JOURNAL OF APPLIED MATHEMATICS AND DECISION SCIENCES, 9(2), 1??

On the Orthogonalization of Arbitrary Boolean Formulae [†]

RENATO BRUNI bruni@dis.uniroma1.it
Dip. di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Michelangelo Buonarroti 12, Roma 00185, Italy

Abstract. The Orthogonal conjunctive normal form of a Boolean function is a conjunctive normal form in which any two clauses contain at least a pair of complementary literals. Orthogonal disjunctive normal form is defined similarly. Orthogonalization is the process of transforming the normal form of a Boolean function to orthogonal normal form. The problem is of great relevance in several applications, e.g. in the reliability theory. Moreover, such problem is strongly connected with the well-known propositional satisfiability problem. Therefore, important complexity issues are involved. A general procedure for transforming an arbitrary CNF or DNF to an orthogonal one is proposed. Such procedure is tested on randomly generated Boolean formulae.

Keywords: Boolean functions, NP-completeness, Reliability.

1. Introduction

Let $\mathbb{B} = \{0, 1\}$, or, equivalently, $\{True, False\}$. A Boolean function of n Boolean variables $x_i \in \mathbb{B}$ is a function $f(x_1, ..., x_n)$ from the Boolean hypercube $\mathbb{B}^n$ to the Boolean set $\mathbb{B}$. We assume the reader familiar with the basic concepts of Boolean algebra (see e.g. [11], [19]). A Boolean function can be represented in several manners. The most used one is by means of a Boolean (or propositional) formula $\mathcal{F}$ in *conjunctive* (CNF) or *disjunctive* (DNF) normal form. Both normal forms are widely used, the choice often depending on the applicative context. Orthogonal conjunctive normal form (OCNF) is a CNF in which any two clauses contain at least a pair of complementary literals. Orthogonal disjunctive normal form (ODNF) is defined similarly. The orthogonal form is of great relevance in solving several hard problems, e.g. in the reliability theory. One of the fundamental issues in reliability is to compute the probability p that a complex system is in *operating* state (and not in *failed* state, see for instance [3], [4]). The state of the system depends on the state x_i (operating or failed) of its

[†] Requests for reprints should be sent to: Renato Bruni, Dip. di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Michelangelo Buonarroti 12, Roma 00185, Italy

i-th component, for $i = 1, ..., n$. Such relationship is usually described by means of a Boolean function $g(x_1, ..., x_n)$, so that, knowing the state of the components, the state of the system is immediately computable. Since the probabilities p_r of each component to be in operating state are generally known, p also can be computed by using g. However, such computation may be difficult [16]. For systems where g is expressed by a Boolean formula in ODNF, this probability is very easily computed by summing the probabilities associated to all individual terms, since any two terms correspond to pairwise incompatible events. Another important application is in the field of mathematical statistics, where the orthogonality property is needed for assuring independence among statistical variables, and in particular in the analysis of Variance, where it is used to separate the variation inside each group from the variation among the groups themselves [18].

A classical problem is therefore to derive the orthogonal form, or disjoint products form, of a Boolean function. A reliability algorithm based on the computation of the orthogonal form is proposed for instance in [14]. In general, however, computation of the orthogonal forms is a hard problem [5]. One of the few interesting classes of formulae for which this can performed efficiently is the class of *shellable* DNF, introduced by Ball and Provan [2]. It has been recently proved by Boros et al. [5] that every positive Boolean function (i.e. a Boolean function that can be written as a DNF having no negative literals) can be represented by a shellable DNF. However, the complexity of recognizing shellable DNF is not known, and testing the lexicoexchange property (a strengthening of shellability) is NP-complete [5].

A procedure to transform a generic normal form formula into an orthogonal normal form, also briefly called orthogonal form, is here described. Such operation is called *orthogonalization*. The proposed procedure is applicable to both CNF and DNF. Therefore, in Sect. 2, we introduce a unified notation for normal forms, in order to represent both CNF and DNF. A basic procedure to orthogonalize a generic formula is described in Sect. 4. During the above process, the size of the formula tends to exponentially increase. This is not surprising, since in Sect. 3 we show that an NP-complete problem like propositional satisfiability [6], [10], [13] becomes easy for formulae in orthogonal form. (This can be related to a procedure for solving satisfiability by counting the number of possible solutions proposed in [12].) Hence, the NP complexity [9] can be seen as being *absorbed* by the orthogonalization process. Improvements on the above basic procedure, with the aim of minimizing the size of the formula both in the final result and during the computation, are then presented in Sect. 5.

2. Notation and Equivalence of Problems

A Boolean CNF formula is the logic conjunction ($\wedge$) of m *clauses*, each clause being the logic disjunction ($\vee$) of *literals*, each literal being either a positive (x_i) or a negative ($\neg x_i$) Boolean (or propositional) variable. By denoting with $P_j \subseteq \{1, ..., n\}$ the set of positive variables of the j-th clause, and with $N_j \subseteq \{1, ..., n\}$ the set of negative variables of the same clause, this is

$$\bigwedge_{j=1...m} (\bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \neg x_i)$$

Conversely, a Boolean DNF formula is the logic disjunction of m *terms*, each term being the logic conjunction of literals, defined as above. By denoting with $P_j \subseteq \{1, ..., n\}$ the set of positive variables of the j-th term, and with $N_j \subseteq \{1, ..., n\}$ the set of negative variables of same term, this is

$$\bigvee_{j=1...m} (\bigwedge_{i \in P_j} x_i \wedge \bigwedge_{i \in N_j} \neg x_i)$$

The proposed procedure will apply to both CNF and DNF. Therefore, a notation which can represent both forms is needed. Clauses and terms can be viewed as pairs of sets (P_j, N_j) of literals plus a logical operator connecting all such literals. Such structures will be called *monomials*, and denoted by m_j. The Boolean function expressed by a single monomial m_j will be denoted by $m_j(x_1, ..., x_n)$. A CNF or DNF formula $\mathcal{F}$ can now be viewed as a collection of monomials. An *external* operator is applied between monomials, that will be here indicated with the symbol $\perp$, and an *internal* operator is applied between literals of the same monomial, that will be here indicated with the symbol $\top$. Both CNF and DNF are therefore representable as follows.

$$\bot_{j=1...m} (\top_{i \in P_j} x_i \top \top_{i \in N_j} \neg x_i)$$

Clearly, $\perp$ means $\wedge$ when considering CNF, and $\vee$ when considering DNF, and vice versa holds for $\top$. Given a so defined monomial m_j, let the set $T_j \subseteq \mathbb{B}^n$ where m_j has value 1 be the set of *true points* of m_j

$$T_j = \{(x_1, ..., x_n) \in \mathbb{B}^n : m_j(x_1, ..., x_n) = 1\}$$

and the set $F_j \subseteq \mathbb{B}^n$ (the complement of T_j with respect to $\mathbb{B}^n$) where m_j has value 0 be the set of *false points* of m_j

$$F_j = \{(x_1, ..., x_n) \in \mathbb{B}^n : m_j(x_1, ..., x_n) = 0\}$$

Given now a generic Boolean formula $\mathcal{F}$, let the global set of true points be $T = \{(x_1, ..., x_n) \in \mathbb{B}^n : f(x_1, ..., x_n) = 1\}$, and the global set of false points be $F = \{(x_1, ..., x_n) \in \mathbb{B}^n : f(x_1, ..., x_n) = 0\}$. When $\mathcal{F}$ has the structure of normal form (CNF or DNF), the following relations hold:

Lemma 1 *In the case of CNF, the sets T and F are given by:*

$$T = \bigcap_{j=1}^{n} T_j \qquad F = \bigcup_{j=1}^{n} F_j$$

Lemma 2 *In the case of DNF, the sets T and F are given by:*

$$T = \bigcup_{j=1}^{n} T_j \qquad F = \bigcap_{j=1}^{n} F_j$$

Note that T and F are not immediately computable from $\mathcal{F}$, nor their cardinalities are. Besides the fact that their cardinality can be exponential in the number of variables, even expressing such sets in some *compressed* but *usable* form appears hard. In fact, knowing the set T (or equivalently F) would give the solution of an NP-complete problem, namely the *propositional satisfiability* problem (see e.g. [10]). Also, knowing the cardinality $|T|$ (or equivalently $|F|$) would give the solution of the decision version of the propositional satisfiability problem, which is still NP-complete. This theoretically means, moreover, that every problem in NP can be polynomially reduced to the problem of finding this cardinality [9].

On the contrary, the sets F_j for CNF and T_j for DNF are immediately computable and expressible in compressed form (see below). However, in the case of a generic CNF or DNF, such sets are not disjoint, but can overlap each other: it can be $T_j \cap T_k \neq \phi$ or $F_j \cap F_k \neq \phi$ for some $j, k \in \{1 \dots m\}$. Due to the above reason, for finding respectively the cardinalities $|T|$ and $|F|$, it would be necessary to identify respectively all the T_j and all the F_j. Since the number of points in T_j and F_j can be exponential in the number of variables, the approach of identifying all the T_j and all the F_j has exponential worst-case time complexity. This is not surprising. On the other hand, if all the T_j (resp. all the F_j) would be pairwise disjoint sets, in order to find the cardinality $|T|$ (resp. $|F|$) it would suffice to know the cardinalities of the T_j (resp. F_j), and sum them. Such cardinalities are, in fact, trivially computable. In order to proceed with our notation unification, dissimilarities between *true* and *false* sets should be overcome.

Consider again the satisfiability problem. It consists in finding if, in the Boolean hypercube $\mathbb{B}^n$, there is at least one true point for all clauses (for DNF formulae), or at least one false point for all terms (for DNF formulae). Altogether, false points are *bad* for CNF, while true points are *bad* for DNF. We will now call the set of such *bad* points B, with the convention that $B = F$ for CNF, and $B = T$ for DNF. Moreover, every monomial m_j has its set of *bad* points B_j of the Boolean hypercube $\mathbb{B}^n$, with the convention that $B_j = F_j$ for CNF, and $B_j = T_j$ for DNF. (More intuitively, every m_j *forbids* a set of points: in the case of CNF, the j-th clause forbids its F_j, while, in the case of DNF, the j-th term forbids its T_j). Conversely, we will call G the set of *good* points, with the convention that $G = T$ for CNF, and $G = F$ for DNF. Every monomial m_j has therefore his set of *good* points G_j, with $G_j = T_j$ for CNF, and $G_j = F_j$ for DNF. Sets B_j and G_j on $\mathbb{B}^n$ are directly obtainable by the structure of m_j. In the case of CNF, B_j (in *implicit* form) is given by a vector of length n, called *pattern*, having 0 for each variable appearing positive in m_j, 1 for each variable appearing negative in m_j, and $*$ (*don't care*) for each variable not appearing in m_j. Expanding every $*$ with both 0 and 1 gives all the points of B_j explicitly. Clearly, G_j is given by $\mathbb{B}^n \setminus B_j$. In the case of DNF, B_j (in *implicit* form) is given by a pattern having 1 for each variable appearing positive in m_j, 0 for each variable appearing negative in m_j, and $*$ for each variable not appearing in m_j. Explicit expression of all points of B_j and G_j are obtainable as above. Pattern notation can be unified by using symbol '+' for 1 in case of CNF, for 0 in the case of DNF, and symbol '−' for 0 in the case of CNF, for 1 in the case of DNF.

Example 1: Suppose $n = 5$. Given monomial $(x_1 \top \neg x_3 \top x_4)$, the pattern for the set of its bad points is $\{-, *, +, -, *\}$, corresponding to $\{0, *, 1, 0, *\}$ in the case of CNF, to $\{1, *, 0, 1, *\}$ in the case of DNF.

Table 1. Conventions used in the unified notation for CNF and DNF.

form	ext. op.	int. op.	bad pt.	good pt.	pattern	
CNF	$\wedge$	$\vee$	F	T	0	1
DNF	$\vee$	$\wedge$	T	F	1	0
Unified	$\perp$	$\top$	B	G	$-$	$+$

The cardinalities of the above B_j and G_j are easily computable, as follows.

LEMMA 3 *Let n be the number of variables, and $l(m_j)$ be the number of distinct literals appearing in m_j. The cardinalities of the above introduced B_j and G_j are $|B_j| = 2^{n-l(m_j)}$, and $|G_j| = 2^n - |B_j| = 2^n - 2^{n-l(m_j)}$.*

We denote with (ϕ) the empty monomial, i.e. the monomial m_ϕ which is an empty set of literals. According to Lemma 3, $B_{(\phi)} = \mathbb{B}^n$, hence (ϕ) has only *bad* points. Finally, we denote with ϕ the empty formula, i.e. the formula $\mathcal{F}_\phi$ which is an empty set of monomials. By definition, ϕ has only *good* points, so $G_\phi = \mathbb{B}^n$.

3. The Orthogonal form

A Boolean formula (in unified notation) is in *orthogonal* normal form when every pair of monomials m_j and m_k contain at least one Boolean variable x_i (not necessarily the same i for all the couples of indexes (j,k)) as a positive instance (x_i) in one of them (for instance m_j) and as a negative instance $(\neg x_i)$ in the other (for instance m_k).

$$m_j = (\ldots \top x_i \top \ldots), \quad m_k = (\ldots \top \neg x_i \top \ldots) \qquad \forall\, j,k \in \{1\ldots m\}$$

The above situation for m_j and m_k is variously expressed in literature: the above monomials are said to be *orthogonal* [5], or to *clash* [8] on x_i, or to *resolve* [17] on x_i, or also to *hit* [7] on x_i.

THEOREM 1 *A Boolean formula is in orthogonal normal form if and only if the above defined sets B_j are pairwise disjoint.*

The above theorem clearly particularizes for CNF as follows,

$$\begin{array}{rl} F_j \cap F_k = \phi & \forall\, j,k \in \{1\ldots m\} \\ (T_j \cap T_k \text{ can be } \neq \phi & \text{for some } j,k \in \{1\ldots m\}) \end{array}$$

and for DNF as follows.

$$\begin{array}{rl} T_j \cap T_k = \phi & \forall\, j,k \in \{1\ldots m\} \\ (F_j \cap F_k \text{ can be } \neq \phi & \text{for some } j,k \in \{1\ldots m\}) \end{array}$$

Proof: We first prove that orthogonal form $\Rightarrow B_j \cap B_k = \phi \quad \forall\, j,k \in \{1\ldots m\}$. If two monomials m_j and m_k clash on at least one variable x_c, the corresponding B_j and B_k are defined by two patterns which respectively have $-$ and $+$ in at least position c, hence they define two sets B_j and B_k which cannot have any common point. We now prove that $B_j \cap B_k = \phi \ \ \forall\, j,k \in \{1\ldots m\} \Rightarrow$ orthogonal form. Since B_j and B_k are disjoint, the patterns corresponding to them must contain in at least one position c

respectively $+$ and $-$ (or $-$ and $+$). This because any other combination ($+$ and $+$, $+$ and $*$, etc.) would contradict the hypothesis of B_j and B_k disjoint. Therefore, by letting x_c be the variable corresponding to position c, monomials m_j and m_k corresponding to such patterns must both contain x_c and clash on it. Finally, since we assumed that every pair of sets B_j, B_k has empty intersection, every pair of monomials m_j, m_k are orthogonal. $\square$

Since the orthogonal form is a necessary and sufficient condition for having all the B_j pairwise disjoint, it is a condition for trivially solving the problem of computing $|B|$, which implies trivially solving the propositional satisfiability problem, with the above implications on all problems in NP.

Example 2: Suppose we are interested in checking satisfiability of:

$$(\neg x_1 \top \neg x_2 \top x_3 \top x_4 \top x_5) \bot (\neg x_1 \top \neg x_2 \top x_3 \top x_4 \top x_5) \bot (x_2 \top x_3 \top x_4 \top x_5) \bot$$
$$\bot (x_3 \top \neg x_4 \top x_5) \bot (x_3 \top x_4 \top \neg x_5) \bot (x_3 \top \neg x_4 \top \neg x_5) \bot (\neg x_3)$$

In our terms, we need to check whether the global B covers the whole $\mathbb{B}^5$. There are many different and very efficient techniques to solve the satisfiability problem (see for a survey [10]). In practical cases, however, without imposing restrictions on the structure of the formula (Horn, quadratic, etc.) they have worst-case exponential time complexity. On the other hand, computing the above defined sets B_j, and their cardinalities, is straightforward:

$$\begin{array}{rcll}
(\ x_1 \top \neg x_2 \top\ x_3 \top\ x_4 \top\ x_5) & \rightarrow & B_1 = \{-,+,-,-,-\} & |B_1| = 1 \\
(\neg x_1 \top \neg x_2 \top\ x_3 \top\ x_4 \top\ x_5) & \rightarrow & B_2 = \{+,+,-,-,-\} & |B_2| = 1 \\
(\ x_2 \top\ x_3 \top\ x_4 \top\ x_5) & \rightarrow & B_3 = \{*,-,-,-,-\} & |B_3| = 2 \\
(\ x_3 \top \neg x_4 \top\ x_5) & \rightarrow & B_4 = \{*,*,-,+,-\} & |B_4| = 4 \\
(\ x_3 \top\ x_4 \top \neg x_5) & \rightarrow & B_5 = \{*,*,-,-,+\} & |B_5| = 4 \\
(\ x_3 \top \neg x_4 \top \neg x_5) & \rightarrow & B_6 = \{*,*,-,+,+\} & |B_6| = 4 \\
(\neg x_3) & \rightarrow & B_7 = \{*,*,+,*,*\} & |B_7| = 16
\end{array}$$

By computing the union of all the B_j, we have that B actually covers $\mathbb{B}^5$ (see Fig. 1 below, reporting the case of a CNF). Hence, the given formula is unsatisfiable. Since the number of points of such union is exponential (in the worst case) in the number of variables, this procedure has exponential time complexity. On the contrary, one could observe that the formula is orthogonal, hence the B_j are pairwise disjoint. On this basis, trivially, $|B| = |B_1|+|B_2|+|B_3|+|B_4|+|B_5|+|B_6|+|B_7| = 32$. This suffices to say that B covers $\mathbb{B}^5$, whose cardinality is $2^5 = 32$, and so the given formula is unsatisfiable. Altogether, by using the fact that the given formula is in orthogonal form, one can very easily solve the satisfiability problem.

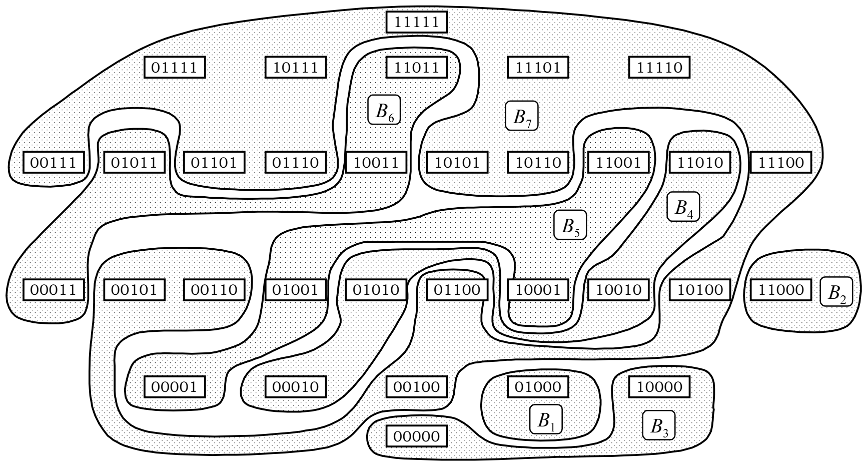

Figure 1. The sets B_j of example 2 on the Boolean hypercube $\mathbb{B}^5$ in the case of a CNF.

4. Basic Orthogonalization Procedure

In order to present a procedure for the orthogonalization of a generic Boolean formula, we first need to define an operation, which will be called *multiplication* and denoted with $\diamond$, applied to a pair of monomials m_j and m_k. The result of such multiplication is a new monomial containing all the literals of m_j and of m_k (but without repeated ones) when the two monomials are not orthogonal, and the empty formula ϕ (i.e. a formula for which there are only *good* points, cfr. Sect. 2) when they are orthogonal.

$$m_j \diamond m_k \;=\; (\mathop{\top}\limits_{i\in P_j} x_i \;\top \mathop{\top}\limits_{i\in N_j} \neg x_i) \diamond (\mathop{\top}\limits_{i\in P_k} x_i \;\top \mathop{\top}\limits_{i\in N_k} \neg x_i) =$$

$$= \begin{cases} \phi & \text{if } m_j \text{ and } m_k \text{ are orthogonal} \\ (\mathop{\top}\limits_{i\in(P_j\cup P_k)} x_i \;\top \mathop{\top}\limits_{i\in(N_j\cup N_k)} \neg x_i\;) & \text{otherwise} \end{cases}$$

THEOREM 2 *Consider any two monomials m_j and m_k, with their corresponding sets B_j, G_j, B_k and G_k. Let $m_l = m_j \diamond m_k$ be their product. The set of the bad points for m_h is $B_l = B_j \cap B_k$, while the set of good points is $G_l = G_j \cup G_k$.*

Proof: Given a generic monomial m_j, by adding literals to m_j the set B_j can in general only be reduced (this means decreasing the false set for

CNF, decreasing the true set for DNF) and therefore the set G_j increased. Monomial m_l can be seen as adding literals to m_j, so $B_l \subseteq B_j$, and can also be seen as adding literals to m_k, so $B_l \subseteq B_k$. Therefore, $B_l \subseteq B_j \cap B_k$. Moreover, any point $x \in B_j \cap B_k$ is a bad point for m_l, hence $x \in B_l$. This proves $B_l = B_j \cap B_k$, and consequentially $G_l = G_j \cup G_k$. Coherently, when m_j and m_k are orthogonal, the result of the multiplication, by definition, is the empty formula ϕ, the sets B_j and B_k are disjoint by Theorem 1, their intersection is empty, and so is the set B_ϕ by definition. $\square$

Given an arbitrary monomial $m_j = (x_h \top x_{h+1} \top ... \top x_i)$, its negation $\neg(m_j)$ is easy computable (by De Morgan's laws [19]) as the following set of monomials connected by our external operator: $(\neg x_h) \perp (\neg x_{h+1}) \perp ... \perp (\neg x_i) = \neg(m_j)$. However, the expression for $\neg(m_j)$ is not unique. One could, in fact, consider a negation which is in orthogonal form, namely the *orthogonal negation* $\neg^o(m_j)$ of m_j. Such negation $\neg^o(m_j)$ is composed by k monomials $o_1^j \perp o_2^j \perp \ldots \perp o_k^j$, the first of them containing the negation of the first variable, the second of them containing the first variable and the negation of the second one, and so on, as follows.

$$(\neg x_h) \perp (x_h \top \neg x_{h+1}) \perp \; ... \; \perp (x_h \top x_{h+1} \top ... \top \neg x_i)$$

Example 3: The orthogonal negation of $m = (x_1 \top x_2 \top \neg x_3)$ is

$$\neg^o(m) = o_1^m \perp o_2^m \perp o_3^m = (\neg x_1) \perp (x_1 \top \neg x_2) \perp (x_1 \top x_2 \top x_3)$$

We also define the multiplication of a monomial m_k by the negation $\neg(m_j)$ of another monomial m_j as the set of monomials obtained multiplying m_k by each of the monomial in $\neg(m_j)$. We denote this operation by $m_k \diamond \neg(m_j)$. Basing on this, a basic orthogonalization operation can be developed. For clarity reasons, we report the procedure without indicating negative variables. However, this does not cause any loss of generality, since negative variables can perfectly be present, and the negations will eventually appear in the result according to elementary rules of Boolean algebra.

Basic Orthogonalization Operation: Consider any two distinct monomials m_j and m_k not already orthogonal. Let C_{jk} be the (possibly empty) set of common literals between m_j and m_k, and D_j and D_k the (possibly empty) sets of literals respectively belonging only to m_j and only to m_k.

$$m_j = (\bigtop_{i \in D_j} x_i \;\; \top \bigtop_{i \in C_{jk}} x_i) \quad m_k = (\bigtop_{i \in D_k} x_i \;\; \top \bigtop_{i \in C_{jk}} x_i)$$

Note that, since they are not orthogonal, they cannot contain complementary literals: $x_i \in m_j \Rightarrow \neg x_i \notin m_k$. Choose one of the sets of different literals, for instance D_j, and consider the monomial m_d composed by all such literals. Compute now its orthogonal negation $\neg^o(m_d) = o_1^d \perp o_2^d \perp \ldots \perp o_j^d$. We have that the (sub)formula $m_j \perp m_k$ is equivalent (in the sense that they both represent the same Boolean function, we prove this throughout the rest of this section) to the following (sub)formula obtained by replacing m_k with $m_k \diamond \neg^o(m_d)$.

$$m_j \perp o_1^d \diamond m_k \perp o_2^d \diamond m_k \perp \ldots \perp o_j^d \diamond m_k$$

The essential point is that the obtained (sub)formula is now in orthogonal form. Hence, the (sub)formula composed by the two monomials m_j and m_k have been orthogonalized. Note that the number of monomials of the result is 1 plus the cardinality of the set of non-common literals (D_j) used. In order to obtain a smaller number of monomials, we always choose the set of non-common literals of minimum cardinality. When one of this two sets is empty, this means that one of the monomials, say m_j, is a subset of the other m_k. Coherently, by choosing D_j for the above procedure, the result is only m_k. In fact, the Boolean (sub)formula $m_j \perp m_k$ is equivalent, in this case, to the Boolean (sub)formula m_k. The following two theorems prove that replacing m_k with $m_k \diamond \neg^o(m_d)$ produces an equivalent formula.

THEOREM 3 *Consider a monomials m_j and the negation $\neg(m_k)$ of another monomial, with their corresponding sets B_j, G_j, $B_{\neg k}$ and $G_{\neg k}$. The set of bad points for their product $m_j \diamond \neg(m_k)$ is $B_j \cap B_{\neg k}$, while the set of good points is $G_j \cup G_{\neg k}$.*

Proof: Denote the set of bad points for the h-th monomial of $\neg(m_k)$ by $B_{\neg k}^h$, and denote the number of monomial composing $\neg(m_k)$ by p. We clearly have $B_{\neg k} = \bigcup_{h=1}^p B_{\neg k}^h$. Moreover, by Theorem 2, for each single monomial product constituting $m_j \diamond \neg(m_k)$ we have that the corresponding set of bad points is $B_j \cap B_{\neg k}^h$. Therefore, the set of bad points of the entire $m_j \diamond \neg(m_k)$ is $\bigcup_{h=1}^p (B_j \cap B_{\neg k}^h)$, which is $B_j \cap B_{\neg k}$. As a consequence, also the set of good points of the entire $m_j \diamond \neg(m_k)$ is $G_j \cup G_{\neg k}$. $\square$

THEOREM 4 *Consider an arbitrary Boolean formula $\mathcal{F}$ in normal form representing the Boolean function $f(x_1, ..., x_n)$. If an arbitrary monomial $m_j \in \mathcal{F}$ is multiplied by the negation $\neg(m_k)$ of another arbitrary monomial $m_k \in \mathcal{F}$, the new Boolean formula obtained $\mathcal{F}'$ still represents the same $f(x_1, ..., x_n)$.*

Proof: It is sufficient to prove that the sets B and G are the same for $\mathcal{F}$ and $\mathcal{F}'$. As can be observed in the following Fig. 2, monomial m_k

determines in $\mathbb{B}^n$ a partition in B_k and G_k. Its negation $\neg(m_k)$ determines a partition $B_{\neg k} = G_k$ and $G_{\neg k} = B_k$. Now multiply another monomial m_j by $\neg(m_k)$, obtaining new monomials $m_j \diamond \neg(m_k)$, add $m_j \diamond \neg(m_k)$ and remove m_j from the formula $\mathcal{F}$, obtaining $\mathcal{F}'$. The set G'_j corresponding to $m_j \diamond \neg(m_k)$, by Theorem 3, is $G_j \cup G_{\neg k}$, which is $\supseteq G_j$. So, the set of *good* points G for the formula $\mathcal{F}$, which is the intersection of all the G_j, cannot decrease. We now prove that G cannot increase. It could only increase in the area of $G_{\neg k}$, since $G'_j = G_j \cup G_{\neg k}$. However, all points of $G_{\neg k}$ are forbidden by the fact that $G \subseteq G_k$. Hence, G is the same for $\mathcal{F}$ and $\mathcal{F}'$, and therefore B also remains the same. The thesis follows. □

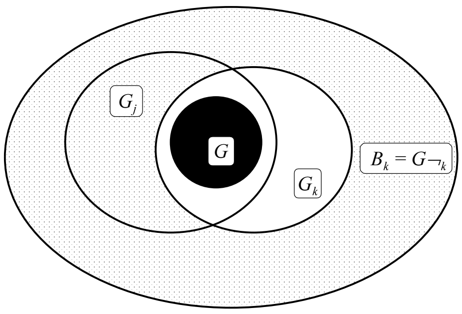

Figure 2. The partition of the Boolean hypercube $\mathbb{B}^n$ determined by B_k and G_k

Example 4: Given the formula composed by two monomials m_1 and m_2.

$$m_1 = (x_1 \top \neg x_2 \top x_5) \perp (\neg x_2 \top x_3 \top x_4) = m_2$$

the defined sets of non-common literals are

$$D_1 = (x_1 \top x_5) \qquad \text{and} \qquad D_2 = (x_3 \top x_4)$$

Their cardinality is the same. We choose D_1, and the orthogonal negation of the monomial corresponding to D_1 is the following.

$$(\neg x_1) \perp (x_1 \top \neg x_5)$$

By using the orthogonalization operation, the above formula becomes

$$(x_1 \top \neg x_2 \top x_5) \perp ((\neg x_1) \diamond (\neg x_2 \top x_3 \top x_4)) \perp ((x_1 \top \neg x_5) \diamond (\neg x_2 \top x_3 \top x_4))$$

which is the following orthogonal formula.

$$(x_1 \top \neg x_2 \top x_5) \perp (\neg x_1 \neg x_2 \top x_3 \top x_4) \perp (x_1 \top \neg x_2 \top x_3 \top x_4 \top \neg x_5)$$

Theorem 5 *Given an arbitrary Boolean formula $\mathcal{F}$ in normal form, representing the Boolean function $f(x_1, ..., x_n)$, it is always possible to transform it into an orthogonal normal form $\mathcal{O}$ still representing same $f(x_1, ..., x_n)$.*

Proof: The (constructive) proof is given by the above orthogonalization operation, since that is a general procedure capable of orthogonalizing any two monomials. Define the orthogonalization of two monomial by means of such procedure as a *step.* Given therefore an arbitrary formula with m monomials, by iterating this orthogonalization operation to exhaustion until every pair of monomials are orthogonal, the orthogonal form is obtained in a finite number of steps, at most $\binom{m}{2}$. □

5. Improvements on the Basic Procedure

Unfortunately, by repeatedly applying above operation to exhaustion, the size of the formula tends to exponentially increase. As remarked above, this is not surprising, since the process of orthogonalization makes easy an NP-complete problem like satisfiability. Hence, the NP complexity [9] can be seen as being *absorbed* by the orthogonalization process, so it is unlikely that the orthogonalization process can be made inexpensive. However, improvements on the above basic procedure, with the aim of minimizing the size of the formula both in the final result and during the computation, are possible, as follows.

5.1. Absorption of Implied Monomials

Consider two generic monomials m_j and m_k appearing in the same formula $\mathcal{F}$, which represents the Boolean function $f(x_1, ..., x_n)$, as follows.

$$m_j = (\mathop{\Large\top}_{i \in P_j} x_i \ \top \mathop{\Large\top}_{i \in N_j} \neg x_i) \qquad m_k = (\mathop{\Large\top}_{i \in P_k} x_i \ \top \mathop{\Large\top}_{i \in N_k} \neg x_i)$$

If $P_j \subseteq P_k$ and $N_j \subseteq N_k$, monomial m_k is logically implied by m_j [11], [19], and can therefore be removed from $\mathcal{F}$ obtaining a smaller formula $\mathcal{F}'$ still representing the same $f(x_1, ..., x_n)$. This operation is applied in order to reduce the number of monomials in the formula.

5.2. Synthesis Resolution

This operation is a special case of the general operation called resolution [1], [17] in the case of CNF, and consensus [15] in the case of DNF. Given a formula $\mathcal{F}$ representing the Boolean function $f(x_1, ..., x_n)$. If $\mathcal{F}$ contains two monomials which are identical except for one literal x_s appearing positive in one monomial and negative in the other, as follows:

$$m_j \;=\; (x_s \top \mathop{\top}\limits_{i \in P_j} x_i \;\top \mathop{\top}\limits_{i \in N_j} \neg x_i) \qquad m_k = (\mathop{\top}\limits_{i \in P_j} x_i \;\top \mathop{\top}\limits_{i \in N_j} \neg x_i \;\top \neg x_s)$$

their resolvent [17] m_r, reported below, can be added to $\mathcal{F}$, obtaining a new formula $\mathcal{F}'$ still representing the same Boolean function $f(x_1, ..., x_n)$.

$$m_r \;=\; (\mathop{\top}\limits_{i \in P_j} x_i \;\top \mathop{\top}\limits_{i \in N_j} \neg x_i)$$

Moreover, their resolvent logically implies both its parents m_j and m_k, hence they can be removed from the formula, obtaining a new formula $\mathcal{F}''$ still representing the same Boolean function $f(x_1, ..., x_n)$. This operation helps in reducing the number of monomials in the formula.

Finally, being our aim not to excessively increase the size of the formula, for each orthogonalization step t, we define the *quality* q_t of such step as the number o_t of clauses orthogonalized by such step divided by the number n_t of new clauses created by such step: $q_t = o_t/n_t$ (it can be computed in advance). In our procedure, we set an initial quality limit q_{limit}, in order to initially perform the most convenient basic orthogonalizations. During iterations, at the beginning of each phase of basic orthogonalization steps, if no steps respecting current limit are possible, current limit is decreased.

Complete Orthogonalization Procedure

```
Input: An arbitrary Boolean formula $\mathcal{F}$ in CNF or DNF
Output: An equivalent Boolean formula $\mathcal{F}'$ in OCNF or ODNF

Repeat :
    If the current formula is orthogonal, Stop
    Else if $q_{\text{limit}}$ allows no basic orthogonaliz. steps, decrease $q_{\text{limit}}$
    Perform all basic orthogonalization steps of quality $q \geq q_{\text{limit}}$
    Perform all possible synthesis resolutions
    Perform all possible absorptions
```

6. Testing of the Procedure

The algorithm was tested on artificially generated CNF formulae obtained from the SATLIB collection of the Darmstadt University of Technology.

Table 2. Orthogonalization procedure on artificially generated CNF formulae.

Problem	m_{init}	m_{max}	m_{ortho}	time
uf20-01	91	1443	130	1.93
uf20-02	91	912	100	1.53
uf20-03	91	859	132	1.59
uf20-04	91	861	134	1.09
uf20-05	91	341	31	0.11
uf20-06	91	723	40	0.58
uf20-07	91	506	85	0.49
uf20-08	91	895	136	1.67
uf20-09	91	1669	144	4.49
uf20-010	91	1068	128	1.72
uf20-011	91	418	130	0.35
uf20-012	91	1117	190	2.58
uf20-013	91	784	65	0.74
uf20-014	91	947	167	2.19
uf20-015	91	980	120	1.78
uf20-016	91	954	102	1.47
uf20-017	91	787	109	1.27
uf20-018	91	1530	105	2.44
uf20-019	91	861	70	1.09
uf20-020	91	1335	98	2.64
uf20-021	91	870	73	0.59
uf20-022	91	700	79	0.81
uf20-023	91	1575	211	3.40
uf20-024	91	837	171	1.63
uf20-025	91	935	82	1.31
uf20-026	91	836	59	1.05
uf20-027	91	888	63	0.78
uf20-028	91	740	93	1.03
uf20-029	91	618	66	0.70
uf20-030	91	509	90	0.61

They represent 3-SAT problems. The following Table 2 reports the number of monomials of the original formula (m_{init}), the number of monomial in the orthogonalized formula produced (m_{ortho}), the maximum number of monomial reached by the formula during the orthogonalization process (m_{max}), and computational time in seconds on a PC Pentium IV 1.7GHz. Such testing is intended solely as a study of the behavior of the orthogonalization procedure, since it currently does not constitute, from the practical point of view, a fast alternative for solving satisfiability problems.

It can be observed that the number of monomials in the orthogonalized formula generally increases, although not always. Moreover, intermediate formulae contains a much larger number of monomials. This turns out to be a general rule in performing similar operations. However, there are practical applications where the advantages of having the orthogonal form completely surmount the disadvantage of such size increase.

7. Conclusions

The orthogonal form of a Boolean formula has remarkable properties. Several hard problems become easy when in orthogonal form. A general procedure for the orthogonalization of an arbitrary CNF or DNF is developed. A unified and coherent notation for representing at the same time CNF and DNF is therefore introduced. The procedure is proved to always produce the orthogonal form (OCNF or ODNF) in a finite number of steps. The problem is indeed computationally demanding. As predictable, in the initial phase of the procedure, the size of the formula tends to exponentially increase. On the other hand, the size of the formula decreases again when approaching to the final phase. In spite of this size growth, orthogonalization appears to be the preferable way to solve some practical problems, for instance in the field of reliability theory. Some computational complexity implications of the orthogonalization process are analyzed.

Acknowledgments The author wish to thank Peter L. Hammer and Endre Boros for introducing the discussed problems to his attention.

References

1. A. Blake. Canonical Expressions in Boolean Algebra. *Ph.D. thesis*, University of Chicago, 1937.
2. M.O. Ball and J.S. Provan. Disjoint products and efficient computation of reliability. *Operations Research*, 36:703-715, 1988.
3. R.E. Barlow and F. Proschan. *Statistical Theory of Reliability and Life Testing.* Holt, Rinehart and Winston, New York, 1975.
4. R.E. Barlow, C.A. Clarotti, and F. Spizzichino (eds.). Reliability and Decision Making. Chapman and Hall, 1993.
5. E. Boros, Y. Crama, O. Ekin, P.L. Hammer, T. Ibaraki, and A. Kogan. Boolean Normal Forms, Shellability, and Reliability Computations. *SIAM Journal on Discrete Mathematics*, 13(2):212-226, 2000.
6. E. Boros, Y. Crama, P.L. Hammer, and M. Saks. A complexity index for Satisfiability Problems. *SIAM Journal on Computing*, 23:45–49, 1994.
7. H. Kleine Büning and T. Lettmann. *Aussagenlogik: Deduktion und Algorithmen.* B.G. Teubner, Stuttgart, 1993. Reprinted in English.
8. V. Chvatal. Resolution Search. *Discrete Applied Mathematics*, 73:81–99, 1997.
9. M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, N.Y., 1979.
10. J. Gu, P.W. Purdom, J. Franco, and B.W. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. *DIMACS Series in Discrete Mathematics* 35:19-151, American Mathematical Society, 1997.
11. P.L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas.* Springer-Verlag, New York, 1968.

12. K. Iwama. CNF Satisfiability Test by Counting and Polynomial Average Time. *SIAM J. Computing* 18:385–391, 1989.
13. D.W. Loveland. *Automated Theorem Proving.* North Holland, 1978.
14. J.S. Provan. Boolean decomposition schemes and the complexity of reliability computations. *DIMACS series in Discrete Math.*, American Mathematical Society, Providence, RI, 213–228, 1991.
15. W.V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62:627–631, 1955.
16. K.G. Ramamurthy. *Coherent Structures and Simple Games.* Kluwer Academic Publishers, Dordrecht, The Netherlands 1990.
17. J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23-41, 1965.
18. H. Scheffè. *The analysis of Variance.* J. Wiley, New York, 1959.
19. I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner series in Computer Science, 1987.