Complexity Results for POMSET Languages

Extended Abstract – CAV '91 proceedings[1]

Joan Feigenbaum
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Jeremy A. Kahn
Math Department
University of California
Berkeley, CA 97420

Carsten Lund
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Abstract

Pratt [13] introduced POMSETs (partially ordered multisets) in order to describe and analyze concurrent systems. A POMSET P gives a set of temporal constraints that any correct execution of a given concurrent system must satisfy. Let $L(P)$ (the *language of* P) denote the set of all system executions that satisfy the constraints given by P. We show the following for finite POMSETs P, Q, and system execution x.

- The POMSET Language Membership Problem (given x and P, is $x \in L(P)$?) is NP-complete.
- The POMSET Language Containment Problem (given P and Q, is $L(P) \subseteq L(Q)$?) is Π_2^p-complete.
- The POMSET Language Equality Problem (given P and Q, is $L(P) = L(Q)$?) is at least as hard as the graph-isomorphism problem.
- The POMSET Language Size Problem (given P, how many x are in $L(P)$?) is span-P-complete.

1 Introduction

Verification of concurrent systems has been studied as a formal language-containment problem for a number of years [1, 15, 5]. In this formulation, one is given a model M represented by a finite transition structure such as a finite state machine, automaton or Petri net (sometimes termed an *implementation*), together with an abstraction A of the model, represented by an automaton or logic formula (sometimes termed a *specification*, defining a property to be proved about the model M). The verification problem consists of testing whether $L(M) \subseteq L(A)$, where $L(X)$ is the formal language associated with X. Typically, M is large and therefore defined implicitly in terms of components. An inherent difficulty in this approach is the computational complexity of the language containment test as a function of the size of the representation of M in terms of components. For example, if M is defined in terms of coordinating state machines, then the size of M grows geometrically with the number of components defining it, and the language containment

[1] Because of space limitations, some of the results in this extended abstract are stated without proof. All proofs are given in the journal version of the paper, which is available in preprint form from the first author.

problem is PSPACE-complete [6, AL6, page 266]. This computational complexity issue has been addressed by a number of heuristics, notably homomorphic reduction [11, 10], inductive methods [2, 12], binary decision diagrams [4, 3, 16], and partial orders [7, 14].

In this paper, we consider the language containment problem for POMSETs (partially ordered multisets), which were introduced by Pratt [13]. Both the implementation and the specification of a system can be represented by POMSETs as follows. Let Σ denote a finite set of *actions* that the system can perform. So actions are things like "send 0 to processor p," "receive message m from processor q," and "wait." Each *vertex* v in the POMSET P corresponds to a distinct *event*. Intuitively, an event is a logical "step" taken by the system. The *label* $l(v)$ is an element of Σ, and distinct vertices may have the same label; this corresponds to the fact that a given action (say "send 0 to processor p") may be performed several times by the system during any execution. Each *arc* (v, w) in P represents a *constraint* of the form "event v must occur before event w in any execution of the system." For example, if $l(v)$ is "receive message m from processor p," and $l(w)$ is "if the value of register r is equal to m then signal processor q_1, else signal processor q_2," then the arc (v, w) has the obvious interpretation. The *language* of P is simply the set of all correct executions of the system.

The following example motivates the use of POMSETs. The language $L = \{ab_{i_1}b_{i_2} \cdots b_{i_n}a\}$, where $i_1 i_2 \cdots i_n$ is a permutation of $12 \cdots n$ and all of the b_i's are distinct, arises often in the description of concurrent processes. Its meaning is "perform action a, then perform each of the actions b_1 through b_n in any order, then perform action a again." An NFA that accepts L must have at least 2^n states. POMSETs, however, offer a much more compact representation: The $(n+2)$-node POMSET of Figure 1 represents L.

Formally, the problem of interest is: Given POMSETs P and Q, is the language of P a subset of the language of Q? We call this the PLC problem, for POMSET Language Containment.

The POMSET P represents the implementation and Q the specification. We show that the PLC problem is Π_2^p-complete.

Note that P and Q are both finite POMSETs. Thus the languages in question are finite, and the strings in them are of finite length. If we were presenting an algorithm for PLC, this finiteness restriction would render the algorithm impractical, because real concurrent systems produce infinite sets of infinite sequences. However, we are giving a lower bound on the complexity of PLC, and hence the finiteness restriction makes our result all the more meaningful: Even in this restricted case, the problem appears to be intractable.

We also give an NP-completeness result for the following simpler problem: Given a

POMSET P and a string x, is x in the language of P? This is called the PLM problem, for POMSET Language Membership.

Once again, the finiteness restriction only strengthens our result, because we are providing a lower bound rather than an algorithm.

In the journal version of this paper, we also consider the following two problems. The POMSET Language Equality problem (PLE) is: Given two POMSETs P and Q, is the language of P equal to the language of Q? The POMSET Language Size problem (PLS) is: Give a POMSET P, what is the number of strings in the language of P? We show that PLE is at least as hard as the graph isomorphism problem and that PLS is complete for the complexity class span-P (cf. Köbler, Schöning, and Toran [9]).

2 Definitions and Notation

Throughout this paper, P and Q denote (finite) POMSETs, and x denotes a (finite) string. We now fix these ideas precisely.

Definition 2.1 *A* **POMSET** *P is a triple (V, A, l). The* **vertex set** *$V(P)$ consists of a finite number n of distinct elements $\{v_1, \ldots, v_n\}$, called the* **events**. *The* **arc set** *$A(P)$ consists of a set of ordered pairs (v, w), where v and w are distinct elements of V, called the* **constraints**. *The directed graph $(V(P), A(P))$ is acyclic. The mapping $l : V \to \Sigma$ assigns an* **action** *to each event in V, and $l(v)$ is called the* **label** *of vertex v.*

Recall that a linear ordering on $V = \{v_1, \ldots, v_n\}$ *extends* a partial ordering of V if, for all pairs v_i, v_j of distinct elements in V, $v_i < v_j$ in the partial ordering implies that $v_i < v_j$ in the linear ordering. Technically, a DAG (directed acyclic graph) may not be a partial ordering, because it may not be transitively closed. When we say that a linear ordering on V extends the DAG (V, A), we mean that it extends the transitive closure of the DAG.

Definition 2.2 *The* **language** *$L(P)$ of a POMSET $P = (V, A, l)$ is a subset of Σ^n, where $n = |V(P)|$. The string $\sigma_1 \cdots \sigma_n$ is in $L(P)$ if there is a linear ordering $v_{i_1} \cdots v_{i_n}$ of the vertex set V that extends the DAG (V, A) and satisfies $l(v_{i_j}) = \sigma_j$, for $1 \le j \le n$.*

3 PLC is Π_2^p-Complete

Theorem 3.1 *The* PLC *problem is* Π_2^p-complete.

Proof: First note that it is obvious that PLC is in Π_2^p. Suppose that we wish to know whether $L(P)$ is contained in $L(Q)$, where $V(P) = \{v_1, \ldots, v_n\}$ and $V(Q) = \{w_1, \ldots, w_n\}$. The following is a Π_2^p expression for $L(P) \subseteq L(Q)$: For all linear orderings $v_{i_1} \cdots v_{i_n}$, there exists a linear ordering $w_{j_1} \cdots w_{j_n}$ such that if $v_{i_1} \cdots v_{i_n}$ extends $A(P)$, then $w_{j_1} \cdots w_{j_n}$ extends $A(Q)$ and $l(v_{i_k}) = l(w_{j_k})$ for $1 \leq k \leq n$. The hypothesis "if $v_{i_1} \cdots v_{i_n}$ extends $A(P)$" is equivalent to "if $l(v_{i_1}) \cdots l(v_{i_n}) \in L(P)$," and the conclusion "then $w_{j_1} \cdots w_{j_n}$ extends $A(Q)$ and $l(v_{i_k}) = l(w_{j_k})$ for $1 \leq k \leq n$" is equivalent to "$l(w_{i_1}) \cdots l(w_{i_n}) \in L(Q)$ and is equal to $l(v_{i_1}) \cdots l(v_{i_n})$."

It is also obvious that PLC is NP-hard, because PLM is the special case of PLC in which $L(P)$ contains just one string, and PLM is NP-complete (see Section 4 below).

We show Π_2^p-completeness by reduction from the following Π_2^p-complete problem (cf. [6, page 166]).

Normalized B_2^c:

Input : Two sets $\{w_1, \ldots, w_m\}$ and $\{y_1, \ldots, y_n\}$ of boolean variables and a set $\{c_1, \ldots, c_k\}$ of clauses. Each clause is of the form $a \Rightarrow b \vee c \vee d$, where a is either w_i or $\overline{w_i}$ for some i and each of b, c, and d is y_j or $\overline{y_j}$ for some j.

Question : Is it the case that, for every truth assignment to the w_i's, there exists some truth assignment to the y_j's such that every c_l is satisfied?

Given an instance $(W = \{w_1, \ldots, w_m\}, Y = \{y_1, \ldots, y_n\}, C = \{c_1, \ldots, c_k\})$ of normalized B_2^c, we construct an instance (P, Q) of PLC as follows.

In $V(P)$, there are three disjoint sets of vertices. The first group contains n vertices, labeled y_1 through y_n. The second group in $V(P)$ contains $2m+k$ vertices. For $1 \leq i \leq m$, there are two vertices in this group labeled w_i; we refer to them as "the positive w_i vertex" and "the negative w_i vertex." For $1 \leq l \leq k$, there is one vertex in the second group labeled c_l. The third group of vertices in $V(P)$ is of size $n + 3k$. There is one vertex in this group labeled y_j, for $1 \leq j \leq n$, and there are three vertices in the third group labeled c_l, for $1 \leq l \leq k$. For every clause c_l in which w_i appears on the left side of the implication, there is an arc in $A(P)$ from the positive w_i vertex to the second-group vertex labeled c_l; for every c_l in which $\overline{w_i}$ appears on the left side of the implication, there is an arc in $A(P)$ from the negative w_i vertex to the second-group vertex labeled c_l. Every w vertex in the second group is joined by an arc to every c vertex in the third group. The rest of the arcs that make up $A(P)$ can be seen in Figure 2, where an example of this construction is given. The subscripts are omitted from the labels of some clause vertices in order to reduce clutter.

In $V(Q)$, there are two vertices labeled y_j, for $1 \leq j \leq n$, and two vertices labeled w_i, for $1 \leq i \leq m$. These are referred to as "the positive y_j (resp. w_i) vertex" and "the negative y_j (resp. w_i) vertex." $V(Q)$ also contains four vertices labeled c_l, for $1 \leq l \leq k$. One group of these c vertices is associated with the y vertices; each c vertex in this group has in-degree 1. For each clause c_l in which the literal y_j appears on the right side of the implication, there is an arc from the positive y_j vertex to a c_l vertex. Similarly, for each clause c_l in which the literal $\overline{y_j}$ appears on the right side of the implication, there is an arc from the negative y_j vertex to a c_l vertex. Note that each label c_l appears three times in this group, once for each literal in the clause. The second group of c vertices is associated with the w vertices; each c vertex in this group has in-degree 2. If w_i or $\overline{w_i}$ appears on the left side of the implication in clause c_l, then there are arcs from both the positive w_i vertex and the negative w_i vertex to the c_l vertex in the second group. See Figure 3 for an example of this construction. Once again, subscripts are omitted from some clause vertices to reduce clutter.

Suppose that (P, Q) is a yes-instance of PLC; so $L(P)$ is contained in $L(Q)$. We must show that (W, Y, C) is a yes-instance of B_2^c. Choose an assignment of truth values to the variables in W. We will construct an assignment of truth values to the variables in Y that, together with the initial assignment to those in W, satisfies all the clauses in C.

Consider the string

$$x = y_1 \cdots y_n w_1 \cdots w_m c_{q_1} \cdots c_{q_t} y_1 \cdots y_n w_1 \cdots w_m c_{q_{t+1}} \cdots c_{4k}$$

in $L(P)$ that is formed as follows. The prefix $y_1 \cdots y_n$ comes from the first group of vertices in $V(P)$. In the first substring $w_1 \cdots w_m$, each w_i represents a choice between the positive w_i vertex and the negative w_i vertex within the second group in $V(P)$. The substring $c_{q_1} \cdots c_{q_t}$ corresponds exactly to the clauses that are nontrivial to satisfy: If a clause vertex v in the second group in $V(P)$ is adjacent to the positive w_i vertex and w_i is TRUE in the initial assignment, then $l(v)$ goes into the substring $c_{q_1} \cdots c_{q_t}$; similiarly, if v is adjacent to the negative w_i vertex and w_i is FALSE in the initial assignment, then $l(v)$ goes into the substring $c_{q_1} \cdots c_{q_t}$. The rest of the string x is constructed in any way that is consistent with the constraints in $A(P)$, subject to y's, then w's, then c's.

Note that x is always in $L(P)$. Because (P, Q) is assumed to be a yes-instance of PLC, x is also in $L(Q)$. Consider the vertices $v(c_{q_1})$, ..., $v(c_{q_t})$ in $V(Q)$ that give rise to the substring $c_{q_1} \cdots c_{q_t}$ of x. These vertices must all be in the first group of c vertices in Q – that is, they must be in the group whose incoming arcs start with y's. This is because none of c_{q_1}, ..., c_{q_t} is preceded in x by two occurrences of w_i, for any i. If $v(c_{q_l})$

is connected to the positive (resp. negative) y_j vertex, then assign the variable y_j the value TRUE (resp. FALSE). Assign arbitrary values to any remaining y variables. Note that no conflicts arise in making this assignment – that is, each y_j is assigned one value. This is because each y_j symbol appears once in the prefix of x, and hence only one of the two y_j vertices is used; if the y_j vertex that's used is adjacent to two vertices $v(c_{q_{l_1}})$ and $v(c_{q_{l_2}})$, then either y_j appears in both $c_{q_{l_1}}$ and $c_{q_{l_2}}$ or $\overline{y_j}$ appears in both $c_{q_{l_1}}$ and $c_{q_{l_2}}$. This assignment, together with the initial assignment to the w variables, satisfies all of the clauses in C. Because the initial assignment to the w variables was arbitrary, this shows that (W, Y, C) is a yes-instance.

Now suppose that (W, Y, C) is a yes-instance of normalized B_2^c. Let x be an arbitrary element of $L(P)$ in the corresponding instance of PLC. We must show that x is also in $L(Q)$.

We construct a truth assignment that corresponds to x as follow. Each symbol in x comes from a vertex in a linear ordering of $V(P)$ that extends $A(P)$. Take the first occurrence of w_i in x, and see whether it corresponds to the positive w_i vertex or the negative w_i vertex. If positive, assign the variable w_i the value TRUE and, if negative, assign it FALSE. Because (W, Y, C) is a yes-instance, there must be an assignment of truth values to the y variables that, together with the assignment to the w's, satisfies every clause in C. This assignment to the y's corresponds to the prefix $y_1 \cdots y_n$ of x in a way that will become clear below. Denote by A the full assignment to y's and w's.

Call a y vertex or w vertex in $V(Q)$ "active" if it corresponds to the truth assignment A – e.g., the positive y_j vertex is active if and only if the variable y_j is TRUE in A. Now Q is the disjoint union of subPOMSETs Q_1 and Q_2, where Q_1 contains exactly the active y vertices and the c vertices that are connected by arcs from active y vertices, and Q_2 contains exactly the active w vertices and the c vertices that are connected by arcs from active w vertices.

The only nontrivial task involved in finding a linear ordering of $V(Q)$ that extends $A(Q)$ and gives rise to x is this: Suppose that clause c_l contains the variable w_i and that the first occurrence of the symbol c_l in x falls between the two occurrences of the symbol w_i; what is the vertex in $V(Q)$ that gives rise to this first occurrence of c_l? By construction, this vertex can be found in $V(Q_1)$ – that is, the active y vertices correspond to the prefix $y_1 \cdots y_n$ of x. Thus x is in the shuffle of $L(Q_1)$ and $L(Q_2)$, which is $L(Q)$. ∎

There are some special cases of PLC that are easily solved in polynomial time. For example, if each element of Σ occurs at most once as a label in each POMSET, then there is at most one bijection ϕ from $V(Q)$ to $V(P)$, given by the labels. If no such ϕ

exists, $L(P) \not\subseteq L(Q)$. Otherwise, let $T(P)$ (resp. $T(Q)$) be the transitive closure of $A(P)$ (resp. $A(Q)$). It is easily seen that $L(P)$ is contained in $L(Q)$ if and only if, for every arc (v, w) in $T(Q)$, the arc $(\phi(v), \phi(w))$ is in $T(P)$. We call this the *unique-label case* of PLC.

Similarly, the *no-autoconcurrence case* of PLC is solvable in polynomial time. "No autoconcurrence" means that, if v and w are in $V(P)$ (resp. $V(Q)$), and $l(v) = l(w)$, then either (v, w) or (w, v) is in $A(P)$ (resp. $A(Q)$). The no-autoconcurrence case can be reduced to the unique-label case as follows: For each $a \in \Sigma$, let $v_1, \ldots, v_m$ be all of the vertices of POMSET P with label a. These vertices must be linearly ordered in $A(P)$, or else there would be autoconcurrence. If the linear order is $v_{i1} < \cdots < v_{im}$, then relabel these vertices $l(v_{i1}) = a_{i1}, \ldots, l(v_{im}) = a_{im}$, where the a_{ij}'s are not in Σ. Do the same for all of the vertices with label a in Q, once again using the labels $a_{i1}, \ldots, a_{im}$.

4 PLM is NP-Complete

Theorem 4.1 *The* PLM *problem is* NP-complete.

Proof: Once again, it is obvious that PLM is in NP. To verify that $x = \sigma_1 \cdots \sigma_n$ is in $P = (V, A)$, where $V = \{v_1, \ldots, v_n\}$, simply guess a linear ordering $v_{i_1} \ldots v_{i_n}$ of V, and check that each arc in A joins a pair of vertices $v_{i_{j_1}}$, $v_{i_{j_2}}$ with $j_1 < j_2$ and that $l(v_i) = \sigma_i$ for each i.

We show completeness by reduction from the archetypal NP-complete problem 3SAT. Recall the statement of this problem.

Three Satisfiability (3SAT):

Input : Clauses $c_1, \ldots, c_n$ on boolean variables $y_1, \ldots, y_m$. Each c_j is of the form $c_{j_1} \vee c_{j_2} \vee cj_3$, where each c_{j_k} is either y_i or $\overline{y_i}$ for some i.

Question : Is there an assignment of truth values to the variables $y_1, \ldots, y_m$ that satisfies all of the clauses $c_1, \ldots, c_n$ simultaneously?

Given an instance $(C = \{c_1, \ldots, c_n\}, Y = \{y_1, \ldots, y_m\})$ of 3SAT, we construct an equivalent instance (x, P) of PLM as follows. The vertex set V of P contains two vertices, say v_{i_1} and v_{i_2}, for each variable y_i and three vertices, say w_{j_1}, w_{j_2}, and w_{j_3}, for each clause c_j. Vertices v_{i_1} and v_{i_2} have label y_i, and vertices w_{j_1}, w_{j_2}, and w_{j_3} all have label c_j. For each clause c_j, consider the variables (say y_r, y_s, and y_t) that occur in c_j. Put in exactly one of arcs (v_{r_1}, w_{j_1}) and (v_{r_2}, w_{j_1}) (resp. $[(v_{s_1}, w_{j_2})$ and $(v_{s_2}, w_{j_2})]$ and $[(v_{t_1}, w_{j_3})$

and $(v_{t_2}, w_{j_3})]$), by choosing the first if y_r (resp. y_s and y_t) occurs in c_j and the second if $\overline{y_r}$ (resp. $\overline{y_s}$ and $\overline{y_t}$) occurs in c_j. The string in the PLM instance is

$$x = y_1 \cdots y_m c_1 \cdots c_n y_1 \cdots y_m c_1 c_1 c_2 c_2 \cdots c_n c_n.$$

See Figure 4 for an example of this construction.

It is easily seen that (x, P) is a yes-instance of PLM if and only if (C, A) is a yes-instance of 3SAT. The key point is that the choice of vertices that map to the prefix $y_1 \cdots y_m$ of x corresponds exactly to the choice of truth values in the satisfying assignment and that this choice "covers" the first occurrence of each c_j symbol in x. ∎

An alternative proof of Theorem 4.1, based on a reduction from the CLIQUE problem, was subsequently given by Kilian [8].

In the journal version of this paper, we show that the special case of PLM in which each label in Σ occurs at most twice is solvable in polynomial time.

5 Results on PLE and PLS

Proofs of the following two theorems are given in the journal version.

Theorem 5.1 *The* PLE *problem is as hard as graph isomorphism.*

Theorem 5.2 *The* PLS *problem is* span-P-*complete.*

6 Discussion

A natural next step to take is to identify interesting special cases of PLC and to develop algorithms for these cases. For these algorithms to be practical, they would have to test containment of infinite languages of infinite sequences. It is unclear how to represent such languages by POMSETs so as to facilitate language-containment testing. Some candidate representations are suggested in Pratt's original paper and in Probst-Li [14].

We propose the following notation. Each language is represented by a deterministic Büchi automaton A and a collection $P_1, \ldots, P_k$ of POMSETs. Assume that each P_i exhibits no autoconcurrence. Each transition of A is labeled by a POMSET P_i. The language given by $(A, P_1, \ldots, P_k)$ consists of all sequences $w_{i_1}\ w_{i_2} \cdots$, where $P_{i_1}\ P_{i_2} \cdots$ is in $L(A)$ and w_{i_j} is in $L(P_{i_j})$.

Suppose that $(A, P_1, \ldots, P_k)$ and $(B, Q_1, \ldots, Q_k)$ are two such representations. Note that an implicit one-to-one correspondence between the two collections of POMSETs is given by their subscripts. Form an automaton B' by starting with B and substituting for each transition label Q_i the corresponding label P_i. Then a sufficient, but not necessary, condition for the language given by $(A, P_1, \ldots, P_k)$ to be contained in the language given by $(B, Q_1, \ldots, Q_k)$ is: $L(A) \subseteq L(B')$ and, for each i, $L(P_i) \subseteq L(Q_i)$.

This test can be performed in polynomial time. We hope to investigate its applicability in future work.

Finally, there is a large gap between the known upper and lower bounds for PLE: We know that the problem is at least as hard as graph isomorphism and that it is in Π_2^p. It would be interesting to determine its exact complexity.

References

[1] S. Aggarwal, R. P. Kurshan, and K. K. Sabnani. *Protocol Specification, Testing and Verification III* (1983), 19–34.

[2] M. C. Brown, E. M. Clarke, and O. Grumberg. *Inf. and Comput.* 81:13–31, 1989.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. LICS '90, 428–439.

[4] O. Coudert, C. Berthet, and J. C. Madre. Springer Verlag LNCS 407, 1989, 365–373.

[5] D. L. Dill. Springer Verlag LNCS 407, 1989, 197–212.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[7] P. Godefroid. DIMACS Series, vol. 3, 1991, 321–340.

[8] J. Kilian. Private communication.

[9] J. Köbler, U. Schöning, and J. Toran. *Acta Informatica* 26:363–379, 1989.

[10] R. P. Kurshan. Springer Verlag LNCS 430, 1990, 414-453.

[11] R. P. Kurshan. Springer Verlag LNCIS 103, 1987, 19–39.

[12] R. P. Kurshan and K. L. McMillan. PODC '89, 239–247.

[13] V. Pratt. *Intl. J. Parallel Programming* 15(1):33–71, 1986.

[14] D. Probst and H. Li. DIMACS Series, vol. 3, 1991, 15–24.

[15] A. P. Sistla, M. Y. Vardi, and P. Wolper. *Theor. Comput. Sci.* 49:217–237, 1987.

[16] H. J. Touati, R. K. Brayton, and R. P. Kurshan. Testing Language Containment for ω-Automata using BDD's, *Formal Methods in VLSI Design* (1991), ACM, to appear.

[17] L. Valiant. *Theor. Comput. Sci.* 8:189–201, 1979.

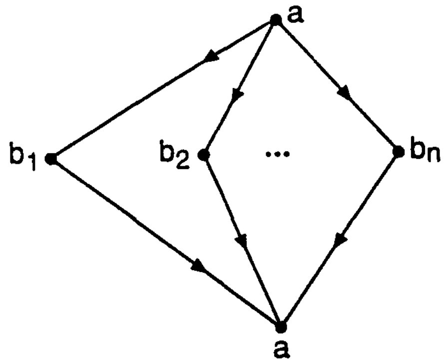


Figure 1

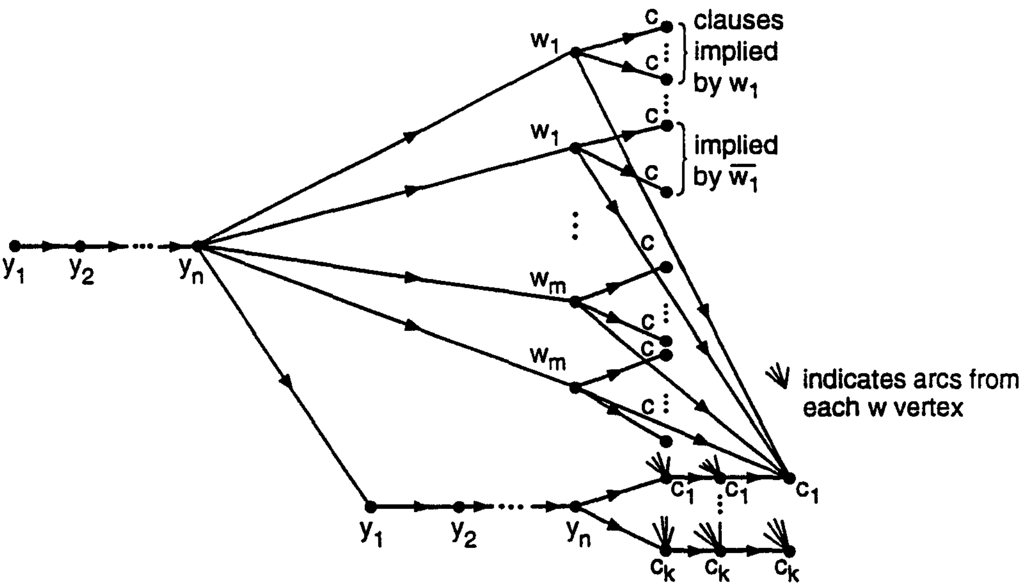


Figure 2

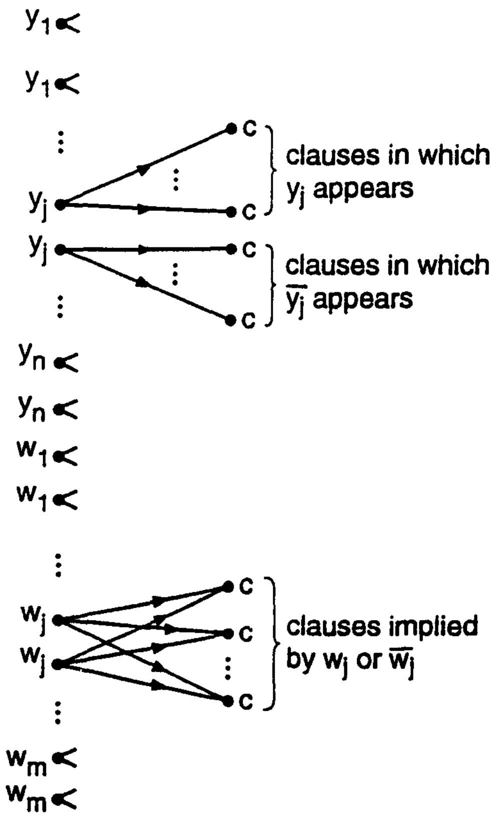
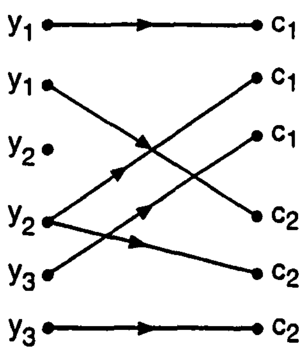


Figure 3

$c_1 = y_1 \vee \overline{y_2} \vee y_3, \qquad c_2 = \overline{y_1} \vee \overline{y_2} \vee \overline{y_3}$



$x = y_1 y_2 y_3 c_1 c_2 y_1 y_2 y_3 c_1 c_1 c_2 c_2$

Figure 4