An Incremental Method for Generating Prime Implicants/Implicates

Alex Kean and George Tsiknis

Technical Report 88-16

July 29th 1988

Department of Computer Science
The University of British Columbia
Vancouver, British Columbia
Canada V6T 1W5

Abstract

Given the recent investigation of *Clause Management Systems(CMSs)* for Artificial Intelligence applications, there is an urgent need for an efficient incremental method for generating prime implicants. Given a set of clauses $\mathcal{F}$, a set of prime implicants Π of $\mathcal{F}$ and a clause C, the problem can be formulated as finding the set of prime implicants for $\Pi \cup \{C\}$. Intuitively, the property of implicants being prime implies that any effort to generate prime implicants from a set of prime implicants will not yield any new prime implicants but themselves. In this paper, we exploit the properties of prime implicants and propose an incremental method for generating prime implicants from a set of existing prime implicants plus a new clause. The correctness proof and complexity analysis of the incremental method are presented, and the intricacy of subsumptions in the incremental method is also examined. Additionally, the role of prime implicants in the *CMS* is also mentioned.

1 Introduction

Traditionally, prime implicants have been used to perform minimization on switching circuits [Biswas'75], [Kohavi'78], [Hwa'74], [Hwang et all'85] and [Rhyne et all'77]. In the realm of Artificial Intelligence applications, the role of prime implicants has also generated a great amount of interest. For instance, in mechanical theorem proving, [Slagle, Chang & Lee'69,'70] introduced the notion of prime consequence (analogous to prime implicants or prime implicates) in consequence-finding using semantic resolution. Also, in the investigation of truth maintenance systems, [Reiter & De Kleer'87] discussed the role of prime implicants as an alternative representation for *Clause Management Systems(CMS).*

In the *Reasoner-CMS* problem solving architecture [Reiter & De Kleer'87], the domain dependent *Reasoner* transmits propositional clauses representing its activities to the domain independent *CMS* . The primary function of the *CMS* is to compute the minimal set of support (section 8) for a given query with respect to the *CMS* database. In this framework, it is appropriate and more efficient for the *CMS* to maintain all the prime implicants of its clauses instead of the clauses themselves. Due to the dynamic nature of the *CMS* , the most complicated, computationally expensive and essential operation is to update the existing database of prime implicants each time a new clause is added. The concern for the expensive updates in *CMS* is our primary motivation in finding an efficient incremental method for generating prime implicants.

Methods for generating prime implicants from Boolean expressions have been studied extensively in the area of switching theory. For example, there is the consensus method [Bartee, Lebow & Reed'62]; the well-known techniques of the Karnaugh Map and the Quine-McCluskey algorithm [Biswas'75, Kohavi'78]; the Semantic Resolution technique explored by [Slagle, Chang & Lee'69,'70]; the elegant Tison's method [Tison'67] etc. It is obvious that all of the conventional methods that generate prime implicants are applicable to the *CMS* update problem. However they are inefficient simply because they are concerned with the generation of prime implicants from an arbitrary Boolean expression. What is needed is an incremental method for generating prime implicants that updates the set of prime implicants when its original corresponding Boolean expression is modified.

More formally, given a Boolean expression $\mathcal{E}_n = C_1 \wedge C_2 \wedge \ldots \wedge C_n$ and its corresponding set of prime implicants denoted by $PI(\mathcal{E}_n)$, then the task can be formulated as computing $PI(\mathcal{E}_{n+1})$ where $\mathcal{E}_{n+1} = \mathcal{E}_n \wedge C_{n+1}$. Obviously, $PI(\mathcal{E}_{n+1})$ can be generated directly from $\mathcal{E}_n \wedge C_{n+1}$.

Unfortunately, this would regenerate a lot of prime implicants that have already been found in $PI(\mathcal{E}_n)$. Ideally, we would like to generate $PI(\mathcal{E}_{n+1})$ from $PI(\mathcal{E}_n) \wedge C_{n+1}$. Again, generating the set of prime implicants from $PI(\mathcal{E}_n) \wedge C_{n+1}$ using the conventional methods results in a lot of redundant computations simply because all the conventional methods do not exploit the fact that the clauses of $PI(\mathcal{E}_n)$ are already prime.

We shall present a new algorithm for generating prime implicants from $PI(\mathcal{E}_n) \wedge C_{n+1}$. There are two criteria for such an algorithm. First, the algorithm should not rely on canonical form[1] of the formula as most of the conventional methods do except [Slagle, Chang & Lee'69,'70] and [Tison'69]. Second, the algorithm should exploit the properties of prime implicants so that the generation of prime implicants will be efficient.

We deem it necessary to indicate at this point that there is not much hope for a "simple" incremental method simply because the PI operator is not monotone. More precisely, there exist sets of prime implicants of P and S such that neither $PI(P \cup S) \subseteq PI(P) \cup PI(S)$ nor $PI(P \cup S) \supseteq PI(P) \cup PI(S)$. As an example, consider $P = \{x\overline{y}, t\}$ and $S = \{t, y\}$, $PI(P \cup S) = \{x, t, y\}$ while $PI(P) = P$ and $PI(S) = S$ and $PI(P) \cup PI(S) = \{x\overline{y}, t, y\}$.

In sections 2 and 3, some preliminary definitions and Tison's Method will be introduced. In Section 4, the incremental algorithm for generating prime implicants from $PI(\mathcal{E}_n) \wedge C_{n+1}$ will be discussed. The proof of correctness of the method is given in section 5. The complexity analysis of the incremental method can be found in section 6, and section 7 describes some futher optimizations of subsumption. The role of prime implicants in *CMS* is discussed in section 8 followed by the conclusions and future work in section 9.

2 Definitions

We shall begin with some definitions and notations that will be used throughout this paper. A variable is denoted by a lowercase letter possibly subscripted(eg. $x, y, z, \ldots, x_1, x_2$). A literal is a positive variable x or a negative variable $\overline{x}$. We call x and $\overline{x}$ a pair of complementary literals.

A clause, denoted by an uppercase letter possibly with subscript, is either a conjunction of literals (Conjunctive Clause) or a disjunction of literals (Disjunctive Clause) without repetition. A conjunctive clause is represented by the juxtaposition of the literals (e.g. $x\overline{y}z$) and a disjunctive

[1]Let S be a set of clauses over a set of variables V. A clause $C \in S$ is said to be in canonical form if every variable in V occurs in C.

clause is connected by the infix operator $\vee$ (e.g. $x \vee \overline{y} \vee z$). We shall refer to a clause as a conjunctive clause unless stated otherwise. Furthermore, for simplicity, a clause is also represented by a set of literals. If $M_1, M_2, \ldots, M_k$ are clauses (or parts of clauses), then for convenience the juxtaposition $M_1 M_2 \ldots M_k$ will represent the clause (or part of the clause) $\bigcup_{i=1}^{k} M_i$.

A Disjunctive Normal Form (DNF) formula is a disjunction of conjunctive clauses and a Conjunctive Normal Form (CNF) formula is a conjunction of disjunctive clauses. By formula we mean a DNF formula unless stated otherwise. The uppercase calligraphic letter possibly subscripted (e.g. $\mathcal{A}, \mathcal{B}, \ldots, \mathcal{Z}$) will be used to denote a formula. For simplicity, a formula is also represented by a set of clauses.

A clause A is said to subsume another clause B or A covers B if every literal in A occurs in B, i.e. $A \subseteq B$. In logical notation, a conjunctive clause A subsumes a conjunctive clause B if $\vdash B \rightarrow A$. For example, $a \wedge b$ subsumes $a \wedge b \wedge c$ because $\vdash a \wedge b \wedge c \rightarrow a \wedge b$. Conversely, a disjunctive clause A subsumes a disjunctive clause B if $\vdash A \rightarrow B$. For instance, $a \vee b$ subsumes $a \vee b \vee c$ because $\vdash a \vee b \rightarrow a \vee b \vee c$. Note that the notion of subsumption or covering in this paper differs from the usual set covering. Intuitively, A subsumes or covers B means A incorporates B. The subsumption or covering relation is transitive, i.e. if A covers B and B covers C, then A covers C.

A clause C is fundamental if C does not contain a complementary pair of literals. For example, the clause $x\overline{y}z$ is fundamental but not $x\overline{y}zy$. Throughout this paper, all clauses will be fundamental unless stated otherwise. Additionally, all definitions and theorems are stated in DNF nevertheless the same notions and results are applicable to CNF by duality.

Definition 1 *Let $A = xA'$ and $B = \overline{x}B'$. The consensus of A and B with respect to the variable x is $CS(A, B, x) = A'B'$ iff $A'B'$ is fundamental.*

The notion of consensus is a restricted type of resolution [Davis & Putnam'60] [Robinson'65]. The restriction is that the resolvent(consensus) must be fundamental. The semantics of fundamentality say that a resolvent that contains a complementary pair is always true(tautology) or false(contradiction) when the resolvent is in CNF or DNF respectively. In resolution, tautology or contradiction is eventually removed as a resolvent thus justifying the restriction on fundamentality.

Definition 2 *Given a conjunctive clause Q and a DNF formula $\mathcal{F}$, Q is an implicant of $\mathcal{F}$ if $\vdash Q \rightarrow \mathcal{F}$. Q is a prime implicant of $\mathcal{F}$ if Q is an implicant of $\mathcal{F}$ and there is no other implicant Q' of $\mathcal{F}$ such that $\vdash Q \rightarrow Q'$.*

Definition 3 *Given a disjunctive clause Q and a CNF formula $\mathcal{F}$, Q is an implicate of $\mathcal{F}$ if $\vdash \mathcal{F} \rightarrow Q$. Q is a prime implicate of $\mathcal{F}$ if Q is an implicate of $\mathcal{F}$ and there is no other implicate Q' of $\mathcal{F}$ such that $\vdash Q' \rightarrow Q$.*

In the design of switching circuits, DNF is the widely accepted representation, therefore the notion of implicant is relevant. On the other hand, in the realm of theorem proving, CNF is the proper representation for refutation therefore the notion of implicate is relevant [Slagle, Chang & Lee'70]. The relationship between consensus and implicant/implicate is stated as the well-known *consensus thereom*, i.e. if Q is the consensus of two conjunctive clauses of a DNF formula $\mathcal{F}$, then Q is an implicant of $\mathcal{F}$. By duality, if Q is the consensus of two disjunctive clauses of a CNF formula $\mathcal{F}$, then Q is an implicate of $\mathcal{F}$.

For example, let the CNF formula $\mathcal{F}$ be $(x \vee z) \wedge (y \vee \overline{z})$. The consensus of $x \vee z$ and $y \vee \overline{z}$ is $Q = x \vee y$ and hence, Q is an implicate of F because $\vdash \mathcal{F} \rightarrow Q$. Conversely, if the DNF formula $\mathcal{F}$ is $(p \wedge q \wedge r) \vee (\overline{p} \wedge q \wedge s)$ then the consensus of $p \wedge q \wedge r$ and $\overline{p} \wedge q \wedge s$ is $Q = q \wedge r \wedge s$, hence Q is an implicant of $\mathcal{F}$ because $\vdash Q \rightarrow \mathcal{F}$.

Finally, given a CNF/DNF formula $\mathcal{F}$, the set Π of the prime implicates/implicants of $\mathcal{F}$ is unique and logically equivalent to $\mathcal{F}$, i.e the disjunction/conjunction of the clauses of Π is logically equivalent to $\mathcal{F}$.

3 Tison's Method

The notion of generalized consensus was first introduced by [Tison'67]. The motivation was to capture a sequence of consensus operations in a unifying framework called generalized consensus.

Definition 4 *Let $\mathcal{A} = A_1 \vee \ldots \vee A_n$ be a DNF formula.*

1. *The variable x is a biform variable in $\mathcal{A}$ if $x \in A_i$ and $\overline{x} \in A_j$ for some i, j.*
2. *The variable x is a monoform variable in $\mathcal{A}$ if $x \in A_i$ for some i and $\overline{x} \notin A_j$ for all j.*
3. *A literal is biform/monoform if its variable is biform/monoform.*

Thus by the above definition each $A_i = B_i M_i$ such that

$$\begin{aligned} B_i &= \text{the conjunction of literals of } A_i \text{ that are biform in } \mathcal{A} \\ M_i &= \text{the conjunction of literals of } A_i \text{ that are monoform in } \mathcal{A}. \end{aligned}$$

Let $M = \bigwedge_{i=1}^{n} M_i$ and call $M = GC(A_1, \ldots, A_n)$ the n-order generalized consensus of $\mathcal{A}$ if

$$B_1 \vee \ldots \vee B_n = True$$

holds irredundantly. Note that if A and B are clauses, $GC(B) = B$ and $GC(A, B) = CS(A, B, x)$ for some biform variable x.

For example, let $\mathcal{A} = \overline{b}\overline{e}f \vee abc \vee \overline{a}d$, then the conjunction of the biform literals are $\overline{b}$, ab and $\overline{a}$ and the monoform literals are $\overline{e}f$, c and d respectively. Since $\overline{b} \vee ab \vee \overline{a} = True$ holds irredundantly, that is the deletion of any B_i invalidates the equation, the generalized consensus of $\mathcal{A}$ is the conjunction of their monoforms $\overline{e}fcd$.

In the following we state some theorems pertaining to generalized consensus which will be used in section 5. The proof of these theorems can be found in [Tison'67] and in [Loui & Bilardi'82].

Theorem 1 *Let $X = GC(T_1, \ldots, T_p)$ and x be a biform variable in $T_1, \ldots, T_p$. Renumbering clauses if necessary, suppose x occurs in $T_1, \ldots, T_m$; $\overline{x}$ in $T_{n+1}, \ldots, T_p$; and neither x nor $\overline{x}$ in $T_{m+1}, \ldots, T_n$, where $1 \leq m \leq n \leq p-1$. There exist m', n', Y, Z such that $m < m' \leq n'+1$, $n' \leq n \leq p-1$, $Y = GC(T_1, \ldots, T_{n'})$, $Z = GC(T_{m'}, \ldots, T_p)$ and $X = CS(Y, Z, x)$.*

This theorem ensures that every generalized consensus of order $n > 2$ can be formed via a sequence of consensus operations of order 2.

Theorem 2 *If formula $\mathcal{F}$ is a set of DNF/CNF clauses then*

1. *each prime implicant/implicate of $\mathcal{F}$ is the generalized consensus of a subset of $\mathcal{F}$.*
2. *if P is the generalized consensus of a subset of $\mathcal{F}$ then P is an implicant/implicate of $\mathcal{F}$.*
3. *if P is an implicant/implicate of $\mathcal{F}$ then there is a prime implicant/implicate of $\mathcal{F}$ that covers P.*

The following Tison's method for generating prime implicants exploits the fact that each biform literal will be used exactly once in the algorithm. Note that a consensus operation is

equivalent to a resolution step cum fundamentality test. Thus Tison's method is similar to [Davis & Putnam'60] computing procedure for quantification theory (DPP) and [Robinson'65] resolution procedure in propositional calculus. In resolution procedure, the search for the empty resolvent is heavily relied on which clauses are selected and in DPP, the resolving (biform) variables play a more important role in selecting the clauses. In the other extreme, Tison's method places the control solely on the set of biform variables. This suggests that Tison's method is very similar to DPP.

Given a DNF Formula $\mathcal{F} = A_1 \vee \ldots \vee A_n$, Tison's method produces a list of all prime implicants of F.

Tison's Method:

Step 1.0 Initially, let L be the list $(A_1, \ldots, A_n)$. Throughout the computation, L is a list of implicants of $\mathcal{F}$ called the implicant list. At the completion of the computation, L is the list of all prime implicants of $\mathcal{F}$.

Step 2.0 For each biform variable x in $(A_1, \ldots A_n)$ perform Step 2.1 and 2.2.

Step 2.1 For every pair of clauses $A_i, A_j \in L$, add to L the consensus of A_i, A_j with respect to x if such a consensus exists.

Step 2.2 Delete from L all clauses Q such that there is a Q' in L and Q' covers Q.

Given the list $(A_1, \ldots, A_n)$ of formula $\mathcal{F}$, Tison's Method generates all and only the prime implicants of $(A_1, \ldots, A_n)$. The proof can be found in [Loui & Bilardi'82].

4 The Incremental Method

In this section, we shall present the extended Tison's method which generates prime implicants incrementally. Let Π be the set of prime implicants of a formula $\mathcal{F}$, C be a new clause and let the set of new implicants $PI(\Pi \cup \{C\})$ be stored in the set Σ. The algorithm is similar to Tison's method with two differences: firstly, the algorithm will only perform consensus with respect to the set of biform literals that occur in the input clause C. Secondly, it will only perform consensus between clauses from Σ and Π but not within the same set Σ or Π.

Incremental Prime Implicant/Implicate Algorithm(IPIA)

Input: A set of prime implicants Π of a formula $\mathcal{F}$ and a clause C.

Output: The set $\Sigma \cup \Pi$ is the set of prime implicants of $\Pi \cup \{C\}$.

Step 1.0 Initialize $\Sigma = \{C\}$. If C is subsumed by some clause in Π delete C and STOP.

Step 2.0 For each biform variable x occurring in C do

Step 2.1 For each $S \in \Sigma$ and $P \in \Pi$ such that S, P have consensus on x do

Step 2.1.1 $T = CS(S, P, x)$

Step 2.1.2 $\Sigma = \Sigma \cup T$.

Step 2.2 Delete any $D \in \Sigma \cup \Pi$ such that there is a $D' \in \Sigma \cup \Pi$ that covers D.

Example 4.1

We will demonstrate the algorithm by the following example. Let $\Pi = \{\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}\}$ of some formula $\mathcal{F}$ and the input clause $C = a\bar{c}t$. Initially, the set Σ contains the input clause C and there are three biform literals in C namely a, $\bar{c}$ and t. Step 2.0 selects the first biform literal a and Step 2.1 selects an element $S \in \Sigma$ which is C and an element $P \in \Pi$ which is $\bar{a}b\bar{c}$. The resulting consensus $b\bar{c}t$ of $a\bar{c}t$ and $\bar{a}b\bar{c}$ is stored in the set Σ. Figure 1 illustrates this relationship, the element $a\bar{c}t$ is connected to the new consensus $b\bar{c}t$, with the prime implicant $\bar{a}b\bar{c}$ attached to the arc. Hence the labelled node is the element from Σ and the labelled arc is the element from Π.

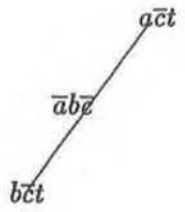

Figure 1.

Since there is no more consensus with repect to the biform literal a, the algorithm proceeds by selecting the next biform variable $\bar{c}$. Again, there is exactly one consensus as illustrated in figure 2.

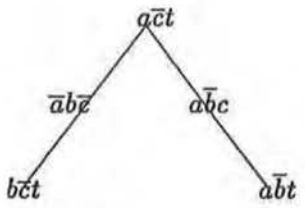

Figure 2.

The next iteration calls for the consensus on the biform variable t. Notice that there are three elements from Σ that have consensus with respect to the bifrom literal t namely, $a\bar{c}t$, $b\bar{c}t$ and $a\bar{b}t$. These elements are the nodes in the tree and hence the algorithm extends the tree in the following sequence. The first node $b\bar{c}t$ is selected and their consensus are shown in figure 3.

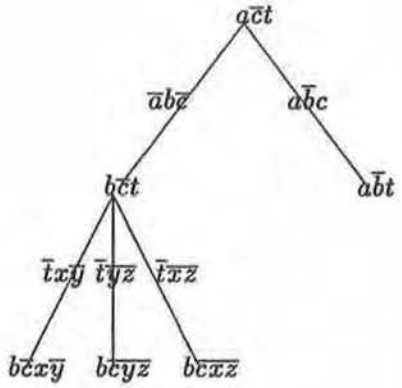

Figure 3.

Next, the second node $a\bar{b}t$ is selected and the resulting consensus are illustrated by the tree shown in figure 4.

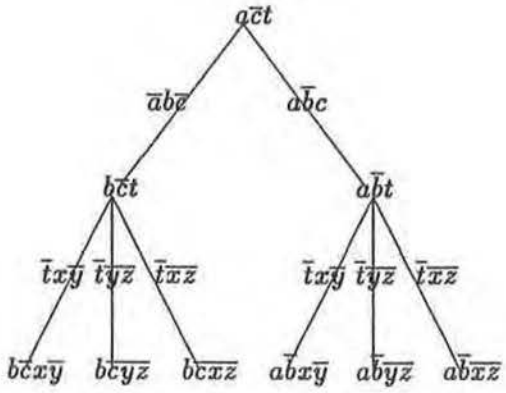

Figure 4.

Finally, the node $a\bar{c}t$ is selected and their corresponding consensus are shown in figure 5.

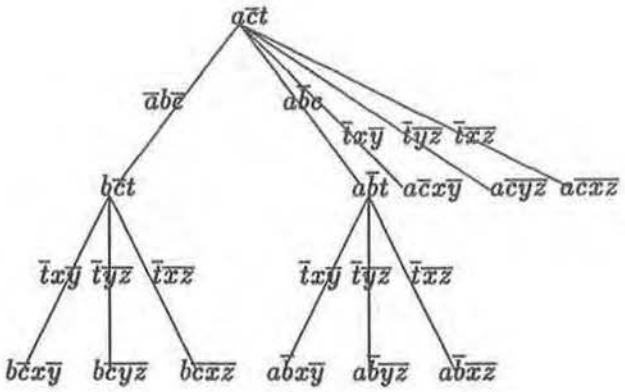

Figure 5.

Notice that there are no subsumptions among them therefore the set $\Sigma \cup \Pi$ after the completion of the algorithm is the set of all prime implicants of $\Pi \cup \{C\}$. The detailed execution of the algorithm for this example is tabulated in appendix A.

This example illustrates that the execution of the incremental algorithm can be represented by a tree whose root is the clause C, with every arc labelled by a clause in Π and every node (except the root) labelled by the consensus of its parent and its associated arc label. Such a tree is called the consensus tree generated from $\Pi \cup \{C\}$ and is denoted by $CTree(\Pi, C)$.

The biform variables of C can be processed by the Step 2.0 of the incremental algorithm in any desired order. Thus at Step 2.0, a specific order is selected and the algorithm proceeds according to this order. We shall call such a selected order the C-variable order. Additionally, the execution of Step 2.0 with respect to the i-th variable in the chosen C-variable order, i.e. all the consensus operations with respect to the i-th variable, is called stage i (or i-th stage) while the sets Σ_i and Π_i denote the sets Σ and Π at the end of stage i.

5 Correctness Proof

The following results are presented for DNF formulae and implicants only. The same results hold for CNF formulae and implicates by duality. First we prove the following lemmata.

Lemma 1 *Let Π be a set of prime implicants of a formula $\mathcal{F}$. Any generalized consensus of a subset of Π is covered by a clause in Π.*

Proof: Let P be a generalized consensus of a subset of Π. By theorem 2, P is an implicant of Π and therefore, there exists a prime implicant P' of Π that covers P. Since Π is a set of prime implicants therefore $P' \in \Pi$. $\square$

Lemma 2 *Given a set of prime implicants* $\Pi = \{P_1, \ldots, P_n\}$ *of a formula* $\mathcal{F}$ *and a clause* $C = x_1 \ldots x_k$. *The set of prime implicants of* $\Pi \cup \{C\}$ *can be generated using Tison's Method by considering only the biform literals that occur in* C.

Proof: Let $v_1, \ldots, v_m, x_1, \ldots, x_k$ be all the biform literals that occur in $\Pi \cup \{C\}$ such that each biform literal $x_i, 1 \leq i \leq k$ occurs in C and each biform literal $v_j, 1 \leq j \leq m$ does not occur in C. The key observation is that Tison's Method is correct independent of the ordering in which the biform literals are considered in Step 2.0 (of Tison's Method). Thus if we adopt the ordering $v_1, \ldots, v_m, x_1, \ldots, x_k$ and after the biform literals $v_1, \ldots, v_m$ have been used by Step 2.0 in Tison's Method, there are no new clauses generated nor old ones being deleted. This is simply because any pair of clauses considered for consensus so far, that is a pair of clauses that contain biform literal v_j, must come from the set Π. Since Π is a set of prime implicants, their consensus must be covered by another prime implicant $P \in \Pi$ by lemma 1. Consequently only the biform literals $x_1, \ldots, x_k$ that occur in C can contribute to generating new prime implicants and subsuming old ones. □

A direct consequence of this lemma is the following corollary.

Corollary 1 *Let* Π *be a set of prime implicants of a formula* $\mathcal{F}$ *and* $C = x_1 x_2 \ldots x_k$ *a clause. If* T *is a generalized consensus of a subset of* $\Pi \cup \{C\}$ *such that some of its biform variables do not occur in* C, *then* T *is subsumed either by a clause in* $\Pi \cup \{C\}$ *or by a generalized consensus of a subset of* $\Pi \cup \{C\}$ *all of whose biform variables occur in* C.

Proof: Assume that Tison's method has been used as described in lemma 2. The correctness of Tison's method ensures that at termination there is a clause T' that covers T. But T' is either a clause of $\Pi \cup \{C\}$ or it has been generated by a sequence of consensus operations with respect to the biform variables in C (by lemma 2). In this case T' is a generalized consensus of clauses in $\Pi \cup \{C\}$ in which all the biform variables occur in C. □

The previous lemma and corollary justifies Step 2.0 in the incremental algorithm where only the biform literals $x_1, \ldots, x_k$ can contribute to generating new prime implicants and subsuming old ones.

Lemma 3 *Let* $C = x_1 x_2 \ldots x_k$ *be a clause,* Π *a set of prime implicants of a formula* $\mathcal{F}$ *and* $\{P_1, \ldots, P_n\}$ *a subset of* Π *such that* $T = GC(P_1, \ldots, P_n, C)$ *exists. If the inputs to the incremental algorithm are* Π *and* C, *then at completion there is a* T' *in* $\Sigma_k \cup \Pi_k$ *that covers* T.

Proof: According to corollary 1, we only need to prove the lemma for subsets of $\Pi \cup \{C\}$ whose biform variables are among $x_1, \ldots, x_k$. Let $P_1, \ldots, P_n, C$ be such a subset and t, $0 \leq t \leq k$ be the smallest natural number such that all the biform variables of $P_1, \ldots, P_n, C$ are among $x_1, \ldots, x_t$. We shall prove by induction on t such that at stage t and subsequent stages, the set $\Sigma_t \cup \Pi_t$ contains a clause that covers T.

If $t = 0$ then $n = 0$, $T = GC(C) = C$ and $\Sigma_0 = \{C\}$. Hence C in Σ_0 covers T.

Suppose $t \geq 1$ and the statement is true for any natural number $< t$. Renumbering the clauses if necessary, suppose that x_t occurs in $P_1, \ldots, P_m$, $\overline{x_t}$ occurs in $P_{l+1}, \ldots, P_n$ and neither x_t nor $\overline{x_t}$ occurs in $P_{m+1}, \ldots, P_l$, where $0 \leq m \leq l \leq n-1$. By theorem 1, there exist m', l', Y, Z such that $m < m' \leq l' + 1$, $l' \leq l \leq n-1$, $Y = GC(C, P_1, \ldots, P_{l'})$, $Z = GC(P_{m'}, \ldots, P_n)$ and $T = CS(Y, Z, x_t) = (Y - \{x_t\}) \cup (Z - \{\overline{x_t}\})$.

We want to show that at the end of stage t there is a clause in $\Sigma_t \cup \Pi_t$ that covers T. By the inductive hypothesis the set $\Sigma_{t-1} \cup \Pi_{t-1}$ at stage $t-1$ contains a clause Y' that covers Y and a clause Z' that covers Z.

1. If $x_t \notin Y'$ then Y' covers T and similarly if $\overline{x_t} \notin Z'$ then Z' covers T.

2. Assume that $x_t \in Y'$ and $\overline{x_t} \in Z'$.

   (a) Evidently, Z' simply cannot be in Σ_{t-1}, otherwise Z' will be the result of a sequence of consensus of C and some clauses from Π with respect to variables in $\{x_1, \ldots, x_{t-1}\}$. Consequently x_t is in Z'. Since $\overline{x_t}$ is also in Z', this contradicts the fact that Z' is fundamental.

   (b) If $Y' \in \Sigma_{t-1}$ and $Z' \in \Pi_{t-1}$ then at stage t, $T' = CS(Y', Z', x_t) = Y' - \{x_t\} \cup Z' - \{\overline{x_t}\}$ in Σ_t covers T.

   (c) If both Y' and Z' are in Π_{t-1} then by lemma 1 there is a T' in Π_{t-1} and obviously in Π_t that covers $CS(Y', Z', x_t)$ which in turn covers T.

Furthermore, since covering is transitive, if $T' \in \Sigma_i \cup \Pi_i$ covers T at stage i, then for any $j \geq i$, the set $\Sigma_j \cup \Pi_j$ also contains a clause that covers T. □

In the proof, cases (a), (b) and (c) justify Step 2.1 in the incremental algorithm saying that we need only to consider consensus of a clause S from Σ and another clause P from Π. We never consider the consensus of two clauses from Σ and never need to consider two clauses from Π.

Theorem 3 (Correctness) *Let Π be a set of prime implicants of a formula $\mathcal{F}$ and $C = x_1 \dots x_k$ a clause. After the completion of the incremental algorithm with input Π and C, the set $\Sigma_k \cup \Pi_k$ contains all and only the prime implicants of $\Pi \cup \{C\}$.*

Proof: *All.* Let P be a prime implicant of $\Pi \cup \{C\}$. By theorem 2, $P = GC(\Phi)$ for some subset Φ of $\Pi \cup \{C\}$. If Φ is a subset of Π alone, then by lemma 1 there is a clause $P' \in \Pi$ that covers P. Since P' is deleted only when there is another clause in $\Sigma \cup \Pi$ that covers it, therefore at termination $\Sigma_k \cup \Pi_k$ contains a clause that covers P simply by the transitivity of covering. On the other hand, if $C \in \Phi$ then by lemma 3, $\Sigma_k \cup \Pi_k$ contains a clause that covers P. In both cases, since P is a prime implicant, the only cover for P is itself, consequently $P \in \Sigma_k \cup \Pi_k$.

Only. Let $P \in \Sigma_k \cup \Pi_k$ and by theorem 2 and lemma 3, P is an implicant of $\Pi \cup \{C\}$. Assume that P is not a prime implicant hence there is a prime implicant P' that covers P. By the previous part, $P' \in \Sigma_k \cup \Pi_k$, consequently P is deleted at Step 2.2 in the incremental algorithm, therefore $P \notin \Sigma_k \cup \Pi_k$. □

6 Complexity Analysis

The present section is devoted to the issues concerning the complexity of the incremental algorithm. We concentrate on the worst case time complexity only, which is calculated in terms of the number of consensus and subsumptions performed.

For the rest of this section we assume that the input to the algorithm consists of the set of prime implicants Π of a formula $\mathcal{F}$ and the clause $C = x_1 x_2 \dots x_k$, where $x_i, 1 \leq i \leq k$ are distinct literals. Moreover the cardinality of Π is assumed to be $|\Pi| = n$. First we prove the following lemma.

Lemma 4 *Each clause $P \in \Pi$ is used in at most one stage of the incremental algorithm.*

Proof: Let $P \in \Pi$ and if P used in more than one stage, P should contain more than one literal complementary to some x_i. We assume that $P = \overline{x}_{i_1} \dots \overline{x}_{i_l} M$ where $i_1 \leq i_2 \leq \dots \leq i_l$ and $1 \leq i_j \leq k$ for $1 \leq j \leq l$, and M is the monoform of P with respect to C.

At each stage $m, 1 \leq m \leq k$, every clause in Σ contains at most the literals $x_m, x_{m+1}, \dots, x_k$. P cannot be used at any stage $m < i_l$ simply because it does not have consensus on x_m with any clause in Σ, i.e. P contains more than one pair of complementary literals with respect to

any clause in Σ. Obviously, P also cannot be used at any stage $m > i_l$ because there is no complementary literal. Evidently, P may only be used at stage $m = i_l$. $\square$

The following theorem estimates the complexity of the algorithm.

Theorem 4 *Given a set of prime implicants Π of a formula $\mathcal{F}$ and a clause C, the incremental algorithm requires at most $O((\frac{n}{k})^{2k})$ operations (consensus and subsumptions), where $n = |\Pi|$ and k is the number of biform variables in C.*

Proof: We assume that every literal $x_1, \ldots, x_k$ in C is a biform literal. Let Π_i, $1 \leq i \leq k$, be the set of the clauses of Π used at the stage i and $|\Pi_i| = n_i$. Firstly, we will calculate the maximum number of consensus operations required. If m_i, $1 \leq i \leq k$, denotes the maximum number of clauses in Σ_i at the end of the stage i then

$$m_1 = n_1 + 1$$

and

$$m_i = m_{i-1} + m_{i-1}n_i = m_{i-1}(n_i + 1)$$

for $2 \leq i \leq k$. Consequently, at most $O(n_1 n_2 \cdots n_k)$ new clauses have been generated at the end of k-th stage. Since each clause is generated by one consensus operation, the upper bound $O(n_1 n_2 \cdots n_k)$ also represents the maximum number of consensus operations required by the algorithm. Furthermore, by assuming that every clause in Π is used at some stage, then by lemma 4 we have

$$n_1 + n_2 + \cdots + n_k = n$$

or with equal distributuion,

$$n_i = \frac{n}{k}, \ for \ 1 \leq i \leq k$$

As a result, the number of the consensus operations, as well as the number of clauses in Σ, is at most $O((\frac{n}{k})^k)$.

The number of required subsumption operations can be easily estimated by observing that every clause in Σ should be checked for subsumption against every other clause in Σ, i.e. $O((\frac{n}{k})^{2k})$, as well as against every clause in Π, i.e. $O(n(\frac{n}{k})^k)$. Consequently, the number of subsumptions performed is at most

$$O((\frac{n}{k})^{2k} + n(\frac{n}{k})^k)$$

If $\log n > \frac{k}{k-1} \log k$, a relation that is true in most applications, then the overall time complexity of the algorithm is simply $O((\frac{n}{k})^{2k})$. $\square$

The last result shows that the algorithm is exponential in time. Many optimizations are applicable here (more on section 7) but they cannot reduce the complexity class of the algorithm mainly because the time complexity of the problem itself is exponential. More precisely, given a set of prime implicants Π of a formula $\mathcal{F}$ and a clause C, the number of the prime implicants of $\Pi \cup \{C\}$ is potentially exponential on the size of Π [Chandra & Markowsky'78], as illustrated in the following example.

Example 6.1 *Let* $C = \overline{a}_1 \ldots \overline{a}_k$ *and* $\Pi = \bigcup_{i=1}^{k} \Pi_i$. *For each* i, $1 \leq i \leq k$, $\Pi_i = \{a_i s_{i1}, \ldots, a_i s_{im}\}$ *and* s_{ij}, $1 \leq i \leq k$, $1 \leq j \leq m$, *are new pairwise distinct variables, different from any* a_i, $1 \leq i \leq k$.

Evidently, Π is a set of prime implicants since neither consensus nor subsumption exists among any pair of its clauses. Moreover, the only subsets of $\Pi \cup \{C\}$ that have generalized consensus are the subsets $\{P_1, \ldots, P_k, C\}$ of size greater than one that contain C and each $P_i \in \Pi_i$. Hence every such consensus $G = GC(P_1, \ldots, P_k, C)$ neither subsumes nor is subsumed by any other clause in $\Pi \cup \{C\}$ and other existing consensus simply because the monoform of G comprises of new pairwise distinct variables that are not compatible for subsumption. Consequently, G constitutes a new prime implicant of $\Pi \cup \{C\}$ and the total number of new prime implicants of $\Pi \cup \{C\}$ is $(m+1)^k$ or in O-notation, if $n = mk = |\Pi|$, then we obtain $O((\frac{n}{k})^k)$.

7 Subsumption and Optimization

In section 5, theorem 3 indicates that subsumption is a necessary operation in order to guarantee the correctness of the incremental algorithm. Unfortunately, performing subsumptions for a set of clauses can be quite expensive. Natually one would question whether the total number of subsumptions can be reduced and still preserve the correctness. In this section we will explore some optimizations of subsumption and reveal some interesting properties of subsumptions in the incremental algorithm.

The alert reader will notice that during the stages of the incremental algorithm, each $S \in \Sigma$ can potentially have consensus with a set of $P \in \Pi$ with respect to the same biform literal. As seen in section 4, this feature can be illustrated as a subtree of a consensus tree (CTree) in which

the parent node is labelled with the clause S, and the set of branches is labelled with each prime implicant P_i respectively. Furthermore, their corresponding consensus nodes S_i are attached at the end of the arcs as brothers.

One property of subsumption in this subtree is that whenever a consensus S_i covers the parent node S, this consensus S_i will cover all of its brothers $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n$. This local property does save some consensus operations in the average case because as soon as S is subsumed by some S_i, no further consensus need be performed for S. Also note that this property is not a necessary condition for the incremental algorithm since Step 2.2 will eventually detect the subsumption. Assuming Σ and Π as defined in the previous section, we have the following lemmata.

Lemma 5 *Given an $S \in \Sigma$ and $P_1, \ldots, P_n \in \Pi$. Let $S_1, \ldots, S_n$ be their corresponding consensus where $S_i = CS(S, P_i, x)$, $1 \leq i \leq n$. If S_i covers S then S_i covers $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n$.*

Proof: Let $S = xM_s$ and $P_1 = \overline{x}M_1, \ldots, P_n = \overline{x}M_n$ where x and $\overline{x}$ are the biform literals and $M_s, M_1, \ldots, M_n$ are all monoforms. Thus their corresponding consensus are $S_1 = M_sM_1, \ldots, S_i = M_sM_i, \ldots, S_n = M_sM_n$. Assume that S_i covers S hence $M_sM_i \subseteq M_s$ or simply $S_i = M_s$. Therefore S_i covers $M_sM_1, \ldots, M_sM_{i-1}, M_sM_{i+1}, \ldots, M_sM_n$. $\square$

Example 7.1 *Let $S = x\overline{u}v\overline{w}$ and let $P_1 = \overline{x}v\overline{w}$, $P_2 = \overline{x}y\overline{w}$ and $P_3 = \overline{x}up\overline{w}$. Their corresponding consensus are shown in the following tree. Notice that the consensus $\overline{u}v\overline{w}$ covers the parent $x\overline{u}v\overline{w}$ and subsequently, covers $\overline{u}vy\overline{w}$ and $\overline{u}vp\overline{w}$.*

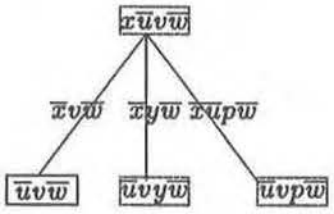

Figure 6.

A similar observation can be made with regard to subsumption across different S_i that have consensus with the same prime implicant P within the same stage. Given a set of parents $S_1, \ldots, S_n \in \Sigma$ and a prime implicant $P \in \Pi$. Their corresponding consensus are $S_1', \ldots, S_n'$ where each $S_i' = CS(S_i, P, x)$. If there exists an S_j' that covers P, then this S_j' covers $S_1', \ldots, S_{j-1}'$, $S_{j+1}', \ldots, S_n'$. Again, this local property says that if the prime implicant $P \in \Pi$ is subsumed,

then P can be deleted immediately. Since the same prime implicant P potentially can have consensus with many S_i within the same stage, the deletion of P as soon as it is subsumed will on average save some consensus operations. This property is also not a necessary condition for the incremental method because Step 2.2 will eventually delete P as well as S_i', $i \in \{1, \ldots, j-1, j+1, \ldots, n\}$.

Lemma 6 *Given $S_1, \ldots, S_n \in \Sigma$ and a prime implicant $P \in \Pi$. Let $S_1', \ldots, S_n'$ be their corresponding consensus where $S_i' = CS(S_i, P, x), 1 \leq i \leq n$. If S_j' covers P then S_j' covers $S_1', \ldots, S_{j-1}', S_{j+1}', \ldots, S_n'$.*

Proof: Let $P = \overline{x}M_p$ and $S_1 = xM_1, \ldots, S_n = xM_n$ where x and $\overline{x}$ are biform literals and $M_p, M_1, \ldots, M_n$ are all monoforms. Their corresponding consensus are $S_1' = M_pM_1, \ldots, S_j' = M_pM_j, \ldots, S_n' = M_pM_n$. Assume that S_j' covers P hence $M_pM_j \subseteq M_p$ or simply $S_j' = M_p$. Therefore S_j' covers $M_pM_1, \ldots, M_pM_{j-1}, M_pM_{j+1}, \ldots, S_n' = M_pM_n$. □

Example 7.2 *Let $P = xab$ and let $S_1 = \overline{x}a$, $S_2 = \overline{x}u$ and $S_3 = \overline{x}w$. Their corresponding consensus are shown in the following trees. Notice that the consensus ab covers the prime implicant xab and subsequently, covers abu and abw.*

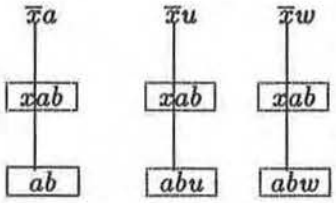

Figure 7.

After observing these local properties of subsumption, naturally one would question whether there exists some global properties of subsumption that allow some reduction in the number of subsumptions. Ideally, one would be willing to sacrifice a constant amount of time to preprocess the set of clauses such that the expensive subsumptions of Step 2.2 in the incremental algorithm could be replaced by lemma 5 and 6 — hereafter called local subsumption check — and still preserve the correctness of the algorithm.

For this reason, we now examine the prospect of precomputing an ordering of the biform literals in C such that it is sufficient to replace Step 2.2 in the incremental algorithm by the local subsumption check. Unfortunately, the following counter-example shows that the local

subsumption check is insufficient. That is to say, no matter what the ordering of the biform literals, there exists subsumptions not detected solely by lemma 5 and 6.

Example 7.3 *Let* $C = acd$ *and* $\Pi = \{\overline{a}xy, \overline{a}p, \overline{c}x, \overline{c}pq\}$.

The biform variables of C are a and c. Figure 8 shows the tree that is generated by the algorithm when the ordering a, c is followed. The reader can notice that the clause dpq in one subtree subsumes $adpq$ in the other subtree and neither lemma 5 nor lemma 6 detects this.

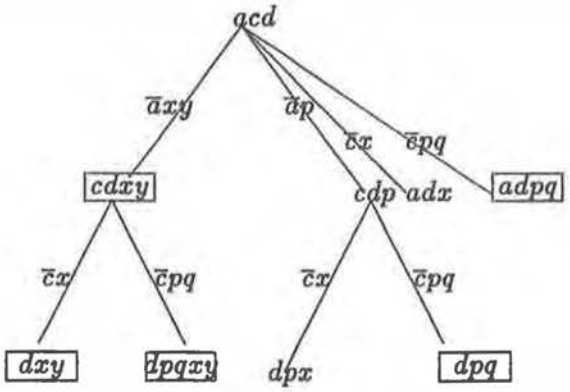

Figure 8: Example 7.3 with ordering a, c.

On the other hand, if the ordering c, a is used, the new tree shown in figure 9 also exhibits a subsumption across subtrees, i.e. the clause dxy in one subtree subsumes the clause $cdxy$ in another subtree and similarly, no local subsumption check detects this either.

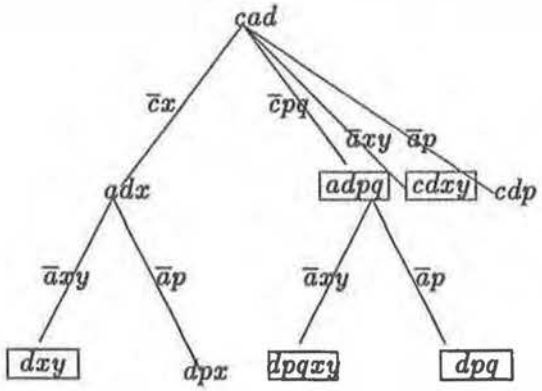

Figure 9: Example 7.3 with ordering c, a.

The above counter-example indicates that in general, no C-variable order will satisfy our goal in replacing Step 2.2 by the local subsumption check. Nevertheless there are cases in which certain orderings are preferable over the others. Recall that the input to the algorithm consists of a set of prime implicants Π and a clause C.

Initially, suppose there is a clause $P \in \Pi$ such that $CS(C, P, x) = C_1$ and C_1 subsumes C for some variable x that occurs in C. Let x be the first biform variable in the ordering according to step 2.0. By lemma 5, the whole subtree are subsumed and C_1 becomes the new root of the consensus tree and the stage that corresponds to x terminates immediately.

This process which can be repeated as long as the above condition holds for the new root is called root optimization. When a stage is reached such that no further root optimization can be applied, the incremental algorithm is resumed with the new root as input clause and the remaining biform variables with respect to C. While each root optimization is relatively inexpensive ($O(\log n)$ where $n = |\Pi|$ with a suitable indexing scheme), it may account for a significant overall saving. More precisely, if C contains k biform variables and m root optimizations can be performed, where $m \leq k$, the complexity of the algorithm is reduced to $O((\frac{n}{k})^{2(k-m)})$.

Finally, there is the case where subsumption occurs among clauses introduced at different stages. We claim that if the algorithm performs local subsumption checking, the only subsumption that might exist (and must be examined) at the end of each stage are subsumptions among the new clauses generated at that stage. In other words, if local subsumption has been checked, there is no subsumption among the clauses that have been generated at different stages. The optimization the above claim suggests is called inter-stage optimization. We feel that it will serve the reader better if we first present the optimized algorithm and give the proof of the above claim afterwards. The optimized algorithm is similar to the original one but, it incorporates the root optimization, the local subsumption check and the inter-stage optimization.

Optimized IPIA

Input: A set of prime implicants Π of a formula $\mathcal{F}$ and a clause C.

Output: The set $\Sigma \cup \Pi$ is the set of prime implicants of $\Pi \cup \{C\}$.

Step 1.0 *Root optimization.*

Step 1.1 Delete any $D \in \Pi \cup \{C\}$ such that there is a $D' \in \Pi \cup \{C\}$ that subsumes D.

Step 1.2 If C has been deleted, STOP.

Step 1.3 If there is a clause $P \in \Pi$ such that $CS(C, P, x) = C'$ for some biform variable x in C and C' subsumes C then set $C = C'$ and repeat Step 1.0.

Step 2.0 Set $\Sigma = \{C\}$.

Step 3.0 For each biform variable x in C do

Step 3.1 Set $\Sigma\_Children = \emptyset$.

Step 3.2 For each clause S in Σ do

Step 3.2.1 *Lemma 5*: If $CS(S, P, x) = S'$ for some $P \in \Pi$ and S' subsumes S then delete S from Σ and set $S\_Children = \{S'\}$
else set $S\_Children = \{CS(S, P, x) \mid P \in \Pi\}$.

Step 3.2.2 *Lemma 6*: Delete any $D \in S\_Children \cup \Pi$ such that there is a $D' \in S\_Children \cup \Pi$ that subsumes D.

Step 3.2.3 Add the remaining $S\_Children$ to $\Sigma\_Children$.

Step 3.3 Delete any $D \in \Sigma\_Children$ such that there is a $D' \in \Sigma\_Children$ that subsumes D (subsumption among children of the same parent need not be checked).

Step 3.4 Add the remainder of $\Sigma\_Children$ to Σ.

Step 1.0 in the optimized algorithm corresponds to the root optimization discussed in this section. Steps 3.2.1 and 3.2.2 perform the local subsumption check while Step 3.3 takes into account the inter-stage optimization and its correctness is assured by the following lemma.

Lemma 7 *Whenever control reaches the end of step 3.3 of the optimized algorithm, no subsumption relation exists between any two clauses in* $\Sigma\_Children \cup \Sigma$.

Proof: We assume that at Step 2.0, $C = x_1 x_2 \ldots x_k$, for each x_i, $1 \leq i \leq k$, is a biform literal and the C-variable order in which the biform variables are considered at Step 3.0 is $x_1, x_2, \ldots, x_k$. Again by *stage* i we mean the execution of Step 3.0 with respect to x_i.

We will now prove the lemma by induction on stage i, $1 \leq i \leq k$. For $i = 1$, all the clauses in $\Sigma\_Children$ come from the same parent. Therefore, at Step 3.2 all the subsumptions among them and their parent have been eliminated.

Assume the lemma is true for any stage $< i$. Suppose at the end of the i-th stage there exist two clauses A and B in $\Sigma\_Children \cup \Sigma$ such that either A subsumes B or B subsumes A. A and B cannot both be in Σ since this contradicts the inductive hypothesis. Furthermore, neither A and B can both be in $\Sigma\_Children$ because if this were the case, one of them would have been deleted at Step 3.3.

Assume, without loss of generality, that $A \in \Sigma\_Children$ and $B \in \Sigma$. In this case, $B = x_i x_{i+1} \ldots x_k M_2$ and there exist $A' \in \Sigma$ and $P \in \Pi$ such that $A' = x_i x_{i+1} \ldots x_k M_1$, $P = \overline{x}_i F M_3$ and $A = CS(A', P, x_i)$ or $A = x_{i+1} x_{i+2} \ldots x_k M_1 M_3$; where, $F \subseteq \{x_{i+1}, \ldots, x_k\}$ and $M_j \cap \{x_i, x_{i+1}, \ldots, x_k\} = \emptyset$ for $j = 1, 2, 3$.

Note that at stage i any clause in Σ contains at least $x_i x_{i+1} \ldots x_k$. Consequently, B can not subsume A since $x_i \in B$ but $x_i \notin A$. On the other hand, if A subsumes B then $M_1 M_3 \subseteq M_2$, which implies that $M_1 \subseteq M_2$ and A' subsumes B. If A' and B are different clauses, then this contradicts the inductive hypothesis; otherwise, B is eliminated at Step 3.2.1. □

Theorem 5 (Correctness of the Optimized IPIA) *Given a set of prime implicants* Π *of a formula* $\mathcal{F}$ *and a clause* C*, After the completion of the Optimized IPIA, the set* $\Sigma_k \cup \Pi_k$ *contains all and only the prime implicants of* $\Pi \cup \{C\}$.

Proof: Theorem 5 is a direct consequence of theorem 3, lemma 5, 6 and 7. □

The complexity of the optimized algorithm is $O((\frac{n}{k})^{2k})$ where $n = |\Pi|$ and k is the number of the biform variables of C that survive the root optimization. Obviously, the new algorithm is in the same complexity class with its predecessor although its average complexity is expected to be lower than the average complexity of the previous algorithm. The explosion in complexity comes from the fact that at each stage, the same clause in Π is used with many clauses in Σ to generate consensus which may get deleted later at Step 3.3.

It would be of great advantage if there were a way to detect in advance which consensus are bound to be deleted. Alas, such a test will inevitably have the same complexity as the generation of the consensus and subsumption check. Consider two clauses S_1 and S_2 in Σ at stage i such that $S_1 = x_i x_{i+1} \ldots x_k M_1$, $S_2 = x_i x_{i+1} \ldots x_k M_2$ and have consensus with the clause $P = \overline{x}_i F M_3$, where F, M_1, M_2, M_3 are as in the proof of lemma 7. In this case $CS(S_1, P, x_i)$ subsumes $CS(S_2, P, x_i)$ iff $M_1 M_3 \subseteq M_2 M_3$. Since $M_1 \not\subseteq M_2$ and $M_2 \not\subseteq M_1$ the subsumption relation among the two consensus cannot be detected by considering S_1 and S_2 alone. A similar argument can be made for the case in which different clauses in Π are used. As a concluding remark, we would like to point out that as a consequence of the above observation, there is not much hope for further optimization of the incremental algorithm.

8 Clause Management System(CMS)

In this section, we will outline how the incremental method is used in the *Clause Management System* environment. Throughout this section, a formula will denote a CNF formula and a clause will denote a disjunctive clause. In [Reiter & De Kleer'87], a problem solving environment consists of a domain dependent *Reasoner* and a domain independent *Clause Management System(CMS).* The *Reasoner* occasionally transmits a clause (it may be a First Order formula) that describes some of its activities. The *CMS* records this clause as a propositional clause (different atomic formulae correspond to different propositional variables) if it is fundamental, i.e. not tautologous; otherwise the *CMS* discards it. In addition, the *Reasoner* can query the *CMS* whenever is required. The query consists of a propositional clause G and the *CMS* must respond with every minimal clause S such that $S \vee G$ is a fundamental logical consequence of the clauses so far transmitted to the *CMS* by the *Reasoner* , i.e. the *CMS* database. Such a clause S is called the minimal fundamental support for G with respect to the *CMS* database.

There are many applications using the *Reasoner-CMS* architecture. For example, [Reiter & De Kleer'87] present how abductive reasoning can be accomplished in the *CMS* paradigm and how searching among alternatives in the search space can be facilitated by the *CMS* . In addition, [De Kleer & Williams'87] demonstrate the use of *Reasoner-ATMS* (a special kind of *CMS*) architecture in diagnositic reasoning.

We will illustrate the *Reasoner-CMS* cooperation by an example taken from [Reiter & De Kleer'87]. Consider a reasoning system with knowledge base KB and assume that the Reasoner in its attempt to prove g has discovered that

$$\begin{aligned} KB &\models p \wedge q \wedge r \rightarrow g \\ KB &\models \neg p \wedge q \rightarrow g \\ KB &\models \neg q \wedge r \rightarrow g. \end{aligned}$$

Thus, the *Reasoner* transmits to the *CMS* the clauses $\overline{p} \vee \overline{q} \vee \overline{r} \vee g$, $p \vee \overline{q} \vee g$ and $q \vee \overline{r} \vee g$. Suppose now that the *Reasoner* is interested in finding the minimal explanation for g. By quering the *CMS* with g it obtains the minimal support for g namely $\{p \vee \overline{q}, \overline{r}\}$. This in turn implies that a minimal explanation for g is either $\overline{p} \wedge q$ or r since $KB \models \overline{p} \wedge q \rightarrow g$ and $KB \models r \rightarrow g$.

Definition 5 *Let Σ be a set of clauses and G a single clause. A clause S is a fundamental support (or support) for G with respect to Σ if*

1. $\Sigma \models S \cup G$.

2. $\Sigma \not\models S$.

3. *$S \cup G$ is fundamental.*

A clause S is a minimal fundamental support (or minimal support) for G with respect to Σ if S is a support for G and there is no other support S' for G such that $\models S' \rightarrow S$.

Note that the definitions of support and minimal support differ from the corresponding definitons given in [Reiter & De Kleer'87] in two respects. Firstly, we insist that a support clause S for G must have an additional property namely, $S \cup G$ is fundamental. Secondly, the minimality is defined with respect to a different ordering among the clauses. According to [Reiter & De Kleer'87], if A and B are clauses, $A \leq B$ iff every literal in A is also in B. According to our definition, $A \leq B$ iff $\models A \rightarrow B$. Consequently given a clause G, any trival support S for G, i.e. $S \cup G$ is a tautology, is not considered as a minimal support. The set of trivial supports for G, i.e. all tautologies that include G, can be easily generated by the *Reasoner* , therefore the *CMS* database should not include the rather large set of trivial supports.

It can be shown that the set of minimal supports for a query G can be computed trivially from the set of prime implicates of the *CMS* database [Reiter & De Kleer'87] [Tsiknis & Kean'88]. More formally, if Σ denotes the *CMS* database and G is the query clause, then the set of support for G, $\Delta(\Sigma, G)$ is defined as

$$\Delta(\Sigma, G) = \{P - G \mid P \in PI(\Sigma) \textit{ and } P \cap G \neq \emptyset \textit{ and } P \cup G \textit{ is fundamental}\},$$

and the set of minimal support for G is defined as

$$\Gamma(\Sigma, G) = \{S \mid S \in \Delta(\Sigma, G) \textit{ and no } S' \in \Delta(\Sigma, G) \textit{ covers } S\}.$$

Since the set $PI(\Sigma)$ and Σ are logically equivalent, the *CMS* may choose to represent the set Σ as it is, the Simple-DB approach, or with a little more effort and memory compute and retain the set $PI(\Sigma)$ on-the-fly, the PI-DB approach.

Under the Simple-DB approach, the *CMS* stores the set of clauses transmitted by the *Reasoner* in its database without any alteration. Updating the *CMS* 's database Σ is trivially simple, that is $\Sigma = \Sigma \cup G$. Nevertheless the query processing is extremely expensive merely because the set $PI(\Sigma)$, Δ and Γ must be computed for every different query G. Note that computing the set

$PI(\Sigma)$ is most expensive. Fortunately once the set $PI(\Sigma)$ is available, the set Δ and Γ can be computed very efficiently by using special indexing and ordering schemes on $PI(\Sigma)$.

Naturally, the PI-DB approach is aimed at minimizing the expensive computation of the set $PI(\Sigma)$ by computing it incrementally. Thus under the PI-DB approach, the *CMS* stores the set of prime implicates of the clauses it has received so far, in contrast with the Simple-DB approach. When a new clause L is transmitted by the *Reasoner* to the *CMS* , the *CMS* computes and stores $PI(\Sigma \cup L)$ using the incremental method described in this paper. As a consequence, the query processing for minimal support can be achieved very efficiently while updating the *CMS* database is also relatively efficient using the incremental algorithm.

In the actual modelling of a *Reasoner-CMS* architecture, one must be cautious about the tradeoff between the Simple-DB and PI-DB approaches. If the *CMS* task is to perform large numbers of updates, then the Simple-DB approach is superior simply because updates in Simple-DB approach take constant time. Conversely, if the *CMS* task is heavily related to query processing, that is computing minimal support, then the Simple-DB approach will require exponential time to compute the set of $PI(\Sigma)$ and also exponential space to store the set of $PI(\Sigma)$ in order to allow the computation of the minimal set of support. In contrast, the PI-DB approach requires only linear time and space in query processing with respect to the size of the PI-DB database.

It is important to note that the size of the PI-DB database can be exponential, that is the number of prime implicates is potentially exponential [Chandra & Markowsky'78]. Consequently, the PI-DB approach potentially needs exponential space to store the prime implicates, but this is also the case for the Simple-DB approach each time a query is processed. The difference is simply that the Simple-DB does not retain the exponential space after it is used but requires heavy recomputation whenever it is needed and conversely, the PI-DB approach uses exponential space but recomputation is kept to a minimum.

In a future paper, we study the full extent of the *Reasoner-CMS* architecture and show that all the theorems in [Reiter & De Kleer'87] hold modulo fundamentality. Additionally, we argue that the PI-DB approach is more suitable for *CMS* in both question-answering and explanation-based problem solving environments [Tsiknis & Kean'88].

9 Conclusions

We have presented an incremental algorithm for generating prime implicants/implicates of a set of clauses. We have prove the correctness of this algorithm and analyzed its complexity. Although the incremental algorithm can be used to generate the prime implicants/implicates of a given set of clauses by incrementally considering one clause at a time, nevertheless it is best suited for situation where new clauses are frequently added over the period in consideration. Moreover, this algorithm, in contrast with previous algorithms for the minimization of Boolean functions domain, does not rely on a canonical form representation of the clauses. This latter feature makes it attractive for many applications in Artificial Intelligence like Truth Maintenance Systems, etc.

Subsequently, we have discussed some optimizations for the original algorithm and presented the optimized IPIA. Unfortunately, the worst case complexity of the new algorithm is identical to the old one's, while its average complexity is expected to be lower. This was expected mainly because the problem of generating prime implicants itself is intractable.

In the last section we briefly explained how a *Clause Management System (CMS)* can be built by employing the incremental algorithm. This is just one of several applications that can exploit the algorithm. Other possible applications using the incremental algorithm are incremental theorem proving, generalized dignostic reasoning (or hypothesis generation) and a general system for nonmonotonic reasoning. In [Tsiknis & Kean'88] we elaborate more on the *CMS* as well as on some of its applications. Finally, we believe that for nonmonotonic reasoning system, a similar incremental method for detecting and resolving inconsistency is vitally important and we include this among the issues for future research.

Acknowledgement

We are indebted to Michael Loui for introducing Tison's Method to us and Ashok Chandra for pointing out the complexity on the number of prime implicants. We are also very grateful to Alan Mackworth, Wolfgang Bibel and Paul Gilmore for their comments and criticism. Finally, Jane Mulligan for her courageous effort in proof reading the paper.

References

[**Biswas'75**] Nripendra N. Biswas, "Introduction to Logic and Switching Theory", Gordon and Breach Science Publishers, 1975.

[**Bartee, Lebow & Reed'62**] Thomas C. Bartee, Irwin L. Lebow and Irving S. Reed, "Theory and Design of Digital Machines", McGraw-Hill Book Company, 1962.

[**Chandra & Markowsky'78**] Ashok K. Chandra and George Markowsky, "On the Number of Prime Implicants", Discrete Mathematics 24, 1978, pp 7-11.

[**Davis & Putnam'60**] Martin Davis and Hilary Putnam, "A Computing Procedure for Quantification Theory", Journal of ACM, Volume 7, 1960, pp 201-215.

[**De Kleer'86**] Johan De Kleer, "An Assumption-based TMS", Artificial Intelligence 28 (1986), pp 127-162.

[**De Kleer & Williams'87**] Johan De Kleer and B.C. Williams, "Diagnosing Multiple Faults", Artificial Intelligence 32 (1987), pp 97-130.

[**Hwa'74**] H.R. Hwa, "A Method for Generating Prime Implicants of a Boolean Expression", IEEE Transactions on Computers, June 1974, pp 637-641.

[**Hwang et all'85**] Hee-Yeung Hwang, Dong-Sub Chao and Michael E. Valdez, "A New Technique for the Minimization of Switching Functions", Conference Proceedings, IEEE Southeastcon'85, 1985, pp 299-304.

[**Kohavi'78**] Zvi Kohavi, "Switching and Finite Automata Theory", Second Edition, McGraw-Hill Book Company, 1978.

[**Loui & Bilardi'82**] Michael C. Loui and Gianfranco Bilardi, "The Correctness of Tison's Method for Generating Prime Implicants", Report R-952, UILU-ENG 82-2218, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Febuary 1982.

[**Reiter & De Kleer'87**] Raymond Reiter and Johan De Kleer, "Foundations of Assumption-Based Truth Maintenance Systems: Preliminary Report", Proceeding of AAAI-87, Seatle, Washington, 1987, pp 183-188.

[**Rhyne et all'77**] V. Thomas Rhyne, Philip S. Noe, Melvin H. McKinney and Udo W. Pooch, "A New Technique for the Fast Minimization of Switching Functions", IEEE Transactions on Computers, Vol. C-26, No. 8, August 1977, pp 757-764.

[**Robinson'65**] J.A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", JACM 12, 1965, pp 23-41.

[**Slagle, Chang & Lee'69**] J.R. Slagle, C.L. Chang and R.C.T. Lee, "Completeness Theorems for Semantics Resolution in Consequence Finding", Proceeding of IJCAI-69, Washington, D.C., 1969, pp 281-285.

[**Slagle, Chang & Lee'70**] J.R. Slagle, C.L. Chang and R.C.T. Lee, "A New Algorithm for Generating Prime Implicants", IEEE Transactions on Computers, Volume C-19, Number 4, April 1970.

[**Tison'67**] P. Tison, "Generalized Consensus Theory and Application to the Minimization of Boolean Functions", IEEE Transaction on Electronic Computers, EC-16, 4, August 1967, pp 446-456.

[**Tsiknis & Kean'88**] George Tsiknis and Alex Kean, "Clause Management Systems(CMS)", in preparation, 1988.

[**Quintus Prolog**] Quintus Prolog User's Guide Version 1.0, March 1987.

10 Appendix A

Incremental Prime Implicant/Implicate Algorithm(IPIA)

Input: A set of prime implicants Π of a formula $\mathcal{F}$ and a clause C.

Output: The set $\Sigma \cup \Pi$ is the set of prime implicants of $\Pi \cup \{C\}$.

Step 1.0 Initialize $\Sigma = \{C\}$. If C is subsumed by some clause in Π delete C and STOP.

Step 2.0 For each biform variable x occurring in C do

Step 2.1 For each $S \in \Sigma$ and $P \in \Pi$ such that S, P have consensus on x do

Step 2.1.1 $T = CS(S, P, x)$

Step 2.1.2 $\Sigma = \Sigma \cup T$.

Step 2.2 Delete any $D \in \Sigma \cup \Pi$ such that there is a $D' \in \Sigma \cup \Pi$ that covers D.

Example 4.1: Let $C = a\bar{c}t$ and let $\Pi = \{ \bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}\}$ and the output of the algorithm is

$$\Sigma \cup \Pi = \{ \begin{array}{l} a\bar{c}\bar{x}\bar{z}, a\bar{c}\bar{y}z, a\bar{c}x\bar{y}, a\bar{b}\bar{x}\bar{z}, \\ a\bar{b}\bar{y}z, a\bar{b}x\bar{y}, b\bar{c}\bar{x}\bar{z}, b\bar{c}\bar{y}z, \\ b\bar{c}x\bar{y}, a\bar{b}t, b\bar{c}t, a\bar{c}t, \\ \bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}\}. \end{array}$$

	Step	Biform	s	p	CS	Σ	Π
0	1.0	-	-	-	-	$a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
1	2.0	a	-	-	-	$a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
2	2.1	a	$a\bar{c}t$	$\bar{a}b\bar{c}$	-	$a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
3	2.1.1	a	$a\bar{c}t$	$\bar{a}b\bar{c}$	$b\bar{c}t$	$a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
4	2.1.2	a	$a\bar{c}t$	$\bar{a}b\bar{c}$	$b\bar{c}t$	$b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
5	2.2	a	$a\bar{c}t$	$\bar{a}b\bar{c}$	$b\bar{c}t$	$b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
6	2.0	$\bar{c}$	-	-	-	$b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
7	2.1	$\bar{c}$	$a\bar{c}t$	$a\bar{b}c$	-	$b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
8	2.1.1	$\bar{c}$	$a\bar{c}t$	$a\bar{b}c$	$a\bar{b}t$	$b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
9	2.1.2	$\bar{c}$	$a\bar{c}t$	$a\bar{b}c$	$a\bar{b}t$	$a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
10	2.2	$\bar{c}$	$a\bar{c}t$	$a\bar{b}c$	$a\bar{b}t$	$a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}\bar{y}z, \bar{t}\bar{x}\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
11	2.0	t	-	-	-	$ab\bar{t}, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
12	2.1	t	$b\bar{c}t$	$\bar{t}x\bar{y}$	-	$ab\bar{t}, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
13	2.1.1	t	$b\bar{c}t$	$\bar{t}x\bar{y}$	$b\bar{c}x\bar{y}$	$ab\bar{t}, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
14	2.1.2	t	$b\bar{c}t$	$\bar{t}x\bar{y}$	$b\bar{c}x\bar{y}$	$b\bar{c}x\bar{y}, ab\bar{t}, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
15	2.2	t	$b\bar{c}t$	$\bar{t}x\bar{y}$	$b\bar{c}x\bar{y}$	$b\bar{c}x\bar{y}, ab\bar{t}, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
16	2.1	t	$b\bar{c}t$	$\bar{t}y\bar{z}$	-	$b\bar{c}x\bar{y}, ab\bar{t}, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
17	2.1.1	t	$b\bar{c}t$	$\bar{t}y\bar{z}$	$b\bar{c}y\bar{z}$	$b\bar{c}x\bar{y}, ab\bar{t}, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
18	2.1.2	t	$b\bar{c}t$	$\bar{t}y\bar{z}$	$b\bar{c}y\bar{z}$	$b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, ab\bar{t}, b\bar{c}t$ $a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
19	2.2	t	$b\bar{c}t$	$\bar{t}y\bar{z}$	$b\bar{c}y\bar{z}$	$b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, ab\bar{t}, b\bar{c}t$ $a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
20	2.1	t	$b\bar{c}t$	$\bar{t}x\bar{z}$	-	$b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, ab\bar{t}, b\bar{c}t$ $a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
21	2.1.1	t	$b\bar{c}t$	$\bar{t}x\bar{z}$	$b\bar{c}x\bar{z}$	$b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, ab\bar{t}, b\bar{c}t$ $a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
22	2.1.2	t	$b\bar{c}t$	$\bar{t}x\bar{z}$	$b\bar{c}x\bar{z}$	$b\bar{c}x\bar{z}, b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, ab\bar{t}$ $b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$
23	2.2	t	$b\bar{c}t$	$\bar{t}x\bar{z}$	$b\bar{c}x\bar{z}$	$b\bar{c}x\bar{z}, b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, ab\bar{t}$ $b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, ab\bar{c}, \bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
24	2.1	t	$a\bar{b}t$	$\bar{t}x\bar{y}$	-	$b\bar{c}x\bar{z}, b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, a\bar{b}t$ $b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
25	2.1.1	t	$a\bar{b}t$	$\bar{t}x\bar{y}$	$a\bar{b}x\bar{y}$	$b\bar{c}x\bar{z}, b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, a\bar{b}t$ $b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
26	2.1.2	t	$a\bar{b}t$	$\bar{t}x\bar{y}$	$a\bar{b}x\bar{y}$	$a\bar{b}x\bar{y}, b\bar{c}x\bar{z}, b\bar{c}y\bar{z}, b\bar{c}x\bar{y}$ $a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
27	2.2	t	$a\bar{b}t$	$\bar{t}x\bar{y}$	$a\bar{b}x\bar{y}$	$a\bar{b}x\bar{y}, b\bar{c}x\bar{z}, b\bar{c}y\bar{z}, b\bar{c}x\bar{y}$ $a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
28	2.1	t	$a\bar{b}t$	$\bar{t}y\bar{z}$	-	$a\bar{b}x\bar{y}, b\bar{c}x\bar{z}, b\bar{c}y\bar{z}, b\bar{c}x\bar{y}$ $a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
29	2.1.1	t	$a\bar{b}t$	$\bar{t}y\bar{z}$	$a\bar{b}y\bar{z}$	$a\bar{b}x\bar{y}, b\bar{c}x\bar{z}, b\bar{c}y\bar{z}, b\bar{c}x\bar{y}$ $a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
30	2.1.2	t	$a\bar{b}t$	$\bar{t}y\bar{z}$	$a\bar{b}y\bar{z}$	$a\bar{b}y\bar{z}, a\bar{b}x\bar{y}, b\bar{c}x\bar{z}, b\bar{c}y\bar{z}$ $b\bar{c}x\bar{y}, a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
31	2.2	t	$a\bar{b}t$	$\bar{t}y\bar{z}$	$a\bar{b}y\bar{z}$	$a\bar{b}y\bar{z}, a\bar{b}x\bar{y}, b\bar{c}x\bar{z}, b\bar{c}y\bar{z}$ $b\bar{c}x\bar{y}, a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
32	2.1	t	$a\bar{b}t$	$\bar{t}x\bar{z}$	-	$a\bar{b}y\bar{z}, a\bar{b}x\bar{y}, b\bar{c}x\bar{z}, b\bar{c}y\bar{z}$ $b\bar{c}x\bar{y}, a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
33	2.1.1	t	$a\bar{b}t$	$\bar{t}x\bar{z}$	$a\bar{b}x\bar{z}$	$a\bar{b}y\bar{z}, a\bar{b}x\bar{y}, b\bar{c}x\bar{z}, b\bar{c}y\bar{z}$ $b\bar{c}x\bar{y}, a\bar{b}t, b\bar{c}t, a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
34	2.1.2	t	$a\bar{b}t$	$\bar{t}x\bar{z}$	$a\bar{b}x\bar{z}$	$a\bar{b}x\bar{z}, a\bar{b}y\bar{z}, a\bar{b}x\bar{y}, b\bar{c}x\bar{z}$ $b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, a\bar{b}t, b\bar{c}t$ $a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$
35	2.2	t	$a\bar{b}t$	$\bar{t}x\bar{z}$	$a\bar{b}x\bar{z}$	$a\bar{b}x\bar{z}, a\bar{b}y\bar{z}, a\bar{b}x\bar{y}, b\bar{c}x\bar{z}$ $b\bar{c}y\bar{z}, b\bar{c}x\bar{y}, a\bar{b}t, b\bar{c}t$ $a\bar{c}t$	$\bar{t}x\bar{y}, \bar{t}y\bar{z}, \bar{t}x\bar{z}, a\bar{b}c, \bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
36	2.1	t	$a\bar{c}t$	$\bar{t}x\bar{y}$	-	$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$,$ab\bar{x}\bar{y}$,$b\bar{c}\bar{x}\bar{z}$ $b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$,$a\bar{b}t$,$b\bar{c}t$ $a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
37	2.1.1	t	$a\bar{c}t$	$\bar{t}x\bar{y}$	$a\bar{c}x\bar{y}$	$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$,$ab\bar{x}\bar{y}$,$b\bar{c}\bar{x}\bar{z}$ $b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$,$a\bar{b}t$,$b\bar{c}t$ $a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
38	2.1.2	t	$a\bar{c}t$	$\bar{t}x\bar{y}$	$a\bar{c}x\bar{y}$	$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$,$ab\bar{x}\bar{y}$ $b\bar{c}\bar{x}\bar{z}$,$b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$,$a\bar{b}t$ $b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
39	2.2	t	$a\bar{c}t$	$\bar{t}x\bar{y}$	$a\bar{c}x\bar{y}$	$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$,$ab\bar{x}\bar{y}$ $b\bar{c}\bar{x}\bar{z}$,$b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$,$a\bar{b}t$ $b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
40	2.1	t	$a\bar{c}t$	$\bar{t}y\bar{z}$	-	$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$,$ab\bar{x}\bar{y}$ $b\bar{c}\bar{x}\bar{z}$,$b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$,$a\bar{b}t$ $b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
41	2.1.1	t	$a\bar{c}t$	$\bar{t}y\bar{z}$	$a\bar{c}y\bar{z}$	$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$,$ab\bar{x}\bar{y}$ $b\bar{c}\bar{x}\bar{z}$,$b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$,$a\bar{b}t$ $b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
42	2.1.2	t	$a\bar{c}t$	$\bar{t}y\bar{z}$	$a\bar{c}y\bar{z}$	$a\bar{c}y\bar{z}$,$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$ $a\bar{b}x\bar{y}$,$b\bar{c}\bar{x}\bar{z}$,$b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$ $a\bar{b}t$,$b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
43	2.2	t	$a\bar{c}t$	$\bar{t}y\bar{z}$	$a\bar{c}y\bar{z}$	$a\bar{c}y\bar{z}$,$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$ $a\bar{b}x\bar{y}$,$b\bar{c}\bar{x}\bar{z}$,$b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$ $a\bar{b}t$,$b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$

	Step	Biform	s	p	CS	Σ	Π
44	2.1	t	$a\bar{c}t$	$\bar{t}x\bar{z}$	-	$a\bar{c}y\bar{z}$,$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$ $a\bar{b}x\bar{y}$,$b\bar{c}\bar{x}\bar{z}$,$b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$ $a\bar{b}t$,$b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
45	2.1.1	t	$a\bar{c}t$	$\bar{t}x\bar{z}$	$a\bar{c}x\bar{z}$	$a\bar{c}y\bar{z}$,$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$,$ab\bar{y}\bar{z}$ $a\bar{b}x\bar{y}$,$b\bar{c}\bar{x}\bar{z}$,$b\bar{c}\bar{y}\bar{z}$,$b\bar{c}x\bar{y}$ $a\bar{b}t$,$b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
46	2.1.2	t	$a\bar{c}t$	$\bar{t}x\bar{z}$	$a\bar{c}x\bar{z}$	$a\bar{c}x\bar{z}$,$a\bar{c}y\bar{z}$,$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$ $a\bar{b}y\bar{z}$,$a\bar{b}x\bar{y}$,$b\bar{c}x\bar{z}$,$b\bar{c}y\bar{z}$ $b\bar{c}x\bar{y}$,$a\bar{b}t$,$b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$
47	2.2	t	$a\bar{c}t$	$\bar{t}x\bar{z}$	$a\bar{c}x\bar{z}$	$a\bar{c}x\bar{z}$,$a\bar{c}y\bar{z}$,$a\bar{c}x\bar{y}$,$ab\bar{x}\bar{z}$ $a\bar{b}y\bar{z}$,$a\bar{b}x\bar{y}$,$b\bar{c}x\bar{z}$,$b\bar{c}y\bar{z}$ $b\bar{c}x\bar{y}$,$a\bar{b}t$,$b\bar{c}t$,$a\bar{c}t$	$\bar{t}x\bar{y}$, $\bar{t}y\bar{z}$, $\bar{t}x\bar{z}$, abc, $\bar{a}b\bar{c}$

11 Appendix B

```
/*=================================================================*/
/*=                                                               =*/
/*= Program: Incremental Prime Implicant Algorithm (IPIA).        =*/
/*= Author : Alex Kean and George Tsiknis                         =*/
/*= Date   : March 1988                                           =*/
/*= Routine: ipia(Goal, PI, NewPI).                               =*/
/*=                                                               =*/
/*= System : Quintus Prolog Release 2.2 (Sun-3, Unix 3.5)         =*/
/*=          Copyright (C) 1987, Quintus Computer Systems, Inc.   =*/
/*=================================================================*/

:- compile(library(basics)).
:- compile(library(lists)).
:- compile(library(sets)).
```

11.1 Consensus

```
/*=================================================================*/
/*= CONSENSUS program:                                            =*/
/*=  css(Clause1, Clause2, Consensus, Biform).                    =*/
/*=                                                               =*/
/*=  - The consensus of "Clauses1" and                            =*/
/*=    "Clause2" with respect the the biform                      =*/
/*=    literal "Biform" is "Consensus".                           =*/
/*=================================================================*/

polar(+X,-X).  polar(-X,+X).

complement(-Literal,Clause) :- select(+Literal,Clause,_).
complement(+Literal,Clause) :- select(-Literal,Clause,_).

resolve(L1,L2,Resolvent,X) :-
    polar(X,Y),
    select(X,L1,IntL1),
    select(Y,L2,IntL2),
    union(IntL1,IntL2,Resolvent).

fundamental(P) :-
    select(C,P,IntP),
    complement(C,IntP), !, fail.
fundamental(_).

css(C1,C2,P,X) :-
    resolve(C1,C2,P,X),
    fundamental(P).
```

11.2 Subsumption

```
/*=================================================================*/
/*= SUBSUMPTION program:                                          =*/
/*=   subsumption(Set_of_Clauses, New_Set).                       =*/
/*=                                                               =*/
/*=================================================================*/

subsumption([],[]).
subsumption([X],[X]).
subsumption(List,NewList) :-
    select(C1,List,List1),
    select(C2,List1,List2),
    subsume(C1,C2,C),
    subsumption([C|List2],NewList).
subsumption(List,List).

subsume(C1,C2,C1) :- subset(C1,C2).
subsume(C1,C2,C2) :- subset(C2,C1).
```

11.3 Main Algorithm

```
/*=================================================================*/
/*= Incremental Method: ipia(Goal, PI, NewPI).                    =*/
/*=   - Goal is a fundamental clause.                             =*/
/*=   - PI is a set of prime implicants.                          =*/
/*=   - NewPI is the set of prime implicants of                   =*/
/*=             PI U {Goal}.                                      =*/
/*=================================================================*/

for_each_pi(Biform, Goal, Sigma, [Resolvent|NewRes]) :-
    select(PI, Sigma, IntSigma),
    css(Goal, PI, Resolvent, Biform),
    for_each_pi(Biform, Goal, IntSigma, NewRes).
for_each_pi(_,_,_,[]).

for_each_si(Biform, [Goal|Tail], Sigma, NewRes) :-
    for_each_pi(Biform, Goal, Sigma, Res1),
    for_each_si(Biform, Tail, Sigma, Res2),
    union(Res1, Res2, NewRes).
for_each_si(_,_,_,[]).

for_each_biform([], OldRes, _, OldRes).
for_each_biform([Biform|BTail], OldRes, Sigma, NewRes) :-
    for_each_si(Biform, OldRes, Sigma, IntRes),
    union(IntRes, OldRes, TempRes),
    subsumption(TempRes, Res),
    for_each_biform(BTail, Res, Sigma, NewRes).

ipia(Goal, Sigma, PI) :-
    for_each_biform(Goal,[Goal],Sigma,NewSi),
    union(NewSi,Sigma,IntSi),
    subsumption(IntSi,PI).
```