Orthogonalization of a Boolean Function

Renato Bruni[z] and Peter L. Hammer[x]

December 20, 2000

Abstract

Orthogonalization is the process of transforming a conjunctive normal form of a Boolean function to orthogonal conjunctive normal form, i.e. to a normal form in which any two clauses contain a pair of complementary literals. Orthogonal disjunctive normal form is defined similarly. The problem is of great relevance in several application, e.g. the reliability theory and the propositional satisfiability problem. We propose a procedure for transforming an arbitrary CNF or DNF to an orthogonal one, and present the results of computational experiments carried out on randomly generated Boolean formulae.

Keywords: Boolean formulae, NP-completeness, Satisfiability.

1 Introduction

Let $B = \{0,1\}$, or, equivalently, $\{True, False\}$. A Boolean function is a function $f(x_1, x_2, \ldots, x_n)$ from the Boolean hypercube B^n to the Boolean set B. A Boolean function can be represented in several ways. A widely used one is by means of a Boolean formula F in Conjunctive (CNF) or Disjunctive (DNF) normal form.

A Boolean CNF formula is the logic conjunction ($\wedge$) of m clauses, which are logic disjunction ($\vee$) of literals, which can be either posited proposition (x_i) or negated ($\neg x_i$). The general structure is therefore the following.

$$(x_{i_1} \vee \ldots \vee x_{j_1} \vee \neg x_{j_1+1} \vee \ldots \vee \neg x_{n_1}) \wedge \ldots \wedge (x_{i_m} \vee \ldots \vee x_{j_m} \vee \neg x_{j_m+1} \vee \ldots \vee \neg x_{n_m})$$

[z] Dipartimento di Informatica e Sistemistica, Universita di Roma "La Sapienza", via Buonarroti 12 - 00185 Roma, Italy. E-mail: bruni@dis.uniroma1.it

[x] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ, 08854-8003 USA. E-mail: hammer@rutcor.rutgers.edu

Conversely, a Boolean DNF formula is the logic disjunction (_) of m terms, which are logic conjunction (^) of literals. The general structure is:

$$(x_{i_1} \wedge \cdots \wedge x_{j_1} \wedge \neg x_{k_1} \wedge \cdots \wedge \neg x_{n_1}) \vee \cdots \vee (x_{i_m} \wedge \cdots \wedge x_{j_m} \wedge \neg x_{k_m} \wedge \cdots \wedge \neg x_{n_m})$$

In order to handle both CNF and DNF, in section 2 we introduce a general notation for normal forms. The orthogonal form results of great relevance in solving several hard problems, e.g. in the reliability theory. A basic procedure to reach the orthogonal form is described in section 4. During the above process, the size of the formula tends to exponentially increase. We therefore present, in section 5, some improvements of the basic procedure, with the aim of minimizing the size of the formula both in the final result and during the computation. The proposed procedure is tested on a set of artificially generated Boolean formulae. Results are in section 6.

2 Notation

We will develop a procedure that applies both to CNF and DNF. We therefore need a notation which can represent both forms.

Clauses and terms can be viewed as sets of literals. We will call both of them monomials m_i. A CNF or DNF formula F is therefore a collection of sets of literals, hence a collection of monomials. We have an operator applied between monomial, that will be here indicated with the symbol ? (external operator), and an operator applied between literals of the same monomial, that will be here indicated with the symbol > (internal operator). Both CNF and DNF will therefore be represented as follows.

$$(x_{i_1} > \cdots > x_{j_1} > \neg x_{k_1} > \cdots > \neg x_{n_1}) ? \cdots ? (x_{i_m} > \cdots > x_{j_m} > \neg x_{k_m} > \cdots > \neg x_{n_m})$$

Where the following conventions hold:

? means ^ if we are considering CNF, and _ if we are considering DNF
> means _ if we are considering CNF, and ^ if we are considering DNF

Given a monomial m_i, we have a set $T_i \subseteq B^n$ where m_i is 1 (True), and a set $F_i \subseteq B^n$ (the complement of T_i with respect to B^n) where m_i is 0 (False). When solving several NP problems on Boolean formulae, e.g. the Satisfiability problem, what we actually want to know is some information about the set of True points $T = \{X \in B^n : f(X) = 1\}$ or, equivalently, about the set of False points $F = \{X \in B^n : f(X) = 1\}$ for the whole

function. Due to normal form, we already have some relations between the T_i and the global T, and between the F_i and F.

Proposition 2.1. In the case of CNF (a conjunction), it results:

$$T = \bigcap_{i=1} T_i$$

whereas, in the case of DNF (a disjunction), it results:

$$T = \bigcup_{i=1} T_i$$

Of course, specular results hold for the False set F:

Proposition 2.2. In the case of CNF (a conjunction) it results:

$$F = \bigcup_{i=1} F_i$$

whereas, in the case of DNF (a disjunction), it results:

$$F = \bigcap_{i=1} F_i$$

In the general case, such sets are not disjoint, but can overlap each other: it can be $T_i \setminus T_j \neq \emptyset$ or $F_i \setminus F_j \neq \emptyset$ for some $i, j \in \{1, \ldots, n\}$. For the above reason, to find respectively the cardinality $|T|$ and $|F|$, we need to identify, at least in an implicit way, respectively all the T_i and all the F_i. The cardinality $|T|$ or $|F|$ gives us, for example, the solution of the feasibility version of the propositional Satisfiability problem, which is well known NP-complete. This theoretically means, moreover, that every problem in NP can be polynomially reduced to the problem of finding this cardinality.

Since the number of points in T_i and F_i is, in the worst case, exponential in the length of the monomials m_i, the approach of identifying all the T_i and all the F_i has exponential worst-case time complexity. This is not surprising. On the other hand, if all the T_i (resp. all the F_i) would be pairwise disjoint sets, in order to find the cardinality $|T|$ (resp. $|F|$) it would suffice to know

the cardinality of the T_i (resp. F_i), and sum them. Such cardinalities are, in fact, trivially computable.

In order to complete notation uni¯cation, let us consider again the Satis¯ability problem. In the case of CNF formulae, it consists in ¯nding if, in the Boolean hypercube B^n, there is at least one true point for all the clauses. Conversely, for DNF formulae, it consists in ¯nding if there is at least one false point for all the terms. Altogether, false points are bad for CNF, while true points are bad for DNF.

We will call the set of such bad points U, with the convention that $U = F$ for CNF, and $U = T$ for DNF. Moreover, every monomial m_i has his set of bad points U_i of the Boolean hypercube B^n, with the convention that $U_i = F_i$ for CNF, and $U_i = T_i$ for DNF. (More intuitively, every m_i forbids a set of points: in the case of CNF, every m_i forbids its F_i, while, in the case of DNF, every term m_i forbids its T_i).

Conversely, we will call V the set of good points, with the convention that $V = T$ for CNF, and $V = F$ for DNF. Every monomial m_i has therefore his set of good points V_i, with the convention that $V_i = T_i$ for CNF, and $V_i = F_i$ for DNF.

Form	internal op.	external op.	bad pt.	good pt.
Uni¯ed	$>$	?	U	V
CNF	$\_$	$\wedge$	F	T
DNF	$\wedge$	$\_$	T	F

Table 1: Convention used to unify notation for CNF and DNF.

Proposition 2.3. The cardinality of the above U_i and V_i are easily computable. Let n be the number of variables, and $l(m_i)$ be the number of distinct literals appearing in monomial m_i, we have $|U_i| = 2^{n-l(m_i)}$, and $|V_i| = 2^n - |U_i| = 2^n - 2^{n-l(m_i)}$.

We will indicate with (Á) the empty monomial, i.e. the monomial $m_Á$ which is an empty set of literals. According to proposition 2.3, $U_Á = B^n$, ($m_Á$ has only bad points). We will instead indicate with Á the empty formula, i.e. the formula $F_Á$ which is an empty set of monomials. By de¯nition, $F_Á$ will always evaluate to (V).

3 The Orthogonal form

We declare a formula to be in orthogonal form when, for every pair of monomials m_i and m_j, at least one boolean variable x_k appears direct in one (for instance m_i) and negated in the other (for instance m_j). Any two monomials have therefore the following structure:

$$m_i = (:::>x_k>:::);\ \ m_j = (:::>:x_k>:::) \quad 8\ i;j\ 2\ f1:::ng$$

We will say that the above terms are orthogonal, or clash [4] on x_k, or resolve [9] on x_k, or also hit [3] on x_k. This holds both for CNF and DNF.

Theorem 3.1. For a Boolean formula in orthogonal form, the sets U_i are pairwise disjoint.

This particularizes for CNF as:

$$F_i \setminus F_j = Á \quad 8\ i;j\ 2\ f1:::mg$$

$$(T_i \setminus T_j \text{ can be } \neq Á \quad \text{for some } i;j\ 2\ f1:::mg)$$

and for DNF as:

$$T_i \setminus T_j = Á \quad 8\ i;j\ 2\ f1:::mg$$

$$(F_i \setminus F_j \text{ can be } \neq Á \quad \text{for some } i;j\ 2\ f1:::mg)$$

Proof: Orthogonal form) $U_i \setminus U_j = Á \quad 8\ i;j\ 2\ f1:::mg$.

U_i corresponding to monomial m_i is a set of points in B^n de¯ned by a pattern obtainable from the m_i itself. For example, U_i corresponding to the CNF clause $(x_1 \_ :x_3)$ on B^4 is de¯ned by the pattern (0, *, 1, *), representing the 4 points (0, 0, 1, 0), (0, 0, 1, 1), (0, 1, 1, 0), (0, 1, 1, 1). If two monomials m_i and m_j clash on at least one variable x_c, the corresponding U_i and U_j are de¯ned by two patterns which respectively have 0 and 1 in at least position c, hence they de¯ne two sets U_i and U_j which cannot have any common point.

Proof: Orthogonal form ($U_i \setminus U_j = Á \quad 8\ i;j\ 2\ f1:::mg$.

Let us consider two Boolean point $x^0 = (x^0{}_1;x^0{}_2;:::;x^0{}_n)\ 2\ U_i$ and $x^{00} = (x^{00}{}_1;x^{00}{}_2;:::;x^{00}{}_n)\ 2\ U_j$, with $U_i \setminus U_j = Á$. x^0 and x^{00} must be di®erent (and binary), hence at least one component is respectively 0 and 1. Let us call

that component x_c Monomials m_i and m_j corresponding to U_i and U_j must therefore both contain the variable x_c, and clash on it.

Example: Suppose we are interested in solving the Satis¯ability problem for the following CNF. To solve our problem we need to check whether the global F covers the whole B^5. In the general case, we can only proceed by identifying F as the intersection of the F_i.

It is straightforward to ¯nd the corresponding F_i (and their cardinality).

$$
\begin{array}{rcl}
(\ x_1 \_ : x_2 \_\ \ x_3 \_\ \ x_4 \_\ \ x_5) & ! & F_1 = \{0;1;0;0;0\} \quad |F_1| = 1 \\
(: x_1 \_ : x_2 \_\ \ x_3 \_\ \ x_4 \_\ \ x_5) & ! & F_2 = \{1;1;0;0;0\} \quad |F_2| = 1 \\
(\ x_2 \_\ \ x_3 \_\ \ x_4 \_\ \ x_5) & ! & F_3 = \{\ast;0;0;0;0\} \quad |F_3| = 2 \\
(\ x_3 \_ : x_4 \_\ \ x_5) & ! & F_4 = \{\ast;\ast;0;1;0\} \quad |F_4| = 4 \\
(\ x_3 \_\ \ x_4 \_ : x_5) & ! & F_5 = \{\ast;\ast;0;0;1\} \quad |F_5| = 4 \\
(\ x_3 \_ : x_4 \_ : x_5) & ! & F_6 = \{\ast;\ast;0;1;1\} \quad |F_6| = 4 \\
(: x_3) & ! & F_7 = \{\ast;\ast;1;\ast;\ast\} \quad |F_7| = 16
\end{array}
$$

No more in a straightforward way, by identifying all the points of the intersection of the F_i , we can observe that F actually covers B^5 (see picture 1 below). Hence, given CNF is unsatis¯able.

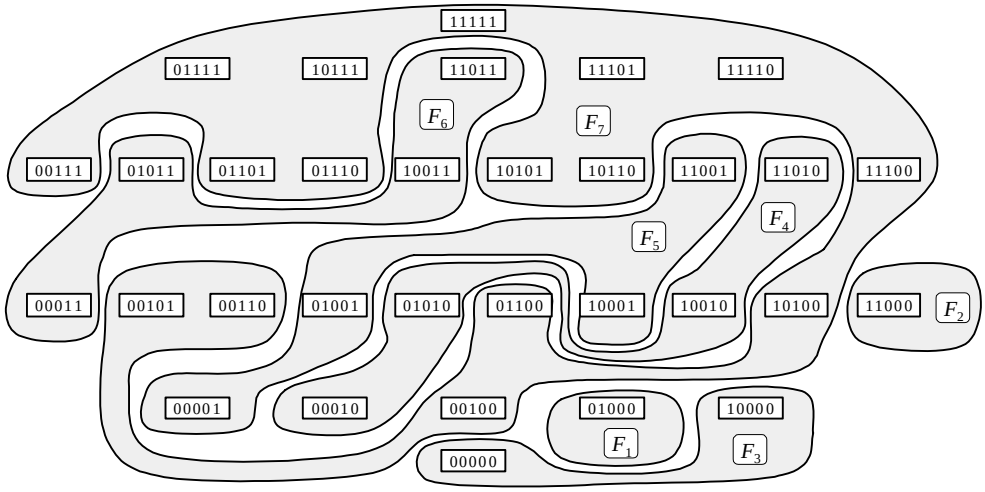

Figure 1: Individuation of the False sets F_i on the Boolean hypercube B^5

The number of points in this intersection is, unfortunately, exponential (in the worst case) in the size of the formula. This gives worst case exponential time complexity to such procedure.

On the other hand, we could observe that the CNF is in orthogonal form, hence we have pairwise disjoint U_i. In the case of CNF, this means pairwise disjoint F_i. On this basis, we easily have $|F| = |F_1| + |F_2| + |F_3| + |F_4| + |F_5| + |F_6| + |F_7| = 32$. Since F covers B^5 i® $|F| = 2^5 = 32$, this is the case, and given CNF is unsatis¯able. This is obtained just by counting, as shown in [6].

4 Basic Orthogonalization operation

Given an arbitrary Boolean formula F in normal form, representing the Boolean function $f(x_1; x_2; :::; x_n)$, our aim is to put it in the orthogonal form O while still representing the same $f(x_1; x_2; :::; x_n)$. This can always be done as follows.

We ¯rst need to de¯ne the multiplication ($¦$) of two monomials m_i and m_j. The result of such multiplication operation is a new monomial containing all the literals of m_i and of m_j (but without repeated ones) when the two monomials are not orthogonal, and Á when they are orthogonal. Note that Á means an empty formula, i.e. a formula for which there are only good points, and not a formula made by an empty monomial, i.e. a formula for which there are only bad points. Formally:

$$m_i ¦ m_j = (x_{ih} > ::: > x_{ik}) ¦ (x_{jh} > ::: > x_{jk}) = \begin{cases} Á & \text{if } m_i \text{ and } m_j \text{ are orthogonal} \\ (x_{ih} > x_{jh} > ::: > x_{ik} > x_{jk}) & \text{else} \end{cases}$$

Proposition 4.1. *Consider any two monomials m_i and m_j, with their corresponding sets U_i, V_i, U_j and V_j. Let $m_k = m_i ¦ m_j$ be their product. The set of the bad points for m_k is $U_k = U_i \setminus U_j$, while the set of good points is $V_k = V_i [V_j$*

We can use such multiplication operation to make any two monomials orthogonal each other. In fact:

Theorem 4.1. *Consider an arbitrary Boolean formula F in normal form representing the Boolean function $f(x_1; x_2; :::; x_n)$. If we multiply an*

arbitrary monomial $m_i \in F$ by the negation $:m_j$ of another arbitrary monomial $m_j \in F$, we obtain a new Boolean formula F' still representing the same $f(x_1; x_2; :::; x_n)$.

Proof: We need to prove that sets U and V are the same for F and F'. As we can observe in the following ¯gure 2, monomial m_j determines in B^n a partition in U_j and V_j. Its negation $:m_j$ determines a partition $U_{:j} = V_j$ and $V_{:i} = U_i$.

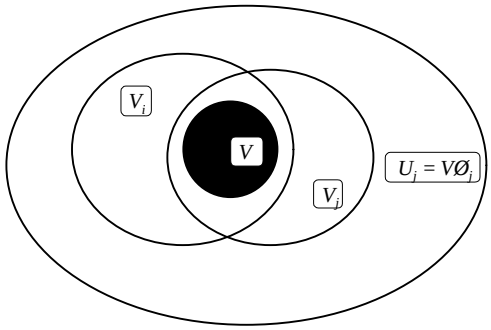

Figure 2: Individuation of the False sets F_i on the Boolean hypercube B^5

Now we multiply another monomial m_i by $:m_j$, obtaining the new monomial m'_i, add m'_i and remove m_i form the formula. The set V'_i corresponding to this new monomial, by proposition 4.1, is $V_i [V_{:j}$, which is $\neg V_i$. So the set of good points for the formula V, which is the intersection of all the V_i, cannot decrease. It could increase in the area of $V_{:j}$, but such area is forbidden by the fact that $V \mu V_j$. Hence, V is the same for F and F', and therefore U also remains the same. The thesis follows.

Given an arbitrary monomial $m_i = (x_h > x_{h+1} > ::: > x_k)$, its negation (by De Morgan's laws) is easy computable as the following set of monomials connected by our external operator. $(:x_h)?(:x_{h+1})?:::?(:x_k) = :(m_i)$. But the expression for $:m_i$ is not unique. We could, in fact, consider a negation which is in orthogonal form, namely the orthogonal negation $:^o(m_i)$ of m_i. $:^o(m_i)$ is made of k monomials $o_1^{m_i} ? o_2^{m_i} ? ::: ? o_k^{m_i}$ corresponding to the negation of the ¯rst literal, the ¯rst and the negation of the second, and so

on, as follows.

$$(:x_h)?(x_h>:x_{h+1})?:::?(x_h>x_{h+1}>:::>:x_k)$$

Example The orthogonal negation of

$$m = (x_1>x_2>:x_3)$$

is

$$:^o(m) = o_1^m?o_2^m?o_3^m = (:x_1)?(x_1>:x_2)?(x_i>x_2>x_3)$$

Basing on the above results, we can develop the following procedure.

Basic Orthogonalization Operation: Without loss of generality, consider any two monomials m_1 and m_2 not already orthogonal.

$$m_1 = (x_i>:::>x_j>x_{j+1}>:::>x_h) \quad m_2 = (x_{j+1}>:::>x_h>x_{h+1}>:::>x_k)$$

m_1 and m_2 have, in general, a common set of literals $c_{1;2} = (x_{j+1}>:::>x_h)$ and two sets of literals d_1 and d_2 which are not in common: $d_1 = (x_i>:::>x_j)$ for m_1, and $d_2 = (x_{h+1}>:::>x_k)$ for m_2. Note that, since they are not orthogonal, they cannot contain complementary literals: if $x_i \in m_1$) $:x_i \notin m_2$.

Choose one of them, say d_1, and consider its orthogonal negation $:^o(d_1) = o_1^{d_1}?o_2^{d_1}?:::?o_j^{d_1}$.

The (sub)formula $m_1?m_2$, is equivalent to the following (sub)formula, in the sense that they both represent the same Boolean function.

$$m_1 ? o_1^{d_1} ¦ m_2 ? o_2^{d_1} ¦ m_2 ? ::: ? o_j^{d_1} ¦ m_2$$

Note that the obtained (sub)formula is in orthogonal form. The number of monomials is 1 plus the cardinality of the set of non-common literals (d_1) we used. In order to obtain a smaller number of monomials, we always choose the set of non-common literals of minimum cardinality.

Example Given a formula made up of the two monomials m_1 and m_2.

$$m_1 = (x_1>:x_2>x_5)?(:x_2>x_3>x_4) = m_2$$

the sets of non-common literals are

$$d_1 = (x_1 > x_5) \qquad \text{and} \qquad d_2 = (x_3 > x_4)$$

Their cardinality is the same. We choose d_1, and its orthogonal negation is the following.

$$(:x_1) ? (x_1 > :x_5)$$

By performing the orthogonalization operation, the above formula is equivalent to

$$(x_1 > :x_2 > x_5) ? ((:x_1) \nmid (:x_2 > x_3 > x_4)) ? ((x_1 > :x_5) \nmid (:x_2 > x_3 > x_4))$$

which is the following orthogonal formula.

$$(x_1 > :x_2 > x_5) ? (:x_1 : x_2 > x_3 > x_4) ? (x_1 > :x_2 > x_3 > x_4 > :x_5)$$

The above is a general procedure to orthogonalize any two monomials. Given any formula, by iterating this orthogonalization operation to exhaustion, until every pair of monomials are orthogonal, we can always reach the orthogonal form.

5 Improvements

Unfortunately, by repeatedly applying above operation to exhaustion, the size of the formula tends to exponentially increase. To reduce the size of the formula, we will make use of the following observation.

5.1 Recognition of Internally Orthogonal Sets of Terms

A set of monomials S is internally orthogonal when each monomial $m_i \in S$ of them is orthogonal to every $m_j \in S$, for all $m_i; m_j \in S$. Given a generic formula, some sets of monomials may already be internally orthogonal.

We can partition the set of monomials in sets S_i which are already internally orthogonal $S_1; :::; S_p$. The original formula can therefore be represented as follows.

$$S_1 ? ::: ? S_p$$

Given a set of monomials S_i, we can consider its orthogonal negation $:^o(S_i)$, which is a set of new monomials $o_j^{S_i}$ corresponding to the negation

of S_i in orthogonal form. This is obtainable in a straightforward way from the de¯nition of orthogonal negation of a monomial.

$$:^o(S_i) = o_1^{S_i} ? o_2^{S_i} ? \cdots ? o_k^{S_i}$$

This lead us to the extended orthogonalization operation: We de¯ne the multiplication (¦) of two sets of monomials S_1 and S_2 as a new set of all the monomials obtained by calculating $m_i ¦ m_j$ for all $m_i \in S_1$ and all $m_j \in S_2$. (The multiplication $m_i ¦ m_j$ is de¯ned above.)

Without loss of generality, given any two internally orthogonal sets, the multiplication (¦) of

$$S_1 = (m_i ? m_{i+1} ? \cdots ? m_j) \quad S_2 = (m_h ? m_{h+1} ? \cdots ? m_k)$$

The (sub)formula $S_1 ? S_2$ is equivalent, in the sense speci¯ed above, to the following (sub)formula.

$$S_1 ? :^o(S_1) ¦ S_2$$

The above is a general procedure to orthogonalize any two sets of internally orthogonal monomials. Given any formula, by iterating this orthogonalization operation to exhaustion, we can always reach the orthogonal form. We can moreover take advantage of the two following simplifying operations.

5.2 Absorption

One monomial is implied by another one if it contains another one. Given a formula containing the following two monomials,

$$m_1 = (x_i > \cdots x_j > x_{j+1} > \cdots > x_h > x_{h+1} > \cdots > x_k) \quad m_2 = (x_{j+1} > \cdots > x_h)$$

the ¯rst can be deleted obtaining a new formula which is equivalent, in the sense speci¯ed above, to the original one. This operation is particularly useful in reducing the number of monomials in the formula.

5.3 Synthesis Resolution

This operation is a special case of the general operation called resolution [9, 1] in the case of CNF, and consensus [8] in the case of DNF. Given a formula containing two monomials which are identical except for one literal x_i appearing positive in one monomial and negative in the other, hence with the following structure:

$$m_1 = (x_i > x_h > \cdots > x_k) \quad m_2 = (: x_i > x_h > \cdots > x_k)$$

we can add to the formula their resolvent [9] obtaining a new formula which is equivalent, in the sense specified above, to the original one.

$$t_3 = (x_h > \cdots > x_k)$$

In particular, in this case, such resolvent absorbs both its parents. We can therefore remove from the formula both its parents, obtaining a new formula which is equivalent, in the sense specified above, to the original one. This operation helps in reducing the number of monomials in the formula.

6 Complete Orthogonalization Operation

So far, we have a set of operation that can be performed on the original formula in order to put it in orthogonal form. Being our aim not to increase too much the size of the formula, we define the quality Q of an orthogonalization step as the number of clauses orthogonalized divided by the number of new clauses created. A simple way to The algorithm is therefore

1. Find a partition in already orthogonal sets of clauses $S_1; \ldots; S_p$
2. Perform all extended orthogonalization steps of quality $Q \geq Q_{limit1}$
3. Perform all basic orthogonalization steps of quality $Q \geq Q_{limit2}$
4. Perform all possible synthesis resolution.
5. Perform all possible absorption
6. Repeat until all orthogonal

7 Testing

The algorithm was tested on a set of CNF formulae representing satisfiability instances. They are obtained from the SATLIB web site of the Darmstadt University of Technology. Such instances are 3-sat artificially generated problem.

Problem	n	m	literals	sol	time	$m_{orthogonal}$
uf20-01	20	91	273	Y	6.52	130
uf20-02	20	91	273	Y	5.00	100
uf20-03	20	91	273	Y	5.32	132
uf20-04	20	91	273	Y	3.42	134
uf20-05	20	91	273	Y	0.32	31
uf20-06	20	91	273	Y	1.80	40
uf20-07	20	91	273	Y	1.54	85
uf20-08	20	91	273	Y	5.55	136
uf20-09	20	91	273	Y	15.47	144
uf20-010	20	91	273	Y	5.84	128
uf20-011	20	91	273	Y	1.07	130
uf20-012	20	91	273	Y	8.34	190
uf20-013	20	91	273	Y	2.28	65
uf20-014	20	91	273	Y	7.07	167
uf20-015	20	91	273	Y	5.67	120
uf20-016	20	91	273	Y	4.77	102
uf20-017	20	91	273	Y	4.22	109
uf20-018	20	91	273	Y	8.06	105
uf20-019	20	91	273	Y	3.56	70
uf20-020	20	91	273	Y	9.17	98
uf20-021	20	91	273	Y	1.92	73
uf20-022	20	91	273	Y	2.48	79
uf20-023	20	91	273	Y	11.53	211
uf20-024	20	91	273	Y	5.28	171
uf20-025	20	91	273	Y	4.39	82
uf20-026	20	91	273	Y	3.50	59
uf20-027	20	91	273	Y	2.49	63
uf20-028	20	91	273	Y	3.38	93
uf20-029	20	91	273	Y	2.24	66
uf20-030	20	91	273	Y	1.96	90

From the above table we can observe that the number of monomials in the orthogonalized formula generally increase, although in some cases this does not hold. Moreover, intermediate formulae contains many more monomials. This turn out to be a general rule in performing such kind of operation. However, there are practical applications where the orthogonal form is of great relevance, and the advantages completely surmount the disadvantage of such size increase.

An example comes from the Reliability theory, in the case we need to compute the fault probability of a system which is made of a connection (serial andnor parallel) of elements whose fault probabilities are known.

8 Conclusions

The orthogonal form of a Boolean formula has remarkable properties. Several hard problems become easy when in orthogonal form. Every logic formula can be transformed in orthogonal form. A general procedure for orthogonalization is developed. 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. Due to relatively novelty of the topic, the presented algorithm can probably still noteworthy improve.

References

[1] A. Blake. Canonical Expressions in Boolean Algebra. Ph.D. thesis, University of Chicago, 1937.

[2] E. Boros, Y. Crama, P. L. Hammer, and M. Saks. A complexity index for Satisfiability Problems. SIAM Journal on Computing, 23:45{49, 1994.

[3] H. Kleine Büning and T. Lettmann. Aussagenlogik: Deduktion und Algorithmen. B.G. Teubner, Stuttgart, 1993. Reprinted in English.

[4] V. Chvatal. Resolution Search. Discrete Applied Mathematics, 73:81{99, 1997.

[5] P.L. Hammer and S. Rudeanu. Boolean Methods in Operations Research and Related Areas. Springer-Verlag, New York, 1968.

[6] K. Iwama. CNF Satisfiability Test by Counting and Polynomial Average Time. SIAM J. Computing 18:385{391, 1989.

[7] D.W. Loveland. Automated Theorem Proving: a Logical Basis. North Holland, 1978.

[8] W.V. Quine. A way to simplify truth fubctions. American Mathematical Monthly, 62:627{631, 1955.

[9] J.A. Robinson. A machine-oriented logic based on the resolution principle. Journal of the ACM, 23{41, 1965.