indicated in Fig. A-2 any m-variable linear network can be constructed using the networks of Fig. A-1. For example, referring to Fig. A-2a, $\overline{f(4)} = \overline{x_0 \oplus x_1 \oplus x_2 \oplus x_3} \; (x_0 \oplus x_1 \oplus x_2 \oplus x_3)$ is realized with ten nodes by feeding the output of the f(3) network in Fig. A-1 into another node to obtain $\overline{f(3)}$, thereby permitting the elimination of the two nodes indicated by dashed lines. The inputs to the node whose output is f(3) in Fig. A-2a can just as well feed everywhere the output $\overline{f(3)}$ feeds. Using the same principle, f(4) is realized with ten nodes in Fig. A-2b with $f_1 = \overline{f(2)}$ and $f_2 = \overline{f'(2)}$ and using the f(2) network of Fig. A-1, where the prime merely emphasizes that f'(2) involves variables disjoint from those of f(2).

In general for $m > 4$, f(m) is realized as shown in Fig. A-2b with $f_1 = \overline{f(\frac{m}{2})}$ and $f_2 = \overline{f'(\frac{m}{2})}$ using networks that realize $f(\frac{m}{2})$ and $f'(\frac{m}{2})$ for m even and with $f_1 = \overline{f(\frac{m+1}{2})}$ and $f_2 = \overline{f'(\frac{m-1}{2})}$ using networks that realize $f(\frac{m+1}{2})$ and $f'(\frac{m-1}{2})$ for m odd; $\overline{f(m)}$ is obtained simply by complementing f_1 and using the corresponding complementary network. It is easily verified that these linear networks require only 3m-2 nodes and are not unique (except possibly $\overline{f(4)}$).

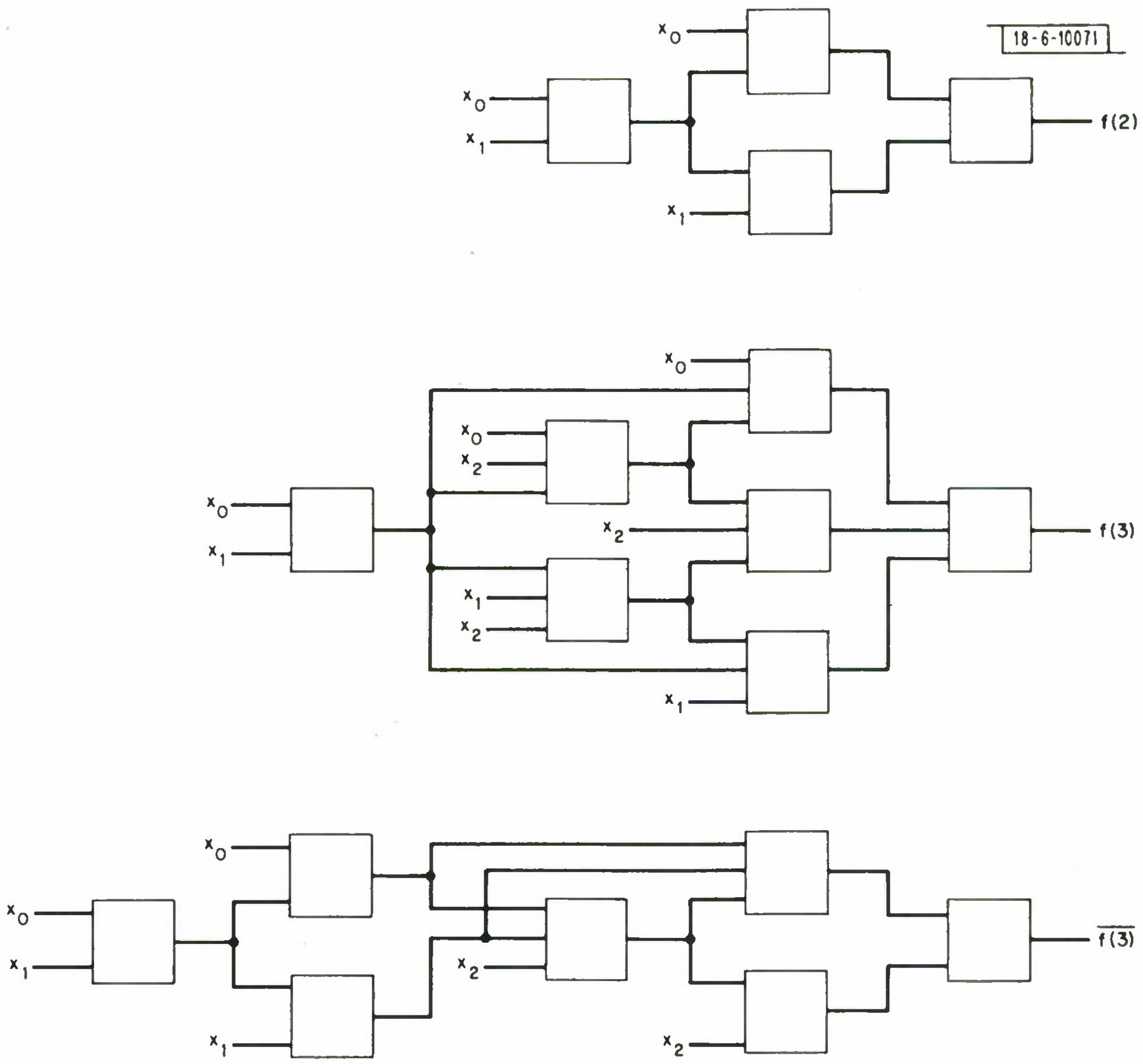
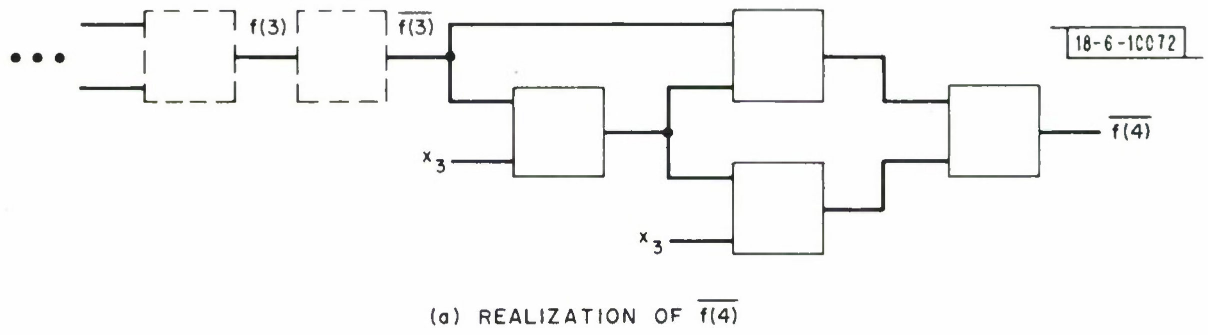


Fig. A-1. Linear two and three-variable networks.



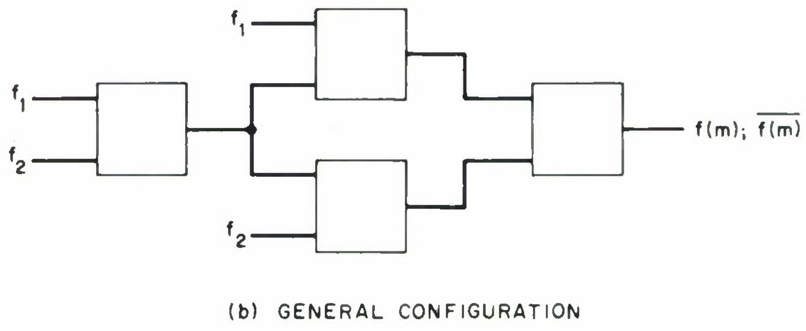


Fig. A-2. Construction of m-variable linear networks.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA – R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Lincoln Laboratory, M.I.T.	2a. REPORT SECURITY CLASSIFICATION Unclassified; 2b. GROUP None
3. REPORT TITLE Efficient Realization of Boolean Functions by Pruning NAND(NOR) Trees	
4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)* Technical Note	
5. AUTHOR(S) *(Last name, first name, initial)* White, Brian E.	
6. REPORT DATE 2 September 1969	7a. TOTAL NO. OF PAGES 48; 7b. NO. OF REFS 8
8a. CONTRACT OR GRANT NO. AF 19(628)-5167; b. PROJECT NO. 1508A; c.; d.	9a. ORIGINATOR'S REPORT NUMBER(S) Technical Note 1969-48; 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* ESD-TR-69-244
10. AVAILABILITY/LIMITATION NOTICES This document has been approved for public release and sale; its distribution is unlimited.	
11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Department of the Navy

13. ABSTRACT

A combinatorial tree structure composed entirely of NAND(NOR) blocks is pruned in a non-exhaustive fashion to yield minimal or near-minimal networks. It is assumed that complemented variables are not available and that there are no fan-in or fan-out limitations. The cost of a network is taken as being primarily determined by the number of logic blocks with the number of inputs and logic levels as secondary factors. The pruning algorithm lends itself to both hand methods and machine computation, although the synthesis procedure has not been programmed.

Of the 68 nondegenerate functional equivalence classes of 3 variables, the minimum number of blocks results in 63 cases; only one more block in excess of the minimum is required in each of the other 5 cases. For 18 randomly selected Boolean functions of 4 variables, the tree solutions yield an average of 8.2 blocks per function with the following distribution:

number of logic blocks	5	6	7	8	9	10	11
number of tree solutions	1	0	2	9	4	1	1 .

It is shown that the linear function $f(n) = x_0 \oplus \ldots \oplus x_n$ of n variables or its complement can be realized with $3n - 2$ NAND(NOR) blocks, for $n \geq 3$.

14. KEY WORDS

Boolean functions	NAND(NOR) logic	algorithms
tree structure	network theory	digital computers

UNCLASSIFIED
Security Classification