Abstract—A generalized DeMorgan’s theorem is presented that can be applied to Boolean equations with any number of complementations. The known form of the DeMorgan’s theorem is a special case of the generalized theorem with one inversion. This new theorem simplifies and generalizes the transformation of Boolean functions to their equivalent sum of product forms.

Two theorems are presented that simplify and generalize the transformation of basic logic equations to NAND and to NOR forms. The NAND and NOR forms can be translated directly into the corresponding schematics for the circuit.

Index Terms—Basic brackets, basic logic equation, Boolean equation transformations, generalized DeMorgan’s theorem, NAND form, NOR form, sum of product form.

I. INTRODUCTION

For the analysis of a given logic circuit, the corresponding sum of product expression is the desirable form for describing the output as a function of the inputs. In general, the major obstacle in obtaining the sum of product form is the necessary repeated application of DeMorgan’s theorem. Consequently, an improved method for the analysis of logic circuits should eliminate the repetitive application of DeMorgan’s theorem. Such a method is shown in Section II.

In most cases, the logic equations are implemented by currently popular NAND or NOR gates. Since the sum of product form cannot be directly interpreted for NAND or NOR gates, transformations have to be applied to bring the equations to forms that can be interpreted for implementation. The available transformation methods lack generality and simplicity [1], [2], [4], [5]. General and simple transformations are developed and shown in Sections III and IV by applying the inverse procedure of Section II.

II. THE GENERALIZED DE MORGAN’S THEOREM

Shannon suggested DeMorgan’s theorem in functional notation [3]:

\[ f(a_1, a_2, \ldots, a_n, +, \cdot)' = f(a'_1, a'_2, \ldots, a'_n, +, \cdot) \]  (1)

This form is incomplete.

Example 1: Let

\[ f = (a_1 + a_2 + a'_3 a'_4)' \]  (2)

Following (1), one may misinterpret and write that

\[ f = (a'_1 a'_2 a_3 + a'_4) \]  (3)

Theorem 1—The Generalized DeMorgan’s Theorem:

\[ f(a_n, +, \cdot, (\ldots)) = f(a'_n, +, \cdot, (\ldots)) \]

where \( n \) equals any number of complementations acting on the variable or operation.

Proof: Existing brackets, (\ldots), have to be retained because a) they retain the appropriate grouping if a product changes to a sum, e.g., \( (a_1 a_2)' a_3 = (a'_1 + a'_2) a_3 \).
and b) they simplify the regrouping if a sum transforms to a product; see proof for (5).

\[ F_n = (F')^{n-2} \]

\[ = F^{n-2} \] (9)

\[ F_n = (F')^{n-4} \]

\[ = F^{n-4} \] (10)

\[ \vdots \]

\[ F_n = \begin{cases} F', & n \text{ even} \\ F', & n \text{ odd} \end{cases} \] (11)

Q.E.D.

The generalized DeMorgan’s theorem can be expressed in algorithmic form.

Algorithm 1: No changes are made on a variable or operation iff the number of nested complementations is even. For an odd number of nested complementations,

\[ a_n \rightarrow a_n', \]

\[ + \rightarrow -, \]

\[ \cdot \rightarrow +. \]

The application of the generalized DeMorgan’s theorem is part of a procedure to obtain the sum of product. This procedure is conveniently done in four steps.

Step 1: A logic expression is obtained from the circuit.

Step 2: Complemented bracket expressions are eliminated by applying the generalized DeMorgan’s theorem.

Step 3: Redundant brackets are eliminated.

Step 4: Basic brackets are eliminated. (Definition of basic brackets follows.)

Any or all of Steps 2–4 may be unnecessary for special cases.

By definition, basic brackets contain a sum that, as a group, is part of a product: e.g., \( (a_1 + a_2a_3) \) \( a_1 + \cdots \). It can be shown that all nonbasic brackets are redundant.

Example 3: The procedure is illustrated by finding the sum of products for the circuit of Fig. 1.

\[ f(A,B,D,K,L,M,P,Q) \]

\[ = RP + (A + B(\bar{S} + T) + D(B(\bar{M}K))) \] (Step 1, basic expression from the circuit.) (12)

\[ = RP + (\bar{A})(\bar{B} + (\bar{S} + T))(D + (B(\bar{M} + K))) \] (Step 2, generalized DeMorgan’s theorem is applied.) (13)

\[ = RP + \bar{A}(\bar{B} + \bar{S} + T)(\bar{D} + B(\bar{M} + K)) \] (Step 3, nonbasic brackets are eliminated.) (14)

\[ = RP + (\bar{A}\bar{B} + \bar{A}\bar{S} + \bar{A}T)(\bar{D} + B\bar{M} + BK) \] (15)

\[ = RP + \bar{A}\bar{B} + \bar{A}\bar{S} + \bar{A}T \]

\[ = \bar{A}\bar{B} + \bar{A}\bar{S} + \bar{A}T + \bar{A}\bar{B} \bar{D} + \bar{A}\bar{S}\bar{B}\bar{M} + \bar{A}\bar{S}\bar{B}K + \bar{A}\bar{T}\bar{D} \]

\[ + \bar{A}\bar{T}\bar{B}\bar{M} + \bar{A}\bar{T}\bar{B}K \]

(Step 4, final change to sum of product.) (16)

III. TRANSFORMATION OF BOOLEAN EQUATIONS TO NAND EXPRESSIONS

Many of today’s logic circuits are implemented with NAND gates. In general, a Boolean equation cannot be directly transferred to a NAND circuit. Existing transformation methods [1], [2], [4], [5] lack generality and/or simplicity. Since a sum of product form can be obtained systematically from a NAND circuit by the procedure shown in Section II, an inverse transformation, as shown in this section, can be applied to obtain a NAND form from a Boolean equation.

Definition: A basic logic equation is any Boolean equation with enclosing brackets and no complemented or redundant brackets.

Definition: A NAND form is a logic equation containing only complemented products.

Example 4:

\[ f(A,B,D,K,L,M,P,Q) \]

\[ = (((M'K)'((PQ)'K)'L)'D)'(B)'A)'). \] (17)

The corresponding circuit is shown as Fig. 2.

Comparing (17) and the circuit, it is evident that each complemented set of brackets represents a NAND gate, and that the literals within a complemented set of brackets are the inputs to the NAND gate. This is utilized for the transformation to NAND form in Theorem 2.

Theorem 2 is applicable to basic logic equations only. It requires enclosing brackets and noncomplemented brackets. Since complemented brackets can be eliminated by (8), Theorem 2 can, if necessary, in conjunction with (8), be applied to any Boolean equation. Redundant brackets cause redundant levels in the implementation and should, therefore, be also excluded.

Theorem 2—Generalized Transformation to NAND Form:

\[ f(a_0,+,(\ldots)) = f((a_0)'(\ldots)(\ldots))'. \] (18)

In algorithmic form, it is expressed as follows.

Algorithm 2: A basic logic equation is transformed to an equivalent NAND form by: a) doubling every bracket; b) replacing each OR operation by an AND operation in inverse brackets; and c) complementing every closing bracket; i.e.,
The double brackets receive a double complementation; therefore, the operations and the groups of variables in double brackets retain their state by the generalized DeMorgan's theorem. The change \( \oplus \rightarrow \) introduces new groups. It retains the even number of sets of nested brackets before and after the inverse brackets and reduces the number of nested brackets around the AND operation to an odd number. Therefore, by the generalized DeMorgan's theorem, the operations and groups of variables ahead of and after the inverse brackets retain their state; and the AND operation in the inverse bracket is equivalent to an OR operation. Q.E.D.

Example 5: Let

\[
f = (\overline{MKD} + PQLD + KLD + B + \bar{A}) \tag{19}
\]

\[
= ((\overline{MKD} + PQLD + KLD + B + \bar{A})) \quad \text{(Step a)}
\]

\[
= ((\overline{MKD})(PQLD)(KLD)(B)(\bar{A})) \quad \text{(Steps b and c)}.
\]

\[
= ((\overline{MK})(PQ)(K)(L)(D)(B)(\bar{A})) \tag{21}
\]

The corresponding NAND circuit can be drawn directly from (21).

Example 6: Grouping of (19) by brackets and transforming it to the NAND form gives

\[
f = ((\overline{MK} + (PQ + K)L)D + B + \bar{A}) \tag{22}
\]

\[
= (((\overline{MK})'((PQ)'(K)'(L)'D)'(B)'(\bar{A})')' \quad \text{(23)}
\]

\[
= (((\overline{MK})((PQ)(K)L)D)(\overline{B})(\bar{A})). \quad \text{(24)}
\]

The circuit for (24) is shown as Fig. 2 with \( (K) \rightarrow K \).

The last example illustrates several points. The transformation can be done in one step, (22) to (23). The form (24) is more convenient for manual work than form (23); form (23) is more convenient for typing or computer work than form (24). Every nested bracket introduces two levels in the circuit. The transformation contains the assumption that the variables are available in the states shown in the original Boolean equation (22). This allows or requires a modification to the NAND form; for example, if the variable \( A \) in (23) is available in either true or inverted form, then \( (\bar{A}) \) in (24) can be changed to \( A \) as expressed in (17). In general, the NAND form as obtained from (18) is modified to a NAND form \( F \), a final NAND form by eliminating any redundant inverters. Redundant inverters are those that can be omitted in the complementations of the equation because the individual variables or group of variables are available in their complemented states, e.g., \( (A)'(B)' \rightarrow (A'B)' \), if \( A \) is available as \( A' \) and \( B \) is available as \( B \).

IV. TRANSFORMATION OF BOOLEAN EQUATIONS TO NOR EXPRESSIONS

Similarly to the NAND transformation, a transformation to a NOR form is possible.

Definition: A NOR form is a logic equation containing only complemented sums.

Example 7:

\[
f = ((A + ((B)' + (C')') + ((D)' + (E + F)''))'). \tag{25}
\]

The corresponding circuit (Fig. 3) for (25) shows that each complemented set of brackets represents a NOR gate.

Theorem 3—The Generalized Transformation to NOR Form:

\[
f(a_1, +, \cdot, a_i\{\ldots + \ldots\})
\]

\[
= f(a_i, +, \cdot, (a_j \ldots \ldots)) \tag{26a}
\]

\[
= f((a_i, +, \cdot, (a_j \ldots \ldots))' \tag{26b}
\]

where

\[
\{a_i\} U \{a_j\} = A
\]

and \( A \) is the set of all variables of the function \( f \). In algorithmic form, we get the following.

Algorithm 3: A basic logic equation is transformed to an equivalent NOR form by omitting all basic brackets, adding brackets around each product, doubling all brackets, replacing every AND operation by an OR operation in inverse brackets, and complementing all closing brackets.

The transition equation (26a) may or may not be used in the transformation. It is included here to show more clearly the general transformation.

Proof: Since the transformation is applied to equations with basic brackets only, each OR operation and each product of the sum within a basic bracket is complemented an even number of times and, by the generalized De-Morgan's theorem, is not changed.

Since each variable of a product is subjected to an even number of complementations, it retains its original state.
Since each and operation is changed to an or operation subjected to an odd number of complementations, it is equivalent to the original and operation. Q.E.D.

Example 8:

\[
f = (A + B\bar{C} + D(E + F)) = (A + (B\bar{C}) + (D|E + F|))
\]

Equation (27b) is the same as (25); its circuit is shown as Fig. 3. The remarks after Example 6 apply in a similar manner to nor transformations.

V. CONCLUSION

A generalized DeMorgan's theorem presented in Section II simplifies considerably the transformation of logic equations with complemented groups to the sum of product form.

The inverse transformations, from basic logic equations to the NAND and NOR forms, are generalized by the theorems given in Sections III and IV.

The theorems can be used for transformations by computers.

ACKNOWLEDGMENT

The author wishes to thank Dr. W. D. Little and A. S. Corbett for numerous suggestions.

A Contextual Postprocessing System for Error Correction Using Binary n-Grams

EDWARD M. RISEMAN, MEMBER, IEEE, AND ALLEN R. HANSON, MEMBER, IEEE

Abstract—The effectiveness of various forms of contextual information in a postprocessing system for detection and correction of errors in words is examined. Various algorithms utilizing context are considered, from a dictionary algorithm which has available the maximum amount of information, to a set of contextual algorithms utilizing positional binary n-gram statistics. The latter information differs from the usual n-gram letter statistics in that the probabilities are position-dependent and each is quantized to 1 or 0, depending upon whether or not it is nonzero. This type of information is extremely compact and the computation for error correction is orders of magnitude less than that required by the dictionary algorithm.

The technique described can allow relatively poor classifiers to become reliable systems by drastically cutting error rates with only modest reject rates. Experimental results are presented on the error, correction, and reject rates that are achievable as a function of the type of contextual information employed, and the size of the data base from which this information is obtained. The most powerful algorithms detect almost all errors and correct between 60 and 95 percent of them. As an example, a set of binary positional trigrams operating on a set of 800 6-letter words can reduce a word error rate of 47 percent to a reject rate of 8.9 percent and an error rate of

REFERENCES


H. Christian Wehrfritz (S'64-M'66) was born in Hetzelsdorf, Germany, on September 14, 1930. He received the B.Sc. degree from the University of Alberta, Edmonton, Alta., Canada, in 1965, and the M.A.Sc. degree from the University of Waterloo, Waterloo, Ont., Canada, in 1970, both in electrical engineering. He did graduate work at Utah State University, Logan, in 1968.

From 1965 to 1966 he was an Electrical Engineer at Canadian General Electric, Toronto, Ont., Canada, where he worked on the installation of electronic control equipment for steel mills. In 1966 he joined Mohawk College of Applied Arts and Technology, Hamilton, Ont., Canada, as a Lecturer of Electrical Engineering. In 1969 he became Supervisor in the Department of Control Systems and Instrumentation at Mohawk College. He specializes in digital controls and computer electronics.

Mr. Wehrfritz is a Registered Professional Engineer in the Province of Ontario, Canada. From 1972 to 1973 he was chairman of the Chapter on Computers and Automation of the Hamilton Section of the IEEE.

Manuscript received October 26, 1972; revised July 5, 1973. This work was supported by the Office of Naval Research under Grant ONR 049-332.

E. M. Riseman is with the Department of Computer and Information Science, University of Massachusetts, Amherst, Mass. 01002.

A. R. Hanson is with the School of Language and Communication, Hampshire College, Amherst, Mass. 01002.