# A new algorithm for incremental prime implicate generation

Teow-Hin Ngair
Institute of Systems Science
National University of Singapore
Kent Ridge, Singapore 0511
Republic of Singapore

## Abstract

Traditional algorithms for prime implicate generation [Quine, 1952; McCluskey, 1956; Tison, 1967; Kean and Tsiknis, 1990; de Kleer, 1992] require the input formulas to be first transformed into a CNF formula. This process, however, requires exponential time in the worst case and can result in an exponential blow up of the input size. Such cases occur frequently when the problem domains are best characterized by some DNF formulas. In this paper, we study a new algorithm which allows a more general input: a conjunction of DNF formulas. We will present empirical results comparing the new algorithm with some existing implementations, and discuss how it can be used in a propositional abductive reasoning system.

## 1 Introduction

In the early 1950's, when researchers were studying various ways of minimizing boolean circuits, they discovered that the notion of *prime implicant* played an important role [Quine, 19521. In particular, it was shown that the sum of products minimization of boolean circuits will consist of only the prime implicants of the propositional description of the circuit. Subsequently, many algorithms were proposed for the purpose of prime implicant generation [McCluskey, 1965; Slagle *et al.,* 1970; Tison, 1967].

Recently, in an attempt to generalize the ATMS, it was realized that the dual notion, *prime implicate,* subsumes the concept of labels calculated by an ATMS. This has lead to the study of Clause Management Systems (CMS) [Reiter and de Kleer, 1987] which essentially accepts a set of propositional clauses and outputs the corresponding set of prime implicates. Similar to an ATMS, the applications for a CMS include diagnosis [de Kleer and Williams, 1987; Reiter, 1987], qualitative physics [Forbus, 1990] and nonmonotonic reasoning [McCarthy, 1980; Reiter, 1980].

In this paper, we present a new algorithm for computing the prime implicates (hence, the prime implicants for the dual problem) based on a $\Phi$ operation described in the next section. The major differences between this algorithm and the existing prime implicate generation algorithms are 1. it was originally motivated entirely from the order-theoretic point of view; and 2. it allows input in a more general form: a conjunction of DNF formulas.

The latter is especially important for problem solving because the existing algorithms can only accept as input a single CNF formula (or a single DNF formula as in [Slagle *et al,* 1970]) which is a special case of conjunction of DNF formulas. These algorithms are needlessly expensive for problems that are naturally encoded by a conjunction of some complex DNF formulas, because these DNF formulas need to be transformed into their CNF equivalence first. This additional step is very expensive computationally and can lead to an exponential increase in the input size. On the other hand, our new algorithm does not suffer from this shortcoming and we will present empirical results to demonstrate its superior performance. Furthermore, we will describe an application in abductive reasoning to show how its generality can be useful to problem solving.

We will adopt the following notation in the later discussion: for any DNF formula $\theta$, $\mathcal{E}(\theta)$ is the set of conjuncts in $\theta$; and for any disjunctive clause $\vartheta$, *i.e.* a disjunction of non-repetitive literals, $\overline{\vartheta}$ is the conjunction of the negation of literals in $\vartheta$. Furthermore, we interpret a disjunctive clause as a special case of a DNF formula, where each conjunct happens to contain a single element.

**Definition:** Given a set of propositional formulas $\Theta$, a disjunctive clause $\vartheta$ is called an *implicate* of $\Theta$ if $\Theta \models \vartheta$. Furthermore, it is called a *prime implicate*[1] if there is no other disjunctive clause $\vartheta'$ such that $\mathcal{E}(\vartheta') \subset \mathcal{E}(\vartheta)$ and $\Theta \models \vartheta'$. A conjunctive clause $\overline{\vartheta}$ is called an *implicant* of $\Theta$ if $\vartheta$ is an implicate of $\neg\Theta$. It is called a *prime implicant* if $\vartheta$ is a prime implicate of $\neg\Theta$. □

By applying de Morgan's law twice, a direct consequence of the dual nature of implicate and implicant is that any algorithm which generates the prime implicates given a conjunction of DNF formulas can automatically be used to generate the prime implicants when given a disjunction of CNF formulas.

## 2 $\Phi$ Operation

Our new algorithm for prime implicate generation is based on order theory. In this section, we introduce the basic definitions

[1] Our definition of prime implicate/implicant is slightly more general than the definition used by some other authors. In particular, we allow clauses that contain complementary literals to be prime. However, one can easily derive one set from the other.

that are relevant to this paper. First, the notion of a closure operation:

**Definition:** Given a lattice $(L, \preceq)$ and a function $f : L \longrightarrow L$, $f$ is called a *closure operation* if it satisfies the following properties:

**C1.** $x \preceq f(x)$, for any $x$ in $L$;
**C2.** $f(f(x)) = f(x)$; and
**C3.** $f(x) \preceq f(y)$, if $x \preceq y$. □

The significance of a closure operation lies in its images which are called *fixed points*. For the purpose of this paper, we need only to observe that for any $x \in L$, $f(x) \succeq x$ is the smallest element in $L$ that remains invariant under $f$. Hence, $f(x)$ is said to be the *least* fixed point above $x$. The operation which leads to the new algorithm is defined below:

**Definition:** Given any poset $(P, \preceq)$, a subset $C \subseteq P$ is said to be *downward closed* if $p \in C$ and $p' \preceq p$ implies $p' \in C$. Furthermore, given any subset $S \subseteq P$, the *downward closure* of $S$ is the set $\downarrow S = \{p \in P \mid \exists p' \in S, p \preceq p'\}$. □

**Definition:** Let $(P, \preceq)$ be any lattice. Given any subset $T \in \mathcal{P}(P)$, the operation $\Phi_T$ is defined on downward closed sets $C \in \mathcal{P}(P)$ by:

$$\Phi_T(C) = \{p \in P \mid \forall t \in T, p \wedge t \in C\},$$

where $\wedge$ is the lattice meet operation induced by $\preceq$. □

An important property of $\Phi$ [Ngair, 1992] is:

**Theorem 1** *Given any $\Phi_{T_1}, \ldots, \Phi_{T_n}$ and any downward closed $C$, $F = \Phi_{T_1} \circ \cdots \circ \Phi_{T_n}(C)$ is the least* common *fixed point of every $\Phi_{T_i}$, $1 \leq i \leq n$, above $C$.*

## 3 Algorithm

Given a propositional system $S$ with a finite number of propositional symbols $x_1, x_2, \ldots, x_n$, we consider the lattice $P$ which is the power set of $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. Let $C_P$ be the downward closure defined by the set $\{\{x_i, \neg x_i\} \mid 1 \leq i \leq n\}$. Note that the elements of $P$ are ordered by the reverse of set inclusion, *i.e.* $\preceq \equiv \supseteq$.

Using the same terminology as in [de Kleer, 1986], we refer to elements of $P$ as *environments* and $P$ as the *environment lattice* of $S$. We interpret an environment $\{x_{i_1}, \ldots, x_{i_k}\}$ as the conjunctive clause $x_{i_1} \wedge \cdots \wedge x_{i_k}$. Therefore, $C_P$ represents the set of all inconsistent conjunctive clauses in $P$. Observe that the ordering on $P$ is defined for the lattice theoretic *meet* operation to coincide with the logical *and* operation. However, bear in mind that set theoretically $\wedge \equiv \cup$ and $\vee \equiv \cap$ in $P$.

Given an environment $p = \{x_{i_1}, \ldots, x_{i_k}\}$, we denote its negation by $\overline{p}$, *i.e.* $\overline{p} = \neg x_{i_1} \vee \cdots \vee \neg x_{i_k}$. Hence, if $p$ is inconsistent, we have $x_{i_1} \wedge \cdots \wedge x_{i_k} \models \perp$ which implies that $\models \neg x_{i_1} \vee \cdots \neg x_{i_k}$, *i.e.* $\overline{p}$ is a tautology.

The above observation can be extended to the case when a given set of formulas $\Theta$ is assumed to be true. In particular, given any environment $p$, we say that $p$ is *inconsistent with* $\Theta$ if $\Theta \wedge p \models \perp$. It is not difficult to see that:

**Lemma 2** *An environment $p$ is inconsistent with $\Theta$ if, and only if, $\overline{p}$ is an implicate of $\Theta$. Furthermore, $p$ is maximal if, and only if, $\overline{p}$ is a prime implicate.*

Hence, to find the prime implicates of $\Theta$, it suffices to find the maximal environments (minimal subsets) inconsistent with $\Theta$. In view of the lemma, we shall also refer to such maximal environments as the *negated* prime implicates.

In the following, we will first discuss an operation which computes the set of environments inconsistent with $\Theta$ and describe later an equivalent algorithm which only generates the maximal environments. From now on, we will always assume that $\Theta$ is a conjunction of DNF formulas.

**Theorem 3** *Given a conjunction of DNF formulas $\Theta = \theta_1 \wedge \cdots \wedge \theta_k$, $\vartheta$ is an implicate of $\Theta$ if, and only if, $\overline{\vartheta}$ is an element of $C'_P = \Phi_{T_k} \circ \cdots \circ \Phi_{T_1}(C_P)$, where $T_i = \mathcal{E}(\theta_i)$ for $1 \leq i \leq k$.*

What remains to be done is to find an algorithm to compute the $\Phi$ operation. The following result tells us how:

**Theorem 4** *If $P$ is an environment lattice, given any downward closed $C$ with maximal elements $\{s_1, \ldots, s_m\}$ and any $T = \{t_1, \ldots, t_n\}$, we have:*

$$\Phi_T(C) = \bigcap_{1 \leq j \leq n} \bigcup_{1 \leq i \leq m} E^i_j \qquad (1)$$

*where $E^i_j = \{p \in P \mid p \wedge t_j \preceq s_i\}$. Furthermore, $E^i_j$ is a downward closed set with an unique least upper bound $e^i_j = s_i - t_j$.*

For efficiency reason, we don't want to compute every element in $\Phi_T(C)$ if we don't have to. In particular, during the computation of $\Phi_T(C)$, we can safely ignore those elements that are already in $C$, because we can simply add $C$ to the resulting set to obtain the desirable results. Therefore, a straight forward optimization is to ignore $E^i_j$ when $s_i - t_j = s_i$, because the $E^i_j$ in this instance is contained entirely in $C$ and would not have contributed any new element in the computation of Eq. (1).

Furthermore, in an actual implementation of the $\Phi$ operation, it is expensive to consider the entire downward closed set. Since we are only interested in the maximal elements of $\Phi_T(C)$, it would be desirable if we could represent every downward closed set by their maximal elements and find equivalent operations for $\cap$ and $\cup$ which operate on this representation. The following result says that this can be achieved.

**Theorem 5** *Given two downward closed sets $A$ and $B$, let $S_A$ and $S_B$ be the maximal-element sets of $A$ and $B$ respectively, then*

$$\begin{aligned} \mathcal{MAX}(A \cap B) &= \mathcal{MAX}(\{a \cup b \mid a \in S_A, b \in S_B\}), \\ \mathcal{MAX}(A \cup B) &= \mathcal{MAX}(S_A \cup S_B), \end{aligned}$$

*where $\mathcal{MAX}$ is the subsumption check operation which returns the set of maximal elements of its input set.*

So if we are given the maximal elements of two downward closed sets $A$ and $B$, the above formulas allows us to compute the maximal elements of $A \cap B$ and $A \cup B$ directly. For the convenience in later discussion, we will denote the above operations as $S_A \sqcap S_B$ and $S_A \sqcup S_B$ respectively.

For the rest of this paper, we will use the maximal elements to represent every downward closed set. In particular, the $\Phi_T$ operation will operate in this representation, *i.e.* taking in a set of maximal elements and using the formulas in Theorem 4 and 5 for computing the corresponding output as a set of maximal elements.

From the above discussion, we obtain the following algorithm for incremental calculation of the prime implicates, where $S$ is the existing set of prime implicates:

**procedure** GEN-PI($S, T$)
for each propositional symbol $x$ in $T$
  if $x$ is not encountered earlier
  then add $\{x, \overline{x}\}$ to $S$;
return $\Phi_T(S)$;
**end-procedure** GEN-PI

Note that most of the tautological prime implicates are represented implicitly in the algorithm and are introduced only when required. This improves the efficiency of the algorithm as well as allows for potentially unbounded number of propositional symbols.

From our discussion that follows Theorem 4, we know that we can safely ignore $e_j^i$ in the computation of $\Phi_T(S)$ when $s_i$ and $t_j$ are mutually exclusive sets. We just need to add $S$ to the resulting set and perform subsumption check to derive the desirable results.

**Example** Consider the following set of clauses:

$$A_3 \Rightarrow A_1; A_4 \Rightarrow A_2; (A_1 \wedge A_2) \vee (A_3 \wedge A_4).$$

After processing the first two clauses, we obtain the following set of negated prime implicates:

$$s_1 = \{A_1, \overline{A}_3\}; \quad s_2 = \{A_2, \overline{A}_4\}; \quad s_3 = \{A_1, \overline{A}_1\};$$
$$s_4 = \{A_2, \overline{A}_2\}; \quad s_5 = \{A_3, \overline{A}_3\}; \quad s_6 = \{A_4, \overline{A}_4\}.$$

By letting $t_1 = \{A_1, A_2\}$ and $t_2 = \{A_3, A_4\}$, the processing of the final clause using the $\Phi$ operation is summarized in the Table 1: first, the set differences $e_j^i = s_i - t_j$ are calculated, *e.g.* $e_1^1 = s_1 - t_1 = \{A_1, \overline{A}_3\} - \{A_1, A_2\} = \{\overline{A}_3\}$, and $e_2^1$ is ignored because $s_1$ and $t_2$ do not share any common element; then, we apply $\sqcup$ to each row in the matrix to obtain $\mathcal{MAX}(\bigcup_{1 \le i \le 6} E_j^i)$ for $j = 1, 2$; finally, we apply $\sqcap$ on these results to get our new set of negated prime implicates $\{\{\overline{A}_3\}, \{\overline{A}_4\}\}$. By combining the new and the old prime implicates and performing subsumption check, the resulting list of prime implicates will be $\{A_1 \vee \overline{A}_1, A_2 \vee \overline{A}_2, A_3, A_4\}$.

To help understand the $\Phi$ operation from a pure logical point of view, we map the above algorithm to the following "resolution" rule which infers new implicates from the existing set of prime implicates and the input DNF formula:

1. Input DNF formula:
   $(x_1^1 \wedge \cdots \wedge x_{m_1}^1) \vee \cdots \vee (x_1^k \wedge \cdots \wedge x_{m_k}^k)$.
2. Any k negated PI's:
   $(y_1^1 \wedge \cdots \wedge y_{n_1}^1), \ldots, (y_1^k \wedge \cdots \wedge y_{n_k}^k)$,
3. Let: $e_j = \{y_1^j, \ldots, y_{n_j}^j\} - \{x_1^j, \ldots, x_{m_j}^j\}$,
   $E = \bigcup_{1 \le j \le k} e_j$;
4. Resolvent (a new implicate): $\bigvee_{l \in E} \neg l$.

From Theorem 3, 4 and 5, we can conclude that the inference is sound and complete in terms of prime implicate generation. In particular, to obtain the new set of prime implicates, we simply filter out from the set of new implicates those that are subsumed, *i.e.* remove an implicate if there exists another implicate that is a sub-clause of it.

The above resolution rule, though it appears simpler, lacks the matrix-like computational structure that is inherent in the order-theoretic approach. This is significant in terms of deriving an efficient algorithm for generating prime implicates.

## 4 Complexity and Optimization

It can be shown ([Ngair, 1992]) that starting with $n$ prime implicates and an input disjunction of $k$ conjuncts, the worst case complexity of computing GEN-PI is $O(c * (n/k)^{2k})$ or $O(c * n^{2k})$ respectively, depending on whether the input is a disjunctive clause or a more general DNF formula. Note that the complexity results for disjunctive clause coincide with those reported in [Kean and Tsiknis, 1990].

The GEN-PI algorithm as presented is not very efficient and can not be used to solve many interesting problems. Various optimization techniques have been proposed for traditional prime implicate generation algorithms [Kean and Tsiknis, 1990]. Many of these techniques are equally applicable to our new algorithm. We have also discovered additional optimization techniques for improving the efficiency of the GEN-PI algorithm and some of them are described below.

Consider the intersection of two maximal-element sets $S_1 = \{p_1, \ldots, p_m\}$ and $S_2 = \{q_1, \ldots, q_n\}$. A simple minded algorithm will compute the set $\{p \cup q \mid p \in S_1, q \in S_2\}$ follow by deleting elements that are not maximal. This is an expensive $O(c*(m*n)^2)$ operation. Therefore, it is desirable to minimize this cost.

**Lemma 6** *If there exists $p \in S_1$ and $q \in S_2$ such that $p \preceq q$, then $S_1 \sqcap S_2 = \{q\} \cup (S_1 \sqcap (S_2 - \{q\}))$.*

The above result says that we can remove every element from $S_1$ and $S_2$ that is subsumed by an element in the other list. The intersection can then be calculated by applying intersection to the two smaller lists followed by an union with the removed elements. This is beneficial since it might reduce a large number of elements that would otherwise be generated by the intersection operation, only to be discarded later.

**Lemma 7** *Given $p \in S_1$ and $q \in S_2$, one needs only to subsumption check $p \sqcap q$ with respect to the set $N = \{p \sqcap q' \mid q' \in S_2, q \neq q'\} \cup \{p' \sqcap q \mid p' \in S_1, p \neq p'\}$.*

By viewing an intersection operation as first forming a cross product matrix between $S_1$ and $S_2$, the above result says that to check the maximality of each element in the matrix, it is sufficient to only compare it with elements from the same row or column. Therefore, one reduces the worst case cost of computing $\mathcal{MAX}$ in the intersection operation from $O((|S_1| * |S_2|)^2)$ to $O((|S_1| + |S_2|) * |S_1| * |S_2|)$. Note that the value $(|S_1| + |S_2|) * |S_1| * |S_2|$ is minimum when $|S_1| = |S_2|$, *i.e.* we have $2 * |S_1|^3$.

It is generally beneficial to compare the output of an intersection operation with the set $S$ in the $\Phi$ procedure, so that no redundant element needs to be involved in further intersection operations which might otherwise generate an even larger redundant set. In the following, we describe a method which helps to identify many such redundant elements.

**Definition:** In the $\Phi_T$ operation, one computes elements of the form $p = e_{j_1}^{i_1} \cup \cdots \cup e_{j_p}^{i_p}$ for some $1 \le p \le k$, such $p$'s are called PPI (*Partial Prime Implicate*). The set $T - \{t_{j_1}, \ldots, t_{j_p}\}$ is called the *complement set* of $p$. A PPI with null complement set is also known as a PI. A PPI $p'$ is said to

| $e_j^i$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $\sqcup$ |
|---|---|---|---|---|---|---|---|
| $t_1$ | $\{\overline{A}_3\}$ | $\{\overline{A}_4\}$ | $\{\overline{A}_1\}$ | $\{\overline{A}_2\}$ | × | × | $\{\{\overline{A}_1\},\{\overline{A}_2\},\{\overline{A}_3\},\{\overline{A}_4\}\}$ |
| $t_2$ | × | × | × | × | $\{\overline{A}_3\}$ | $\{\overline{A}_4\}$ | $\{\{\overline{A}_3\},\{\overline{A}_4\}\}$ |
| New negated prime implicates ($\sqcap$) | | | | | | | $\{\{\overline{A}_3\},\{\overline{A}_4\}\}$ |

Table 1: An example of executing the $\Phi$ operation

be *generated* from a PPI $p$ if $p' = p \cup e_{j_1}^{i_1} \cdots \cup e_{j_q}^{i_q}$ for some $0 \leq q \leq k - p$. □

**Lemma 8** *Given a disjunctive clause represented by $T$ and a PPI $p$ with complement set $X$, any PI $q$ generated by $p$ will be subsumed by $S$ if $p \cup \overline{X}$ is subsumed by $S$.*

Therefore, any PPI which satisfies the condition in the above lemma can be safely deleted without affecting the prime implicate generation process. The saving is significant because such PPI may generate many more new PPI's in the $\Phi$ operation, if it is not deleted. From empirical experience, we may eliminate as much as eighty percent of PPI's by the above optimization.

## 5 Comparison with existing algorithms

In this section, we compare the GEN-PI algorithm with some existing prime implicate generation algorithms, in particular, the IPIA algorithm described in [Kean and Tsiknis, 1990] and the CLTMS algorithm described in [de Kleer, 1992].[2] We will compare the algorithms based on two examples[3] which represent two types of problem, one is an artificial set of clauses which is designed to produce an exponential number of prime implicates while the other is derived from more practical problems. The first example is usually referred to as the "m(x)k(y)" problem. It has two integer parameters, $x, y$ with $x * y + 1$ input clauses and will produce $(x+1)^y + x * y$ (non-tautological) prime implicates:

$$s_1^1 \Rightarrow a_1; \quad \ldots; \quad s_x^1 \Rightarrow a_1;$$
$$s_1^2 \Rightarrow a_2; \quad \ldots; \quad s_x^2 \Rightarrow a_2;$$
$$\cdots$$
$$s_1^y \Rightarrow a_y; \quad \ldots; \quad s_x^y \Rightarrow a_y;$$
$$\neg a_1 \vee \cdots \vee \neg a_n.$$

The second example is Kean and Tsiknis's propositional encoding of a familiar diagnosis problem; the 5-gate adder circuit (see [Kean and Tsiknis, 1992]).

The GEN-PI and the CLTMS algorithms are both implemented in Common Lisp and run on a Sun 4/490 machine. On the other hand, the IPIA algorithm is implemented in Quintus Prolog running on a Sun Sparc-1 machine.[4] The results of the comparison are listed in Table 2 where all liming information are in seconds and PI# is the number of non-tautological prime implicates.

[2]The latter algorithm is also called IPIA in the actual paper. We use CLTMS here to avoid confusion.

[3]Both examples were originally proposed by Alex Kean in a series of e-mail correspondences between Alex Kean, Johan de Kleer and the author.

[4]The timing information of the IPIA program was provided by Alex Kean.

| Examples | PI# | CLTMS | GEN-PI | IPIA |
|---|---|---|---|---|
| m3k6 | 4114 | 7-15 | 9 | N.A. |
| m3k7 | 16405 | 30-120 | 45 | 40320 |
| circuit | 9700 | 1138 | 300 | > 1 day |

Table 2: Comparison of three PI generation algorithms

Note that the CLTMS algorithm is sensitive to the input order of the clauses. This is reflected by the two-value entries for the "m3k6" and "m3k7" examples. In both the examples, input of the $x * y$ Horn clauses are given in lexicographical order based on the two indices of $s$. The two values represent the times needed depending on how we prioritize the two indices.

From the results of the comparison, we can conclude that if the input is already in a CNF, the GEN-PI algorithm will perform as efficiently as, or better than, some of the existing prime implicate generation systems. In the following, we show that the GEN-PI algorithm will greatly outperform the existing algorithms if the input is in a more general form.

## 6 The generality of the GEN-PI algorithm

As already emphasized earlier in this paper, the GEN-PI algorithm naturally allows input formula to be a conjunction of DNF formulas. Therefore, an obvious advantage to this new algorithm is its flexibility in terms of what it can take as input, *i.e.* a set of DNF formulas instead of just a set of disjunctive clauses. Although a set of DNF formulas can be converted to a single CNF formula by converting each DNF formula to an equivalent CNF formula. We will show in the following, however, that such a conversion is computationally very expensive.

Since the conversion of a DNF formula to a CNF formula is computationally equivalent to the process of covcrting a CNF formula to a DNF formula (modulo a linear term), it suffices to demonstrate a particular CNF formula of length *O(n)* such that any of its DNF equivalence is necessary of length exponential in n. Let us consider the DNF formula: $f = (x_1 \vee x_2) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$, where each $x_i$, $1 \leq i \leq 2n$, is a distinct propositional symbol. It can be shown that:

Theorem 9 *Any DNF formula equivalent to the propositional formula f has at least $2^n$ conjuncts.*

Hence, the problem of converting a CNF formula to an equivalent DNF formula (or vice versa) is a provably intractable problem.

Examples: The following shows some instances where DNF

formulas arise naturally in problem encoding.

1. An exclusive-or constraint, *e.g.* $\mathbf{xor}(x_1, \ldots, x_n)$, commonly encounter in a constraint satisfaction problem, can be intuitively encoded as a single DNF formula $(x_1 \wedge \neg x_2 \wedge \cdots \wedge \neg x_n) \vee \cdots \vee (\neg x_1 \wedge \cdots \wedge \neg x_{n-1} \wedge x_n)$ but needs at least $O(n^2)$ disjunctive clauses in CNF encoding. In general, it is much simpler to use a DNF encoding for a constraint describing a complete set of possible outcomes.

2. An OR-gate in a diagnostic problem, *e.g.* $(x_1 \vee \cdots \vee x_n) \wedge \neg \mathbf{AB}(G) \Rightarrow x$, can be naturally represented as a single DNF formula but needs $n$ disjunctive clauses in CNF encoding. In the worst case, consider the "black-box" circuit shown in Figure 1. It can be encoded as a single DNF formula $(\neg x_1 \wedge \neg x_2) \vee \cdots \vee (\neg x_{2n-1} \wedge \neg x_{2n}) \vee \mathbf{AB}(G) \vee x$ but needs at least $2^n$ disjunctive clauses in CNF encoding (Theorem 9).

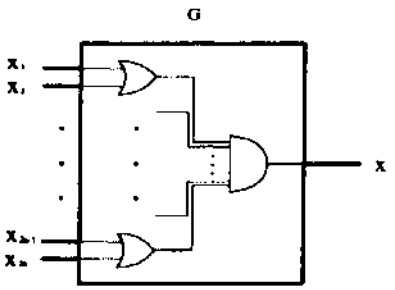


Figure 1: A "black-box" circuit

The above results and examples tell us that the existing prime implicate generation algorithms are very inefficient because they need to perform the additional expensive operation of converting every DNF formula to a CNF formula if the input contains some DNF formulas. Such conversion also entails the possibility of enormous growth in the input size to the algorithms. Note that such additional processing cost is not hidden in the GEN-PI algorithm. In particular, the algorithm treats each DNF formula as a single input with computational complexity determines roughly by the size of output. For instance, consider the following set of clauses (generalization of the example in Section 3) which produces only 2 non-tautological prime implicates, $A_1$ and $A_2$:

$$A_3 \Rightarrow A_1; \qquad A_4 \Rightarrow A_2;$$
$$\cdots$$
$$A_{2n-1} \Rightarrow A_1; \qquad A_{2n} \Rightarrow A_2;$$
$$(A_1 \wedge A_2) \vee \cdots \vee (A_{2n-1} \wedge A_{2n}).$$

Feeding it to the prime implicate generation algorithms, we obtain the run times listed in Table 3. It is clear that GEN-PI is the only reasonable algorithm for computing the prime implicates with such input even with reasonably small size. The poor performance of the other algorithms is due to their additional costs in converting the final input clause into $2^n$ disjunctive clauses and processing the exponentially larger input.

A point which we have not addressed so far is the seemingly workable solution of encoding a DNF formula as a linear size CNF formula by introducing new literals. For instance, any

| n | PI# | GEN-PI | CLTMS | IPIA |
|---|---|---|---|---|
| 5 | 2 | 0.05 | 0.1 | 0.23 |
| 10 | 2 | 0.13 | 20 | 111 |
| 13 | 2 | 0.20 | 1105 | N.A. |
| 15 | 2 | 0.24 | > 1 hour | N.A. |
| 100 | 2 | 10.25 | N.A. | N.A. |

Table 3: Comparison of algorithms with DNF input

DNF formula

$$(X_1^1 \wedge \cdots \wedge X_{n_1}^1) \vee \cdots \vee (X_1^k \wedge \cdots \wedge X_{n_k}^k)$$

can be encoded as the following set of disjunctive clauses:

$$C_1 \Rightarrow X_1^1; \quad \cdots; \quad C_1 \Rightarrow X_{n_1}^1;$$
$$C_2 \Rightarrow X_1^2; \quad \cdots; \quad C_2 \Rightarrow X_{n_2}^2;$$
$$\cdots$$
$$C_k \Rightarrow X_1^k; \quad \cdots; \quad C_k \Rightarrow X_{n_k}^k;$$
$$C_1 \vee \cdots \vee C_k$$

where $C_1, \ldots, C_k$ are new propositional symbols. This method does not suffer from exponential conversion cost and preserves all the prime implicates. In particular, it is not difficult to show that by filtering out the prime implicates containing the new propositional symbols from the prime implicates of the encoded problem, we obtain the prime implicates of the original problem. However, the cost of generating additional new prime implicates usually outweighs the benefit of the compact encoding. Consider the previous example that has only two non-tautological prime implicates. Using the encoding method, we ended up with $3^{n-1} + 2 * n$ non-tautological prime implicates. Computationally, this is much more expensive than just converting the DNF formula to $2^n$ disjunctive clauses and processing them as a CNF formula. Hence, the restriction in the traditional algorithms of accepting only a CNF formula does in fact lead to a significant computational overhead in problem solving.

## 7 Abductive Reasoning

Abduction is a form of reasoning where given an inference system, some facts $\Sigma$ and a goal formula $g$, the reasoner finds an explanation $e$ for $g$, *i.e.* $\Sigma \wedge e \models g$. Usually, we also require *consistency* in the explanation that the reasoner finds, *i.e.* we want $\Sigma \wedge e$ to be consistent. In this section, we study a limited form of abductive reasoning where the inference system is restricted to the propositional logic and the explanation is restricted to conjunctive clause. To this end, we consider the notion of *minimal support* originally proposed in [Reiter and de Kleer, 1987] to solve this problem. However, the original proposal restricts the goal formula to a disjunctive clause. We will show that such a restriction is unnecessary and a much more complex class of goal formulas can be specified *directly* using the GEN-PI algorithm.

**Definition:** Given a set of propositional formulas $\mathcal{F}$ and any formula $Z$, a disjunctive clause $Y$ is said to be a support for $Z$ with respect to $\mathcal{F}$ iff $\mathcal{F} \not\models Y$ and $\mathcal{F} \models Y \vee Z$. $Y$ is said to be a *minimal support* for $Z$ with respect to $\mathcal{F}$ iff no sub-clause $Y' \subset Y$ is a support for $Z$ with respect to $\mathcal{F}$.

It is immediate that a conjunctive clause $\neg Y$ is a consistent explanation for $Z$ in the system $\mathcal{F}$ if, and only if, $Y$ is a support for $Z$ with respect to $\mathcal{F}$, or equivalently, there exists minimal support $Y'$ for $Z$ with respect to $\mathcal{F}$ such that $Y' \subseteq Y$. In the following, we describe how the $\Phi$ operation can also be used for generating minimal supports. By observing that $\mathcal{F} \models Y \vee Z$ if, and only if, $\mathcal{F} \wedge \neg Z \models Y$, we have:

**Lemma 10** *A disjunctive clause $Y$ is a support for $Z$ with respect to $\mathcal{F}$ iff $Y$ is not an implicate of $\mathcal{F}$ but is an implicate of $\mathcal{F} \wedge \neg Z$. $Y$ is a minimal support for $Z$ with respect to $\mathcal{F}$ iff $Y$ is not an implicate of $\mathcal{F}$ but is a prime implicate of $\mathcal{F} \wedge \neg Z$.*

In this paper, we assume that $Z$ is a disjunction of CNF formulas, *i.e.* $\neg Z$ is a conjunction of DNF formulas represented by say, $T_1, \ldots, T_n$. To calculate the set of minimal supports for $Z$ with respect to $\mathcal{F}$, we can apply $\Phi_{T_1}, \ldots, \Phi_{T_n}$ sequentially to the set $S$ of prime implicates of $\mathcal{F}$. Subsequently, by removing elements of the resulting prime implicates that are subsumed by some elements of $S$, we will obtain the set of minimal supports for $Z$.

Since any propositional formula can be transformed into a disjunction of CNF formulas,[5] the above technique allows us to compute the set of all minimal supports for any goal formula using the GEN-PI algorithm.

**Example** Consider the following set of clauses:

$$\mathcal{F} = \{A \Rightarrow B_1, A \Rightarrow B_2, A \Rightarrow B_3, A \Rightarrow B_4\}$$

The set of prime implicates of $\mathcal{F}$ is simply the union of $\mathcal{F}$ and the set of binary tautologies. To find the set of minimal supports for $(B_1 \wedge B_2) \vee (B_3 \wedge B_4)$, we apply $\Phi_{\{\{\overline{B}_1\},\{\overline{B}_2\}\}} \circ \Phi_{\{\{\overline{B}_3\},\{\overline{B}_4\}\}}$ to the prime implicates of $\mathcal{F}$ to get a new set of prime implicates $I_1 \cup I_2$ where $I_1 = \{\overline{A}, \overline{B}_1 \vee \overline{B}_2, \overline{B}_3 \vee \overline{B}_4\}$, and $I_2$ is the set of binary tautologies except $\overline{A} \vee A$. In this case, the set of prime implicates in $I_1 \cup I_2$ that are also implicates of $\mathcal{F}$ is exactly $I_2$. Hence, the set of minimal supports is $I_1$.

## 8 Conclusion

Traditional algorithms for prime implicate generation can only accept a CNF formula as their input. For problems that are naturally encoded by a conjunction of some DNF formulas, these algorithms can be needlessly expensive. This is because the transformation of the input formulas into their CNF equivalence is a very expensive process and can result in a dramatic explosion of the input size.

In this paper, we studied a more general algorithm GEN-PI for prime implicate generation. The algorithm is based on a $\Phi$ operation originally inspired by the order-theoretic study of the extended ATMS [Gunter et al., 1991]. It is superior to the existing approaches in that it allows a larger class of input formulas, namely, any conjunction of DNF formulas.

We also presented empirical comparisons between the GEN-PI algorithm and two existing implementations that are mainly based on the generalized consensus theory iTison, 1967]. In the more restrictive case of a CNF formula, the results show that the GEN-PI algorithm performs at least as efficiently as these algorithms, while it greatly outperforms them when the input contains some DNF formulas.

The usefulness of the GEN-PI algorithm is further demonstrated by its application in generating explanations for complex formulas in a propositional abductive reasoning system.

[5] Of course, a DNF formula or a CNF formula would suffice, but disjunction of CNF formulas allows a greater flexibility and better efficiency in encoding a formula (see previous section).

## References


[de Kleer and Williams, 1987] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. Artificial Intelligence, 32:97-130,1987

[de Kleer, 1986] Johan de Kleer. An assumption-basedTMS. Artificial Intelligence, 28:127-162,1986.

[de Kleer, 1992] Johan de Kleer. An improved incremental algorithm for generating prime implicates. Proc. of AAAI-92, pages 780-785,1992.

[Forbus, 1990] Kenneth D. Forbus. The qualitative process engine. In Readings in Qualitative Reasoning About Physical Systems, pages 220-235. Morgan Kaufmann, 1990.

[Gunter et al, 1991] Carl A. Gunter, Teow-Hin Ngair, Prakash Panangadcn, and Devika Subramanian The common order-theoretic structure of version spaces and ATMS's (extended abstract). Proc. ofAAAI-91, pages 500-505, 1991.

[Kean and Tsiknis, 1990] Alex Kean and George Tsiknis. An incremental method for generating prime implicants/implicates. J. Symbolic Computation, pages 185-206,1990.

[Kean and Tsiknis, 1992] Alex Kean and George Tsiknis. Assumption-based reasoning and clause management systems. Computational Intelligence, pages 1-24, 1992.

[McCarthy, 1980] J. McCarthy. Circumscription: A Form of Non-monotonic Reasoning. Artificial Intelligence, 13:27-39,1980.

[McCluskey, 1956] E. J. McCluskey. Minimization of boolean functions. Bell SystemTechnical Journal, 35:1417-1444,1956.

[McCluskey, 1965] E. J. McCluskey. Introduction to the Theory of Switching Circuits. McGraw-Hill, 1965.

[Ngair and Provan, 1992] Teow-Hin Ngair and Gregory Provan. Focusing ATMS for problem-solving: A formal approach. Technical Report MS-CIS-92-61, University of Pennsylvania, 1992.

[Ngair, 1992] Teow-Hin Ngair. Convex Spaces as an Order-theoretic Basis for Problem Solving. PhD thesis. Department of Computer and Information Science, University of Pennsylvania, July 1992.

[Quine, 1952] W. V. Quine. The problem of simplifying truth functions. American Mathematical Monthly, 59:521-531,1952.

[Reiter and de Kleer, 1987] Raymond Reiter and Johan de Kleer. Foundations of assumption-based truth maintenance systems: Preliminary report. Proc. ofAAAI-87, pages 183-188,1987.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. Artificial Intelligence, 13:81-132, 1980.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. Artificial Intelligence, 32:57-95,1987.

[Slagle etai, 1970] James R Slagle, Chin-Liang Chang, and Richard C. T. Lee. A new algorithm for generation prime implicants. IEEE Trans on Computers, C-19(4):304-310,1970.

[Tison, 1967] P. Tison. Generalization of consensus theory and application to the minimization of boolean functions. IEEE Trans. Electronic Computers, EC-16:100-105,1967.