On Computing Prime Implicants and Prime Implicates [†]

TR 92-3

Anavai Ramesh *George Becker* *Neil V. Murray*

Inst. for Programming & Logics
Dept. of Computer Science
State Univ. of N.Y. at Albany
Albany, NY 12222

rameshag / becker / nvm @cs.albany.edu

Abstract

Path dissolution is an inferencing mechanism for classical logic that efficiently generalizes the method of analytic tableaux. Two features that both methods enjoy are (in the propositional case) strong completeness and the ability to produce a list of essential models (satisfying interpretations) of a formula. The latter feature is particularly valuable in a setting in which one wishes to make use of satisfying interpretations rather than merely to determine whether any exist.

In this paper we describe a method for employing dissolution to compute the *prime implicants* **and the** *prime implicates* **of an arbitrary propositional formula.**

[†] This research was supported in part by National Science Foundation Grant CCR-9101208.

On Computing Prime Implicants and Prime Implicates †

Anavai Ramesh George Becker *Neil V. Murray*

1. Introduction

Path dissolution is an inferencing mechanism for classical logic that has several unusual properties. It works by selecting a link and restructuring the formula so that all paths through the link are eliminated. The nature of the restructuring is such that one cannot rely on CNF (conjunctive normal form): Even if a formula starts out in CNF, a single dissolution step produces an unnormalized formula. One consequence of eliminating all paths through a link is strong completeness: *Any* sequence of dissolution steps will eventually create a linkless formula. The paths that remain may be interpreted as models (satisfying interpretations) of the formula.

In this paper, we describe how to apply dissolution to the problem of computing the prime implicants and the prime implicates of a propositional formula. A key notion in this development is that of a linkless formula in *negation normal form* (NNF).

Other algorithms for computing prime implicants and prime implicates have been developed by Slagle, Chang, and Lee [8], by Jackson and Pais [1], and by Kean and Tsiknis [2]. Our algorithm combines the path-based techniques from [1] with techniques that are inference-based as in [2]. A key feature of the algorithm we develop is that it does not rely on CNF or DNF, as do the others.

In the next section we describe our path semantics viewpoint and our graphical representation of formulas in classical logic. A brief introduction to path dissolution is also given in this section.

2. Path Dissolution

We assume the reader to be familiar with the notions of *atom, literal*, and *formula* from classical logic. We consider only formulas in *negation normal form* (NNF): The only connectives used are conjunction and disjunction, and all negations are at the atomic level.

We introduce a number of technical terms and definitions in this section. Although the treatment is brief, the examples plus the reader's intuition should provide an adequate understanding of this background material.

2.1. Semantic graphs

A *semantic graph G* is a triple *(N,C,D) of nodes, c-arcs*, and *d-arcs*, respectively, where a node is a literal occurrence, a c-arc is a conjunction of two semantic graphs, and a d-arc is a disjunction of two semantic graphs. Any of N,C,D may be empty. If N is empty, G is either TRUE (empty conjunction) or FALSE (empty disjunction). Each semantic graph used in the construction of a semantic graph is called an *explicit subgraph*, and each proper explicit subgraph is contained in exactly one arc. Note that when a graph contains occurrences of TRUE and FALSE, the obvious truth-functional reductions apply. Unless otherwise stated, we will assume that semantic graphs are automatically so reduced, and that empty graphs are FALSE. We will typically use *G* to refer to both the graph and to the corresponding node set when the meaning is evident from context.

We use the notation $(X,Y)_c$ for the c-arc from *X* to *Y* and similarly use $(X,Y)_d$ for a d-arc; the subscript may be omitted when no confusion is possible.

In Figure 1 below, the formula on the left is displayed graphically on the right:

$((\neg C \wedge A) \vee D \vee E) \wedge (\neg A \vee (B \wedge C)) \equiv$

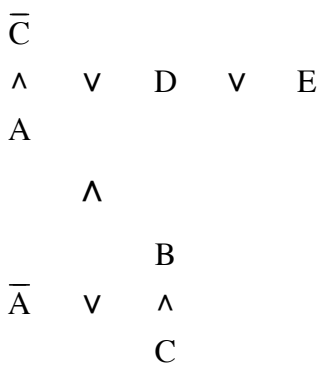


Figure 1.

Note that c-arcs and d-arcs are indicated by the usual symbols for conjunction and disjunction [†]. Essentially, the only difference between a semantic graph and a formula in NNF is the point of view, and we will use either term depending upon the desired emphasis. For a more detailed exposition, see [3].

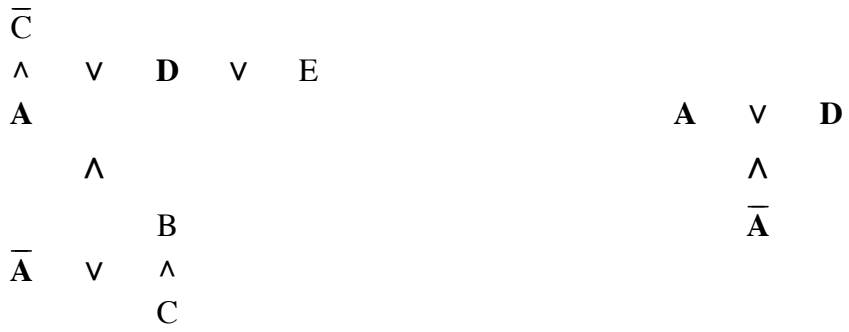
If A and B are nodes in a graph, and if $\mathbf{a} = (X,Y)_\alpha$ is an arc ($\alpha = c$ or $\alpha = d$) with A in X and B in Y, we say that **a** is the arc *connecting* A and B, and that A and B are α-*connected*. In Figure 1, C is c-connected to each of B, A, $\overline{C}$, D, and E and is d-connected to $\overline{A}$.

Let G be a semantic graph. A *partial c-path through G* is a set of nodes such that any two are c-connected, and a *c-path* through G is a partial c-path that is not properly contained in any partial c-path. The c-paths of the graph in Figure 1 above are: {$\overline{C}$, A, $\overline{A}$}, {$\overline{C}$, A, B, C}, {D, $\overline{A}$}, {D, B, C}, {E, $\overline{A}$}, {E, B, C}. We similarly define d-path using d-arcs instead of c-arcs. The following lemma is obvious.

Lemma 1. Let G be a semantic graph. Then an interpretation I satisfies (falsifies) G iff I satisfies (falsifies) every literal on some c-path (d-path) through G.

2.1.1. Subgraphs

We will frequently find it useful to consider subgraphs that are not explicit; that is, given any set of nodes, we would like to examine that part of the graph consisting of exactly that set of nodes. The previous example is shown below on the left; the subgraph relative to the set {A, D, $\overline{A}$,} is the graph on the right.



For a precise definition of subgraph, see [3].

2.1.2. Blocks

The most important subgraphs are the *blocks*. A *c-block H* is a subgraph of a semantic graph with the property that any c-path p that includes at least one node from *H must pass through H*; that is, the subset of p consisting of the nodes that are in *H* must be a c-path through *H*. A *d-block* is similarly defined with d-paths. In Figure 1, the subgraph relative to the node set {A,D,E,$\overline{A}$,C} is a c-block. However, it is not a d-block since the d-path {$\overline{A}$,B} restricted to the subgraph is {$\overline{A}$}, which is a proper sub-

path of $\{\overline{A},C\}$ in the subgraph.

A *full block* is a subgraph that is both a c-block and a d-block. One way to envision a full block is to consider conjunction and disjunction as n-ary connectives. Then a full block is a subset of the arguments of one connective, i.e., of one explicit subformula. For example, in Figure 1, $\{\overline{C},A,E\}$ is a full block. Full blocks may be treated as essentially explicit subgraphs, and the Isomorphism Theorem from [3] assures us that they are the only structures that may be so treated.

2.2. Path Dissolution

A *link* is defined to be a complementary pair of c-connected nodes. Path dissolution is in general applicable to collections of links; here we restrict attention to single links. Suppose then that we have literal occurrences A and $\overline{A}$ residing in conjoined subgraphs X and Y, respectively. Consider, for example, the link $\{A,\overline{A}\}$ in Figure 1. Then

$$X = \begin{matrix}\overline{C}\\ \wedge \\ A\end{matrix} \vee D \vee E \quad \text{and} \quad Y = \overline{A} \vee \begin{matrix}B\\ \wedge \\ C\end{matrix} .$$

The c-path complement of A with respect to X, written CC(A, X), is defined to be the subgraph of X consisting of all literals in X that lie on paths that do not contain A; the *c-path extension* of A with respect to X, written CPE(A, X), is the subgraph containing all literals in X that lie on paths that *do* contain A. In Figure 1, CC(A, X) = D ∨ E; CPE(A, X) = $\overline{C}$ ∧ A. It is intuitively clear that the paths through (X ∧ Y) that do not contain the link are those through (CPE(A, X) ∧ CC($\overline{A}$, Y)) plus those through (CC(A, X) ∧ CPE($\overline{A}$,Y)) plus those through (CC(A, X) ∧ CC($\overline{A}$, Y)). The reader is referred to [4] for the formal definitions of CC and of CPE and for the proofs of the lemmas below; there, they are stated in full generality for arbitrary subgraphs H. In this paper, we need these structures only with respect to single nodes, and the lemmas are restricted accordingly.

Lemma 2. The c-paths of CPE(A, G) are precisely the c-paths of G that include A.

Corollary. CPE(A, G) is exactly the subgraph of G relative to the set of nodes that lie on c-paths that include A.

Lemma 3. The c-paths of CC(A, G) are precisely the c-paths of G that do not

include A.

Corollary. CC(A, *G*) is exactly the subgraph of *G* relative to the set of nodes that lie on c-paths that do not include A.

Lemma 4. CC(A, *G*) ∪ CPE(A, G) = *G*.

Let $H = \{A, \overline{A}\}$ be a link, and let $M = (X, Y)_c$ be the smallest full block containing *H*. The only way that *H* can be a single c-block is if *H* is a full block (it is trivially a d-block). In that case, *H* = *M*, and A and $\overline{A}$ must be (up to commutations and reassociations) arguments of the same conjunction. We define DV(*H, M*), the *dissolvent of H in M*, as follows: If *H* is a single c-block, then DV(*H, M*) = CC(A, *M*) = CC($\overline{A}$, *M*) = ∅. Otherwise (i.e., if *H* consists of two c-blocks),

$$\mathrm{DV}(H, M) = \begin{array}{c} \mathrm{CPE}(A, X) \\ \wedge \\ \mathrm{CC}(\overline{A}, Y) \end{array} \vee \begin{array}{c} \mathrm{CC}(A, X) \\ \wedge \\ \mathrm{CPE}(\overline{A}, Y) \end{array} \vee \begin{array}{c} \mathrm{CC}(A, X) \\ \wedge \\ \mathrm{CC}(\overline{A}, Y) \end{array}$$

It follows from the corollaries and Lemma 4 that either of the two graphs shown below may also be used for DV(*H*, *M*).

$$\begin{array}{c} X \\ \wedge \\ \mathrm{CC}(\overline{A}, Y) \end{array} \vee \begin{array}{c} \mathrm{CC}(A, X) \\ \wedge \\ \mathrm{CPE}(\overline{A}, Y) \end{array} \quad \text{or} \quad \begin{array}{c} \mathrm{CC}(A, X) \\ \wedge \\ Y \end{array} \vee \begin{array}{c} \mathrm{CPE}(A, X) \\ \wedge \\ \mathrm{CC}(\overline{A}, Y) \end{array}$$

The three versions of DV(*H, M*) are not identical as graphs, but all three do have the identical c-paths: all those of the original full block *M* except those of CPE(A, *X*) ∧ CPE($\overline{A}$, *Y*), i.e., except those through the link.

Theorem 1. Let *H* link in a semantic graph *G*, and let M be the smallest full block containing *H*. Then *M* and DV(*H*, *M*) are equivalent.

A proof of Theorem 1 (in its full generality) can be found in [4].

We may therefore select an arbitrary link *H* in *G* and replace the smallest full block containing *H* by its dissolvent, producing (in the ground case) an equivalent graph. We call the resulting graph the *dissolvent of G with respect to H* and denote it Diss(*G,H*). Since the paths of the new graph are all that appeared in *G* except those that contained the link, this graph has strictly fewer c-paths than the old one. As a result, finitely many dissolutions (bounded above by the number of c-paths in the

original graph) will yield a linkless equivalent graph. This proves

Theorem 2. At the ground level, path dissolution is a strongly complete rule of inference.

3. Fully Dissolved Formulas

If we dissolve in semantic graph G until it is linkless, we call the resulting graph the *full dissolvent of G* and denote it by FD(G). Observe that FD(G) is dependent on the order in which links are activated. However, the set of c-paths in FD(G) is unique: It is exactly the set of satisfiable c-paths in G.

Because FD(G) is link-free, the consequences of G are represented in the d-paths of FD(G). This is clarified by Theorem 4 below.

4. Prime Implicates and Prime Implicants

Recently, Jackson and Pais [1] developed a new algorithm for computing the consequences of a ground formula. The algorithm is path-based and stems from ideas surrounding Bibel's connection method. Consequences expressed as minimal clauses that are implied by a formula are its *prime implicates*; minimal conjunctions that imply a formula are its *prime implicants*. Implicates are useful in certain approaches to non-monotonic reasoning [5,7], where all consequences of a formula set (e.g., the support set for a proposed commonsense conclusion) are required. The implicants are useful in situations where satisfying models are desired, as in error analysis during hardware verification.

We introduce the algorithm PI in this section, and we begin with some basic definitions.

4.1. Definitions

A conjunction P (of literals) is an implicant of a formula G, iff $P \models G$.

A conjunction C subsumes another C′ iff $C' \models C$.

If conjunction C′ is not equivalent to *false* then C subsumes C′ iff $C \subseteq C'$. *False* is subsumed by all conjunctions. A *false* conjunction can subsume another *false* conjunction only.

Conjunction C is a prime implicant of a formula G iff

1) C is not *false*
2) C is an implicant of G and
3) and for all literals l_i in C, $(C - \{l_i\}) \not\models G$.

A disjunction P (of literals) is an implicate of a formula G, iff $G \models P$.

A disjunction D subsumes another D′ iff $D \models D'$.

If disjunction D′ is not equivalent to *true* then D subsumes D′ iff $D \subseteq D'$. *True* is subsumed by all disjunctions. A *true* disjunction can subsume another *true* disjunction only.

Disjunction D is a prime implicate of a formula G iff

1) D is not *true*
2) D is an implicate of G and
3) and for all literals l_i in D, $G \not\models (D - \{l_i\})$

In the discussion that follows, we will often refer to subsumption of d- and c-paths rather than of disjuncts and conjuncts. Paths are defined as sets of literal occurrences, but with regard to subsumption, we consider the *literal sets* of paths. In this way, no change in the standard definitions is necessary.

In the proof of Theorem 3 below, we make use of the following lemma:

Lemma 5. Let G be a semantic graph, and let H be a subgraph of G such that all c-paths through H are only partial c-paths in G. Then there exists a d-path $\mathbf{q}$ through G that does not meet H.

Proof: By induction on the size of G. If G consists of a single literal L, H must be empty and {L} is the required d-path through G. So suppose G is a disjunction or a conjunction of X and Y, and let $H = H_x \cup H_y$, where H_x is H restricted to X and H_y is H restricted to Y.

If $G = X \vee Y$, then all c-paths of H_x and of H_y are only partial in X and in Y, respectively. By the induction hypothesis there is a d-path $\mathbf{q}_x$ through X that does not meet H_x and a d-path $\mathbf{q}_y$ through Y that does not meet H_y. But then $\mathbf{q}_x\mathbf{q}_y$ is a d-path through G that does not meet H.

If $G = \overset{X}{\underset{Y}{\wedge}}$, then all c-paths of at least one of H_x and H_y (say H_x) are only partial in (say) X. (Otherwise, we could choose a c-path from each of H_x and H_y that is a c-path of X and of Y respectively, from which we immediately obtain a c-path of H that is also

a c-path through G, contrary to the hypothesis.) By the induction hypothesis there is a d-path $\mathbf{q}_x$ through X that does not meet H_x. But $\mathbf{q}_x$ is a d-path through G, and the proof is complete. □

Theorem 3. In any non-empty formula G in which no d-path contains a link, every implicant of G is subsumed by some c-path of G.

Proof: Let C be an implicant of G. Since $C \models G$, $G \vee \neg C$ is valid and is therefore spanned by its full set of (disjunctive) links. Let R be the set of all literals in G that are linked to $\neg C$. It suffices to show that G_R contains a c-path through G since $R \subseteq C$. So assume that all c-paths of G_R are partial in G. There is a d-path $\mathbf{q}$ in G that does not meet G_R by Lemma 1. But since $G \vee \neg C$ is spanned, and since G is linkless, some literal in $\mathbf{q}$ is linked to a literal in $\neg C$, contradicting the maximality of R. □

Corollary: Every prime implicant of a reduced CNF formula (with no true disjuncts) is subsumed by some c-path in the formula.

This follows from theorem 3 as such a CNF formula has no d-paths with links.

Theorem 4. In any non-empty formula in which no c-path contains a link, every implicate of the formula is subsumed by some d-path in the formula.

Proof: The proof is similar to that of theorem 3, replacing conjunct by disjunct, implies by implied by, unsatisfiable by valid, satisfiable by falsifiable, c-path by d-path, true by false, implicant by implicate, and $P \not\models G$ by $G \not\models P$. □

Corollary: Every prime implicate of a reduced DNF formula (with no false conjuncts) is subsumed by some d-path in the formula.

This follows from Theorem 4 as such a DNF formula has no c-paths with links.

Theorem 5. The full dissolvent FD(G) (with respect to either c-links or d-links) of a graph G, is equivalent to G.

Theorem 6. Every c-path through a graph is an implicant of the graph.

Theorem 7. Every d-path through a graph is an implicate of the graph.

Theorem 8. Suppose G is a non-null graph without d-links and suppose

$\Psi(G) = \{P \mid P \text{ is a c–path through } G \wedge P \neq \text{false} \wedge (\forall \text{ c–paths } Q \text{ through } G, Q \not\subset P)\}$,

then $\Psi(G)$ is the set of all prime implicants of G.

Theorem 9. Suppose G is a non-null graph without c-links and suppose

$\psi(G) = \{P \mid P \text{ is a d–path through } G \wedge P \neq \text{true} \wedge (\forall \text{ d–paths Q through } G, Q \not\subset P)\}$,

then $\psi(G)$ is the set of all prime implicates of G.

4.2. PI - An Algorithm to Compute Prime Implicates

This algorithm is an extension of Jackson and Pais' [1] MM algorithm to NNF formulas. Depending upon whether the input formula is in CNF or DNF, and upon whether prime implicates or implicants are desired, the MM algorithm may have to be run twice.

One advantage of the PI algorithm is the avoidance of both CNF and DNF; these normal forms can be exponentially larger than equivalent formulas in NNF. In situations where, if used alone, PI would also be run twice, dissolution may be used in place of the first run. Resolution would also suffice, but dissolution has a much more efficient test for termination, because it is strongly complete. However, resolution requires and maintains CNF whereas dissolution does neither.

The algorithm computes the d-paths which are neither tautologies nor subsumed by other d-paths of a graph. From Theorem 9, we know that all prime implicates of the formula G are exactly those d-paths in the full dissolvent FD(G), (with respect to c-paths) which are not subsumed by other d-paths and are not tautologies.

We first get the (non null) full dissolvent FD(G), which is in NNF. The algorithm PI then recursively traverses the semantic graph of FD(G) (which is represented as a binary tree) in a left to right, bottom up manner computing partial paths on the way, and eliminating tautologies and subsumed paths when they are created. The input of each recursive call is a subgraph and the partial d-paths of the portion of the graph to the left of it in the tree. We assume for expository purposes that all logical operators are binary. The algorithm can be trivially extended to operators of arbitrary arity.

When the subgraph is a single literal, each path is extended (if it does not form a tautology) by this literal. We say it is *trivially extended* if the literal is already in the partial path, *properly extended* otherwise, just as in the MM algorithm. Only properly extended paths have to checked for subsumption by trivially extended paths as in the MM algorithm.

In the case of a disjunct, PI first extends the paths with respect to the first operand, and then extends the resulting paths over the second. In the case of a conjunct PI extends the paths through each of the operands separately, and then combines them while eliminating any subsumed paths.

Note that, for a conjunct, any path which has been trivially extended by the first operand will subsume all extensions of this path by the second operand, and hence these paths are excluded when calling PI on the second operand.

PI is defined as follows.

```
PI(Paths, G)
   if (Paths = ∅) return ∅
   if  ( G is a literal) then
       Paths'' := ∅
       Paths' := ∅
       for  P in Paths do
          if  ( G ∈ P) then
             Paths' := Paths'∪ P
          else
             if  ( ¬G ∉ P) then
                Paths'' := Paths'' ∪( P ∪ {G})
             endif
          endif
       endfor
       Paths'' :=  Paths'∪ ( Paths'' - { P ∈ Paths'' | ∃ P'∈ Paths'∧ P'⊂ P})
       return Paths''
   else if ( G = ( X , Y)c) then
          Paths' := PI(Paths, X)
          Paths'' := PI( (Paths - Paths' ), Y)
          Paths'' := (Paths'∪ Paths'') – { P | P ∈ Paths' ∧ ∃ P' ∈ Paths'' ∧ P' ⊂ P}
                       – { P | P  ∈ Paths'' ∧ ∃ P' ∈ Paths' ∧ P' ⊂ P}
          return Paths''
   else if ( G = ( X , Y)d) then
          Paths' := PI(Paths, X)
          Paths'' := PI(Paths', Y)
          return Paths''
   endif
```

Theorem 10. Given a formula FD(*G*) in NNF, PI computes all and only the prime implicates of FD(*G*).

Proof sketch: We show that the recursive function PI({∅}, FD(*G*)) computes ψ(FD(*G*)), where FD(*G*) is the full dissolvent with respect to c-links. The correctness of the algorithm PI then follows from Theorems 5 and 9.

The prime implicates of a formula G can be computed in any of the three following ways using PI:

Option 1

1. Compute G', the full dissolvent of G
   (with respect to c-links)
2. If G' is empty
   then the only prime implicate is *false*
   else call PI($\{\varnothing\}$, G')

Option 2

1. Compute $G' = \neg\,\mathrm{PI}(\{\varnothing\}, \neg G) = \mathrm{FD}(G)$ $(\not\equiv \mathrm{FD}(G))$
2. If G' is empty
   then the only prime implicate is *false*
   else call PI($\{\varnothing\}$, G')

Option 3

1. Compute G', the full dissolvent of G
   (with respect to c-links)
2. If G' is empty
   then the only prime implicate is *false*
   else compute $G'' = \neg\mathrm{FD}(\neg G'))$
   (with respect to d-links of G')
3. If G'' is empty
   then G'' (hence G) is a tautology and has no prime implicates
   else call PI′($\{\varnothing\}$, G'')
   where PI′ = PI without the tautology check

We may compute prime implicants in the same manner as for prime implicates: The steps are the same except that the given graph is dissolved on d-links, *true* is replaced by *false* and a modified version of PI (obtained by replacing $(X,Y)_{\mathrm{d}}$ by $(X,Y)_{\mathrm{c}}$ and vice versa) is used.

Theorem 11. Given a formula G in NNF, the modified version of PI computes all and only the prime implicants of G.

4.3. Potential optimizations

Currently PI does not use any heuristic such as that used by the MM algorithm. However, we hope that similar heuristics will eliminate subsumed paths and tautologies (or contradictions when computing implicants) earlier. We expect that one can also use heuristics in conjunction with dissolution so as to eliminate some redundant paths.

References

1. Jackson, P., and Pais, J. Computing Prime Implicants. *Proceedings of the 10th International Conference on Automated Deduction*, Kaiserslautern, W. Germany, July, 1990. In *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Vol. 449, 543-557.

2. Kean, A. and Tsiknis, G. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation* **9** (1990), 185-206.

3. Murray, N.V., and Rosenthal, E. Inference with path resolution and semantic graphs. *J.ACM* 34,2 (April 1987), 225-254.

4. Murray, N.V., and Rosenthal, E. Path dissolution: A strongly complete rule of inference. *Proceedings of the 6th National Conference on Artificial Intelligence*, Seattle, WA, July 12-17, 1987, 161-166.

5. Przymusinski, T.C. An algorithm to compute circumscription. *Artificial Intelligence* **38** (1989), 49-73.

6. Ramesh, A., Becker, G., and Murray, N.V. On computing prime implicants and prime implicates. Technical Report TR 92-3, SUNY at Albany, January, 1992.

7. Reiter, R. and de Kleer, J. Foundations of assumption-based truth maintenance systems: preliminary report. *Proceedings of the 6th National Conference on Artificial Intelligence*, Seattle, WA, July 12-17, 1987, 183-188.

8. Slagle, J.R., Chang, C.L., and Lee, R.C.T. A new algorithm for generating prime implicants. *IEEE Transactions on Computers,* **C-19(4)** (1970), 304-310.