Maximal Antichain Lattice Algorithms for Distributed Computations

Vijay K. Garg*

Parallel and Distributed Systems Lab,
Department of Electrical and Computer Engineering,
The University of Texas at Austin,
Austin, TX 78712
garg@ece.utexas.edu http://www.ece.utexas.edu/~garg

Abstract. The lattice of maximal antichains of a distributed computation is generally much smaller than its lattice of consistent global states. We show that a useful class of predicates can be detected on the lattice of maximal antichains instead of the lattice of consistent cuts obtaining significant (exponential for many cases) savings. We then propose new online and offline algorithms to construct and enumerate the lattice of maximal antichains. Previously known algorithm by Nourine and Raynoud [NR99,NR02] to construct the lattice takes $O(n^2m)$ time where n is the number of events in the computation, and m is the size of the lattice of maximal antichains. The algorithm by Jourdan, Rampon and Jard [JRJ94] takes $O((n + w^2)wm)$ time where w is the width of the computation. All these algorithms assume as input the lattice of maximal antichains prior to the arrival of a new event. We present a new online incremental algorithm, OLMA, that computes the newly added elements to the lattice without requiring the prior lattice. Since the lattice may be exponential in the size of the computation, we get a significant reduction in the space complexity. The OLMA algorithm takes $O(mw^2 \log w_L)$ time and $O(w_L w \log n)$ space where w_L is the width of the lattice of maximal antichains. The lower space complexity makes our algorithm applicable for online global predicate detection in a distributed system. For the purposes of analyzing offline traces, we also propose new enumeration algorithms to traverse the lattice.

1 Introduction

A distributed computation can be modeled as a partially ordered set (poset) of events based on the happened-before relation [Lam78]. Given any poset, there are three important distinct lattices associated with it: the lattice of consistent cuts (or ideals), the lattice of normal cuts, and the lattice of maximal antichains. The lattice of consistent cuts captures the notion of consistent global states in a distributed computation and has been discussed extensively in the distributed

* supported in part by the NSF Grants CNS-1115808, CNS-0718990, CNS-0509024, and Cullen Trust for Higher Education Endowed Professorship.

computing literature [Mat89,CM91,GM01]. The other lattices have not received as much attention. For a poset P, its completion by normal cuts is the smallest lattice that has P as its suborder [DP90]. Its applications to distributed computing are discussed in [Gar12]. In this paper, we discuss the lattice of maximal antichains with applications to global predicate detection.

For the set of events in a distributed computation ordered with happened-before relation, a subset of events forms an *antichain* if all events in the subset are pairwise concurrent. Informally, an antichain captures a possible set of events that could have occurred concurrently, and the events do not have any causal or happened-before relationship with each other. The lattice of all antichains is isomorphic to the lattice of all consistent cuts. An antichain A is *maximal* if there does not exist any event that can be added to the set without violating the antichain property. The *lattice of maximal antichains*, denoted by $L_{MA}(P)$ is the set of all maximal antichains under the order consistent with the order on the lattice of consistent cuts.

The lattice of maximal antichains captures all maximal sets of concurrent events and has applications in detection of global predicates because it is usually much smaller than the lattice of consistent cuts. In the extreme case, the lattice of consistent cuts may be exponentially bigger in size than the lattice of maximal antichains. We show in this paper that some global predicates can be detected on the lattice of maximal antichains instead of consistent cuts, thereby providing an exponential reduction in the complexity of detecting them. Fig. 1(i) shows a distributed computation with six events. For example, process P_1 executes a send event a, and then receives a message at event d. The message sent by P_1 is received by P_3 as event e. Fig. 1(ii) shows the computation with vector clocks. Fig. 2(i) shows the poset corresponding to the computation. Its lattices of consistent cuts and maximal antichains are shown in Fig. 2(ii), and (iii) respectively.

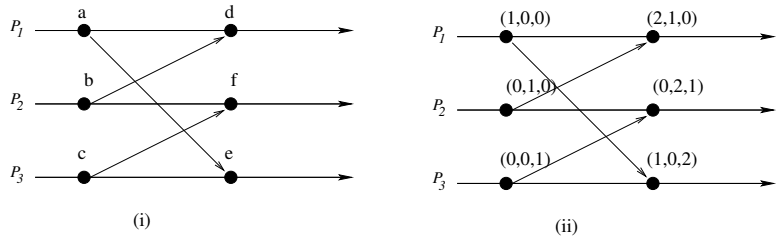

Fig. 1: (i) A computation (ii) Its equivalent representation in vector clocks

In this paper, we also discuss algorithms for computing L_{MA} for a distributed computation (or a trace of events) given as a finite poset P with implicit representation using vector clocks. Incremental algorithms assume that we are given a poset P and its lattice of maximal antichains L and we are required to construct L_{MA} of the poset P' corresponding to P extended with an element x. The algorithms by Jourdan, Rampon and Jard [JRJ94], and Lourine and Ray-

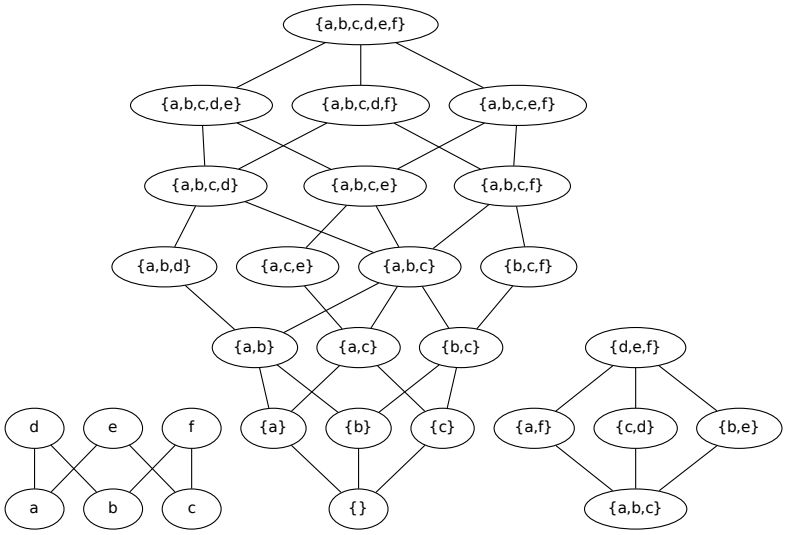

Fig. 2: (i) The original poset. (ii) Its lattice of ideals (consistent cuts) (iii) Its lattice of maximal antichains

naud [NR99,NR02] fall in this class. These algorithms store the entire lattice L_{MA} and have space complexity $O(mn \log n)$ where n is the size of the poset P, and m is the size of $L_{MA}(P)$. The algorithm by Jourdan, Rampon and Jard [JRJ94] has $O(w^3 m)$ time complexity and the algorithm by Lourine and Raynaud [NR99,NR02] has $O(mn)$ time complexity.

Our first algorithm called ILMA is a simple modification of the algorithm by Nourine and Raynoud [NR99,NR02] based on vector clocks. The algorithm requires $O(wm \log m)$ time and $O(wm \log n)$ space where w is the width of the poset P. The second algorithm called OLMA does not require lattice $L_{MA}(P)$ to compute elements of the new lattice. Let m_x be the size of the set of maximal antichains that include the element x. Then, the algorithm OLMA requires $O(w^2 m_x \log w_L)$ time and $O(w_L w \log n)$ space, where w_L is the width of the lattice of maximal antichains. Since w_L is much smaller than m, there is a significant reduction in the space complexity. The algorithm OLMA answers the open problem posed in [NR99]: "One open question is the enumeration of the family generated by a basis without computing the tree or the lattice with the same complexity."

Even though the OLMA algorithm has lower space complexity than ILMA, in the worst case the size of w_L can be exponential in the number of processes. If the goal is to not *construct*, but simply *enumerate* (or check for some global predicate) all the elements of L_{MA}, then we propose BFS and DFS based enumeration of lattice of maximal antichains. Earlier algorithms for enumeration of closed sets by Ganter [Gan84] use lexical enumeration. The BFS and DFS algorithms are more meaningful in the context of distributed systems. For ex-

ample, when searching for a maximal antichain satisfying a given predicate the programmer may be interested in the maximal antichain that appears first in the BFS enumeration. It is important to note that algorithms for BFS and DFS enumeration of lattices are different from the standard graph-based BFS and DFS enumeration because our algorithms cannot store the explicit graph corresponding to the lattice. Hence, the usual technique of marking the visited nodes is not applicable.

Table 1 summarizes the time and space complexity for construction and enumeration algorithms for the lattice of maximal antichains with the notation in Table 2.

Table 1: Lattice Construction of Maximal Antichains.

Incremental Algorithms	Time Complexity	Space Complexity
Jourdan et al.[JRJ94]	$O(w^3m)$	$O(mn \log n)$
Nourine and Raynaud[NR99,NR02]	$O(mn)$	$O(mn \log n)$
Algorithm ILMA [this paper]	$O(wm \log m)$	$O(mw \log n)$
Algorithm OLMA [this paper]	$O(m_x w^2 \log w_L))$	$O(w_L w \log n)$
Offline Algorithms		
Jourdan et al.[JRJ94]	$O((n+w^2)wm)$	$O(mn \log n)$
Nourine and Raynaud[NR99,NR02]	$O(mn^2)$	$O(mn \log n)$
Algorithm ILMA [this paper]	$O(nwm \log m)$	$O(mw \log n)$
BFS-MA [this paper]	$O(mw^2 \log m)$	$O(w_L w \log n)$
DFS-MA [this paper]	$O(mw^4)$	$O(nw \log n)$
Lexical by Ganter [Gan84]	$O(mn^3)$	$O(n \log n)$

Table 2: The notation used in the paper

Symbol	Definition	Symbol	Definition
n	size of the poset P	m	size of the maximal antichains lattice L
w	width of the poset P	m_x	number of strict ideals $\geq D(x)$ (Section 2)
w_L	width of the lattice L		

The paper is organized as follows. Section 2 gives the background definitions. Section 3 discusses the lattice of maximal antichains and some other lattices that are useful for incremental lattice construction. Section 4 discusses incremental and online construction of the lattice of maximal antichains. Section 5 discusses enumeration of the lattice of maximal antichains. We discuss distributed computing applications in Section 6.

2 Background: Posets with Implicit Representation

We assume that the reader is familiar with the basic concepts of posets and lattices [DP90]. A partially ordered set (or *poset*) is a pair $P = (X, \leq)$ where X is a set and $\leq$ is a reflexive, antisymmetric, and transitive binary relation on X. We write $x \leq y$ when $(x, y) \in P$. If either $x \leq y$ or $y \leq x$, we say that x and

y are *comparable*; otherwise, we say x and y are *incomparable* or *concurrent*. A subset $Y \subseteq X$ is called an *antichain (chain)*, if every distinct pair of points from Y is incomparable (comparable) in P. The *width* (*height*) of a poset is defined to be the size of a largest antichain (chain) in the poset.

Given a subset $Y \subseteq X$, the *meet* of Y, if it exists, is the greatest lower bound of Y and the *join* of Y is the least upper bound. In Fig. 2(i), the meet of the set $\{d, e\}$ is a. The meet of the set $\{b, c\}$ does not exist. An element is join-irreducible (meet-irreducible) if it cannot be expressed as join (meet) of other elements. In Fig. 2(i), the elements $\{a, b, c\}$ are join-irreducible but $\{d, e, f\}$ are not. A poset $P = (X, \leq)$ is a *lattice* if joins and meets exist for all finite subsets of X. The largest element of a lattice is called the *top* element.

Let P be a poset with a given chain partition of width w. In a distributed computation, P would be the set of events executed under the happened-before partial order. Each chain would correspond to a total order of events executed on a single process. In such a poset, every element e can be identified with a tuple (i, k) which represents the k^{th} event in the i^{th} process. In this paper, we keep the order relation of the poset implicit using vector clocks [Mat89,Fid89] as explained next. For $e \in P$, let $D[e]$, the *down-set* of e, be the elements of P that are less than or equal to P. The set $D[e]$ can equivalently be captured using a vector $e.V$ such that $e.V[i] = j$ iff there are exactly j elements on chain i that are less than or equal to e. It is easy to verify that $e \leq f$ iff $e.V \leq f.V$.

In this paper, we also use the set $D(e)$, the strict down-set of e, which contains all elements in the poset P that are strictly less than e. The reader should note the difference in the notation $D[e]$ and $D(e)$; the former is a down-set and the latter a strict down-set. The notation $D(e)$ can be extended to apply for sets as follows: $D(Y) = \cup_{e \in Y} D(e)$. We also use the dual notation for *up-sets*, $U(Y)$ and $U[Y]$.

A subset Q is an ideal (order ideal, or a consistent cut) of P if it satisfies the constraint that if f is in Q and e is less than or equal to f, then e is also in Q. For any element $e \in P$, $D[e]$ is always an ideal. In distributed computing, when a distributed computation is modeled as a poset of event, the order ideals are called *consistent cuts*, or *consistent global states* [CL85].

Any ideal Q of P can be represented using a vector $Q.V$ with the interpretation that $Q.V[i] = j$ iff exactly j elements of chain i are in Q. We use the set and the vector notation for an ideal interchangably. Given two ideals Q and R, their intersection (union) is simply the component-wise minimum (maximum) of the vectors for Q and R. The set of order ideals is closed under both union and intersection and therefore forms a lattice under the set containment order.

3 Maximal Antichain Lattice

We first define three different but isomorphic lattices: the lattice of maximal antichain ideals, the lattice of maximal antichains and the lattice of strict ideals. Besides giving an insight in the structure of the lattice of maximal antichains, these lattices have different closure properties making them useful in different

contexts. The lattice of strict ideals is closed under union and is used in our ILMA and OLMA algorithms. The lattice of maximal ideals is closed under intersection and is used in our DFS-MA algorithm.

Definition 1 (Maximal Antichain). *An antichain A is maximal in a poset $P = (X, \leq)$ if every element in $X - A$ is comparable to some element in A.*

In Fig. 3(i), the set $\{d, e\}$ is an antichain but not a maximal antichain because f is incomparable to both d and e. The set $\{d, e, f\}$ is a maximal antichain. It is easy to see that A is a maximal antichain iff $D(A) \cup U[A] = X$.

Definition 2 (Maximal Ideal). *An ideal Q of a poset (X, P) is a* maximal antichain ideal *(or, maximal ideal) if the set of its maximal elements, denoted by $maximal(Q)$, is a maximal antichain.*

The set of maximal ideals is closed under intersection but not union. In Fig.3(ii) the ideals $\{a, b, c, d\}$ and $\{a, b, c, e\}$ are maximal ideals, but their union $\{a, b, c, d, e\}$ is not a maximal ideal.

Definition 3 (Lattice of Maximal Ideals of a Poset). *For a given poset $P = (X, \leq)$, its lattice of maximal ideals is the poset formed with the set of all the maximal ideals of P under the set inclusion. Formally,*

$$L_{MA}(P) = (\{A \subseteq X : A \text{ is a maximal ideal of } P\}, \subseteq).$$

For the poset in Figure 2(a), the set of all maximal ideals is:
$\{\{a, b, c\}, \{a, b, c, d\}, \{a, b, c, e\}, \{a, b, c, f\}, \{a, b, c, d, e, f\}\}$.
The poset formed by these sets under the $\subseteq$ relation is shown in Figure 2(iii). This poset is a lattice with the meet as the intersection.

A lattice isomorphic to the lattice of maximal ideals is that of the maximal antichains.

Definition 4 (Lattice of Maximal Antichains of a Poset). *For a given poset $P = (X, \leq)$, its lattice of maximal antichains is the poset formed with the set of all the maximal antichains of P with the order $A \preceq B$ iff $D[A] \subseteq D[B]$.*

In Section 4 we discuss incremental algorithms for lattice construction. In these algorithms, we have the lattice $L_{MA}(P)$ for a poset P and our goal is to construct $L_{MA}(P \cup \{x\})$ where x is a new event that is not less than any event in P. It would be desirable if all the elements of $L_{MA}(P)$ continue to be elements of $L_{MA}(P')$. However, this is not the case for maximal antichains. An antichain that is maximal in P may not be maximal in $P \cup \{x\}$. For example, in Fig. 3, suppose that f arrives last. The set $\{d, e\}$ is a maximal antichain before the arrival of f, but not after. The algorithm by Jourdan, Rampon and Jard explicitly determines the maximal antichains that get changed when a new event arrives. In this paper, and also in [NR99], the problem is circumvented by building the lattice of strict ideals instead of the lattice of maximal antichains. If S is a strict ideal of P then it continues to be one on arrival of x so long as x is a maximal element of $P \cup \{x\}$. The lattice of strict ideals is isomorphic to the lattice of maximal antichains, but easier to implement via an incremental algorithm.

Definition 5 (Strict Ideal). *A set Y is a strict ideal of a poset $P = (X, \leq)$, if there exists an antichain $A \subseteq X$ such that $D(A) = Y$.*

Definition 6 (Lattice of Strict Ideals of a Poset). *For a given poset $P = (X, \leq)$, its lattice of strict ideals is the poset formed with the set of all the strict ideals of P with the $\subseteq$ order.*

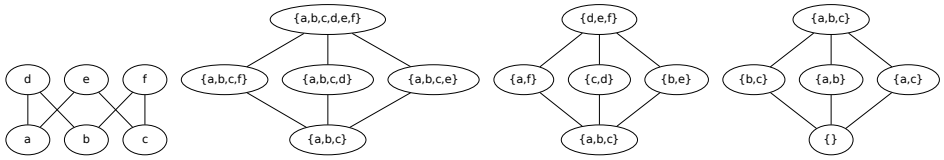

Fig. 3: (i) The original poset. (ii) Its lattice of maximal ideals (iii) Its lattice of maximal antichains (iv) Its lattice of strict ideals

Fig. 3 shows a poset with the three isomorphic lattices: the lattice of maximal ideals, the lattice of maximal antichains and the lattice of strict ideals. To go from the lattice of maximal ideals to the lattice of maximal antichains, a maximal ideal Q is mapped to the antichain $maximal(Q)$. Conversely, a maximal antichain A is mapped to the maximal ideal $D[A]$. To go from the lattice of antichains to the lattice of strict ideals, a maximal antichain A is mapped to the set $D(A)$. Conversely, a strict ideal Z is mapped to an antichain as the minimal elements in Z^c, the complement of Z. For example, when Z equals $\{b, c\}$, its complement is $\{a, d, e, f\}$. The set of the minimal elements of the set $\{a, d, e, f\}$ is $\{a, f\}$ which is the antichain corresponding to $\{b, c\}$. The correctness of this mapping is shown in the proof of correctness of ILMA algorithm (Theorem 1).

4 Incremental Algorithms to Construct Lattice of Maximal Antichains

In this section, we give two incremental algorithms, ILMA and OLMA. The ILMA algorithm makes it easier to understand the difference in the technique of previous algorithms and the OLMA algorithm.

4.1 The ILMA Algorithm

The ILMA algorithm is a modification of the algorithm given by Nourine and Raynaud [NR99], based on computing the lattice of strict ideals. There are two main differences. First, our algorithm does not use the lexicographic tree used in their algorithm. Another difference is that we have used the implicit representation of the poset and the lattice based on vector clocks making the algorithm more suitable for distributed systems.

The ILMA algorithm, shown in Fig. 4, is based on computing closure under union of strict ideals. It takes as input the poset P, an element x, and the lattice of maximal antichains of P. The poset and the lattice are assumed to be represented using vector clocks. It outputs the lattice of maximal antichains of $P' = P \cup \{x\}$. At step 1, we compute the vector clock corresponding to the set $D(x)$. The vector clock for x, V corresponds to $D[x]$. By removing x from the set $D[x]$, we get $D(x)$. The removal of x is accomplished by decrementing $S[i]$ in step 1. At step 2, we add S to L and make L closed under union.

Input: P: a finite poset as a list of vector clocks
L: lattice of maximal antichains as a balanced binary tree of vector clocks
x: new element
Output: L' := Lattice of maximal antichains of $P \cup \{x\}$ initially L

// **Step 1**: Compute the set $D(x)$
Let V be the vector clock for x on process P_i;
$S := V$; $S[i] := S[i] - 1$;
// **Step 2**:
if $S \notin L$ then
    $L' := L' \cup \{S\}$;
    forall vectors $W \in L$: if $max(W, S) \notin L$ then $L' := L' \cup max(W, S)$;

Fig. 4: The Algorithm ILMA for Construction of Lattice of Maximal Antichains

The above algorithm can also be used to compute the lattice of maximal antichains of any poset in an offline manner by repeatedly invoking the algorithm with elements of the poset in any total order consistent with the poset. To start the algorithm, the initial poset P would be a minimal element of P and the corresponding lattice L would be a singleton element corresponding to the empty strict downset.

We now show the correctness of the algorithm.

Theorem 1. *The lattice L' constructed by ILMA algorithm is isomorphic to the lattice of maximal antichains of P'.*

Proof. It is easy to verify that every vector in L' is a strict ideal I of the poset $P' = P \cup \{x\}$. By induction, we can assume that L contains all the strict ideals of P. Step (1) adds the strict ideal for $D(x)$ and step (2) takes its union with all existing strict ideals. Since *max* is an idempotent operation, it is sufficient to iterate over L once.

The bijection from L' to the set of maximal antichains is as follows. Let I be a strict ideal in L', i.e., there exists an antichain A such that $D(A) = I$. Let I^c denote the complement of I, and let B equal to the set of the minimal elements of I^c. Thus, $B = minimal(I^c)$. It can be shown that every strict ideal I gives a unique antichain B by this construction. To show that B is a maximal antichain

it is sufficient to show that $D(B) \cup U[B] = X$. This claim follows from the facts that $A \subseteq minimal(I^c)$, $D(A) = I$ and $U[minimal(I^c)] = I^c$.

The time complexity of ILMA is dominated by Step 2. Checking if the vector is in L requires $O(w \log m)$ time if L is kept as a balanced binary search tree of vector clocks. Thus, the time complexity of Step 2 is $O(wm \log m)$. By repeatedly invoking ILMA algorithm for a maximal element, we can construct DM completion of a poset with n elements in $O(nwm \log m)$ time. The algorithm by Jourdan, Rampon and Jard takes $O((n + w^2)wm)$ time, and the algorithm by Nourine and Raynaud takes $O(mn^2)$ time. The space complexity is dominated by storage requirements for L. With implicit representation, we have to store m elements where each element is stored as a vector of dimension w of coordinates each of size $O(\log n)$. Hence, the overall space complexity is $O(mw \log n)$.

4.2 The OLMA Algorithm

In the ILMA algorithm, we traverse the lattice L for every element x. It requires us to maintain the lattice L which may be exponentially bigger than poset P, making the algorithm impractical for distributed computations. We now show an online algorithm OLMA which does not require the lattice L but only uses the poset P. Let M be the set of new elements (strict ideals) generated due to x. The time complexity of OLMA is dependent on the size of M independent of the size of the lattice L.

The incremental online algorithm OLMA is shown in Fig. 5 At lines (1) and (2), we compute the vector S for the set $D(x)$. At line (3), we check if S is already in $L_{MA}(P)$. Note that we do not store the lattice $L_{MA}(P)$. The check at line (3) is done by checking if S is a strict ideal of P. If this is the case, we are done and M is an empty set. Otherwise, we need to enumerate all strict ideals that are reachable from S in the lattice $L_{MA}(P')$. We do so in lines (5)-(15) by traversing the strict ideals greater than or equal to S in the BFS order. The set $\mathcal{T}$ consists of strict ideals that have not been explored yet. At line (7), we remove the smallest strict ideal H and enumerate it at line (8). For global predicate detection applications, we would evaluate the global predicate on H at this step. To find the set of strict ideals that are reachable by taking union with one additional event e, we explore the next event e after the ideal H along every process. There are two cases to consider.

If $(D(e) \subseteq H)$, then the smallest event on process k that will generate new strict ideal by taking union with H is the successor of e on process k, $succ(e)$, if it exists. Since $D(succ(e))$ contains e which is not in H, we are guaranteed that $max(H, D(succ(e)))$ is strictly greater than H. It is also a strict ideal because it corresponds to union of two strict ideals H and $D(succ(e))$.

If $(D(e) \not\subseteq H)$, then the smallest event on process k that will generate new strict ideal by taking union with H is e. We add to $\mathcal{T}$ the strict ideal $max(H, D(e))$.

This method of BFS traversal is guaranteed to explore all strict ideals greater than or equal to S as shown by the next theorem.

```
    Input: a finite poset P, x maximal element in P' = P ∪ {x}
    Output: enumerate M such that L_MA(P') = L_MA(P) ∪ M

(1)     S := the vector clock for x on process P_i;
(2)     S[i] := S[i] − 1;
(3)     if S is not a strict ideal of P then
(4)     // BFS(S): Do Breadth-First-Search traversal of M
(5)         T := set of vectors initially {S};
(6)         while T is nonempty do
(7)             H := delete the smallest vector from T in the levelCompare order;
(8)             enumerate H;
(9)             foreach process k with next event e do
(10)                if (D(e) ⊆ H) then
(11)                    if (succ(e) exists then T := T ∪ {max(H, D(succ(e)))};
(12)                else
(13)                    T := T ∪ {max(H, D(e))};
(14)            endfor;
(15)        endwhile;
(16)    endif;

        int levelCompare(VectorClock a, VectorClock b)
(1)     if (a.sum() > b.sum()) return 1;
(2)     else if (a.sum() < b.sum()) return -1;
(3)     for (int i = 0; i < a.size(); i++)
(4)         if (a[i] > b.[i]) return 1;
(5)         if (a[i] < b[i]) return -1;
(6)     return 0;
```

Fig. 5: The Algorithm OLMA for Construction of Lattice of Strict Ideals

Theorem 2. *The Algorithm OLMA enumerates all $H \in M$ such that $L_{MA}(P') = L_{MA}(P) \cup M$.*

Proof. We first show that M contains only strict ideals greater than or equal to $D(x)$. Since we enumerate only the vectors deleted from $\mathcal{T}$ (at line 8) it is sufficient to show the claim for $\mathcal{T}$. The claim is initially true because $\mathcal{T}$ initially contains S which is a strict ideal equal to $D(x)$. For induction, we assume that H deleted at line 7 is also such a strict ideal. We add vectors only at lines 11 and 13. Since the set of strict ideals is closed under union, and both $D(e)$ and $D(succ(e))$ are strict ideals, we get that the vector added to $\mathcal{T}$ at lines 11 and 13 are also strict ideals. Both the ideals contain H due to the max operation, and hence they are greater than or equal to $D(x)$.

We now show that any strict ideal greater than or equal to $D(x)$ is added to $\mathcal{T}$ at some point in the algorithm. Let T be any strict ideal of P' greater than or equal to $D(x)$. We use induction on r, the size of $T - D(x)$. When r is zero, T equals $D(x)$, and is part of $\mathcal{T}$ due to line (5). Assume by induction that any strict ideal, T' with $|T' - D(x)| < r$ is added to $\mathcal{T}$. Let A be a minimal

set such that $D(A) = T$. Since $r > 0$, there exists $y \in A$ that is not in $D(x)$. Let $T' = D(A - \{y\})$. T' must be a proper subset of T; otherwise, A is not a minimal set such that $D(A) = T$. From induction hypothesis T' is added to $\mathcal{T}$ in the algorithm. Let z be the predecessor of y on the process that contains y. We have the following cases.
Case 1: $z \in T'$. Since $T' \in \mathcal{T}$ and $z \in T'$, y would be considered at line (9) when T' is explored. Since $T' = D(A - \{y\})$ and T' is a proper subset of T, we know that $D(y) \not\subseteq T'$. By line (13), T is added to $\mathcal{T}$.
Case 2: $z \notin T'$. There are two cases to consider.
Case 2.1: $D(z) \subseteq T'$. Since $T' \in \mathcal{T}$ and $D(z) \subseteq T'$, z would be considered at line (9) when T' is explored. Then by line (11), we add $max(T', D(y))$. Therefore, T is added to $\mathcal{T}$.
Case 2.2: $D(z) \not\subseteq T'$. When T' is explored, we add to $\mathcal{T}$, $T'' := max(T', D(z))$ due to line (13). Clearly, $D(z) \subseteq T''$ Since $T'' \in \mathcal{T}$ and $D(z) \subseteq T''$, when we explore T'', we add $max(T'', D(y))$ due to line (11). Therefore, we get that T is added to $\mathcal{T}$

Finally, we show that no vector is added to $\mathcal{T}$ again once it has been deleted. The function *levelCompare* provides a total order on all vectors. The vector H deleted is the smallest in $\mathcal{T}$. Any vector that we add due to H is strictly greater than H (either at line (11) or line (13)). Hence, once a vector has been deleted from $\mathcal{T}$ it can never be added back again.

We now analyze the time and space complexity of the algorithm OLMA. Lines (7) to (15) are executed for every strict ideal in M. Suppose that the number of strict ideals greater than or equal to $D(x)$ is m_x. The foreach loop at line (9) is executed w times. Computing max of two vectors at lines (11) and (13) take $O(w)$ time. Adding it to the set $\mathcal{T}$ takes $O(w \log w_L)$ time if $\mathcal{T}$ is maintained as a balanced binary tree of the vectors, where w_L is the maximum size of $\mathcal{T}$. Since $\mathcal{T}$ corresponds to BFS enumeration, it can also be viewed as the width of the lattice L in the worst case. Hence, the total time complexity for enumerating M is $O(m_x w^2 \log w_L)$. Recall that the ILMA algorithm traversed over the entire lattice when adding a new element resulting in $O(wm \log m)$ complexity for incremental construction.

We now compute the complexity of the OLMA algorithm to build the lattice for the entire poset. For simplicity, we bound m_x by m. Since the OLMA algorithm would be called n times, the time complexity is $O(nmw^2 \log w_L)$. The space complexity of the OLMA algorithm is $O(w_L w \log n)$ bits to store the set $\mathcal{T}$ where w_L is the maximum size that $\mathcal{T}$ will take during BFS enumeration.

5 Traversal Based Algorithms for Enumerating Lattice of Maximal Antichains

In some applications (such as global predicate detection discussed in Section 6), we may not be interested in storing L_{MA} but simply enumerating all its elements (or storing only those elements that satisfy given property). In this section, we

consider the problem of enumerating all the maximal antichains of a computation in an offline manner. In the OLMA algorithm, we enumerated all strict ideals greater than or equal to $D(x)$, when x arrives. We can use the OLMA algorithm in an offline manner as well. We simply use $BFS(\{\})$ instead of $BFS(D(x))$ which enumerates all the ideals. The time complexity is $O(mw^2 \log w_L)$ and the space complexity is $O(w_L w \log n)$. We call this algorithm BFS-MA.

We now show that the space complexity can be further reduced by using DFS (depth first search) enumeration of L_{MA}. The depth first search enumeration requires storage proportional to the height of L_{MA} which is at most n.

In previous section, we had used the lattice of strict ideals instead of lattice of maximal ideals. In this section, we use the lattice of maximal ideals because the lattice of maximal ideals is closed under intersection which allows us to find the smallest maximal ideal that contains a given set (at line (3) in 6).

One of the main difficulties is to ensure that we do not visit the same maximal antichain ideal twice because we do not store all the nodes of the lattice explicitly and hence cannot use the standard technique of marking a node visited during traversal. The solution we use is similar to that used for the lattice of ideals [AV01] and the lattice of normal cuts [Gar12]. Let $pred(H)$ be the set of all maximal ideals that are covered by H in the lattice. We use the total order *levelOrder* on the set $pred(H)$. We make a recursive call on H from the maximal ideal G iff G is the biggest maximal ideal in $pred(K)$ in the total order given by *levelOrder*. To find $pred(H)$, we first note that every maximal antichain of a poset P is also a maximal antichain of its dual P^d. Hence the lattice of maximal antichains of P is isomorphic to the lattice of maximal antichains of P^d. Traversing $L_{MA}(P)$ in the upward direction (in the Hasse diagram) is equivalent to traversing $L_{MA}(P^d)$ in the backward direction.

The algorithm for DFS enumeration is shown in Fig. 6. From any maximal ideal G, we explore all enabled events to find maximal ideals with at least one additional event. There are at most w enabled events and for each event it takes $O(w^2)$ time to compute the smallest maximal ideal K at line (3). At line (4) we check if K covers G using the characterization provided by Reuter [Reu91] as follows. A maximal ideal K covers the maximal ideal G in the lattice of maximal ideals iff $(K - G) \cup (U[Maximal(G)] - U[Maximal(K)])$ induces a complete height-one subposet of P with $(K - G)$ as the maximal elements and $(U[Maximal(G)] - U[Maximal(K)])$ as minimal element. This check can be performed in $O(w^2)$ time. In line (5), we traverse K using recursive call only if M equals G. Since there can be w predecessors for K and it takes $O(w^2)$ time to compute each predecessor; the total time complexity to determine whether K can be inserted is $O(w^3)$. Hence the overall time complexity of the algorithm is $O(mw^4)$.

The main space requirement of the DFS algorithm is the stack used for recursion. Every time the recursion level is increases, the size of the maximal ideal increases by at least 1. Hence, the maximum depth of the recursion is n, and the space requirement is $O(nw \log n)$ bits because we only need to store vectors of dimension w at each recursion level.

```
    Algorithm DFS-MaximalIdeals(G)
    Input: a finite poset P, starting state G
    Output: DFS Enumeration of all maximal ideals of P
(1)         output(G);
(2)         for each event e enabled in G do
(3)             K := smallest maximal ideal containing Q := G ∪ {e};
(4)             if K does not cover G then go to the next event;
(5)             M := get-Max-predecessor(K) ;
(6)             if M = G then
(7)                 DFS-MaximalIdeals(K);

    function VectorClock get-Max-predecessor(K) {
    //takes K as input vector and returns the maximal ideal that is biggest in the
levelCompare order

(1)         H = maximal ideal in P^d that has the same maximal antichain as K
(2)         // find the maximal predecessor using maximal ideals in the dual poset
(3)         for each event e enabled in the cut H in P^d do
(4)             temp := advance along event e in P^d from cut H;
(5)             // get the set of maximal ideals reachable in P^d
(6)             pred := smallest Maximal ideal containing temp that covers H
(7)         return the maximal ideal that corresponds to maxPred in P;
```

Fig. 6: Algorithm DFS-MA for DFS Enumeration of Maximal Ideals

6 Application of Lattice of Maximal Antichains

Global predicate detection problem has applications in distributed debugging, testing, and software fault-tolerance. The problem can be stated as follows. Given a distributed computation (either in an online fashion, or an offline fashion), and a global predicate B (a boolean function on the lattice of consistent global states), determine if there exists a consistent global state that satisfies B. The global predicate detection problem is NP-complete [CG98] even for the restricted case when the predicate B is a singular 2CNF formula of local predicates [MG01]. The key problem is that the lattice of consistent global states may be exponential in the size of the poset. Given the importance of the problem in software testing and monitoring of distributed systems, there is strong motivation to find classes of predicates for which the underlying space of consistent global states can be traversed efficiently. The class of linear predicates [CG98] and relational predicates [TG93,IG06] are two such classes. We now describe a class called *antichain* predicates which satisfies the property that they hold on the lattice L_{CGS} iff they hold on the lattice L_{MA}. We give several examples of predicates that occur in practice which belong to this class.

A global predicate B is an *antichain-consistent* predicate if its evaluation depends only on maximal events of a consistent global state and if it is true on a subset of processes, then presence of additional processes does not falsify the predicate. Formally,

Definition 7 (Antichain-Consistent Predicate). *A global predicate B defined on L_{CGS} is an* antichain-consistent predicate *if for all consistent global states G and H: $(maximal(G) \subseteq maximal(H)) \wedge B(G) \Rightarrow B(H)$.*

We now give several examples of antichain-consistent predicate.

- *Violation of mutual exclusion*: Consider the predicate, B, "there is more than one process in the critical section." The relevant critical section events for this predicate are entry to the critical section and exit from the critical section. B is true in a global state G iff $maximal(G)$ has more than one critical section event. Clearly, if B is true in G and $maximal(G)$ is contained in $maximal(H)$, then it is also true in H.
- *Violation of resource usage*: The predicate, B, "there are more than k concurrent activation of certain service," a slight generalization of the previous example, is also antichain-consistent.
- *Global Control Point*: The predicate, B, "Process P_1 is at line 35 and P_2 is at line 23 concurrently," is also antichain-consistent.

We can now show the following result.

Theorem 3. *There exists a consistent global state that satisfies an antichain-consistent predicate B iff there exists a maximal ideal that satisfies B.*

Proof. Let G be a consistent global state that satisfies B. If G is a maximal ideal, we are done. Otherwise, consider G^c. Since G is not a maximal ideal, there exists $y \in minimal(G^c)$ such that y is incomparable to all elements in $maximal(G)$. It is easy to see that $G_1 = G \cup \{y\}$ is also a consistent global state. Furthermore, $maximal(G) \subseteq maximal(G_1)$. Since B is antichain-consistent, it is also true in G_1. If G_1 is a maximal ideal, we are done. Otherwise, by repeating this procedure, we obtain H such that $maximal(G) \subseteq maximal(H)$, and H is a maximal ideal. From the definition of antichain-consistent, we get that $B(H)$.

The converse is obvious because every maximal ideal is also an ideal.

Hence, instead of constructing the lattice of ideals, we can use algorithms in Section 4 and Section 5 to detect an antichain-consistent global predicate resulting in significant reduction in time complexity.

References

[AV01] Sridhar Alagar and Subbarayan Venkatesan. Techniques to tackle state explosion in global predicate detection. *IEEE Transactions on Software Engineering*, 27(8):704–714, 2001.

[CG98] Craig M. Chase and Vijay K. Garg. Detection of global predicates: Techniques and their limitations. *Distributed Computing*, 11(4):191–201, 1998.

[CL85] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, February 1985.

[CM91] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proc. of the Workshop on Parallel and Distributed Debugging*, pages 163–173, Santa Cruz, CA, May 1991.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK, 1990.

[Fid89] C. J. Fidge. Partial orders for parallel debugging. *Proc. of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging*, 24(1):183–194, January 1989.

[Gan84] B. Ganter. Two basic algorithms in concept analysis. Technical Report 831, Techniche Hochschule, Darmstadt, 1984.

[Gar12] Vijay K. Garg. Lattice completion algorithms for distributed computations. In *Proc. of Principles of Distributed Systems - 16th International Conference, OPODIS 2012*, December 2012.

[GM01] V. K. Garg and N. Mittal. On slicing a distributed computation. In *21st Intnatl. Conf. on Distributed Computing Systems (ICDCS' 01)*, pages 322–329, Washington - Brussels - Tokyo, April 2001. IEEE.

[IG06] Selma Ikiz and Vijay K. Garg. Efficient incremental optimal chain partition of distributed program traces. In *ICDCS*, page 18. IEEE Computer Society, 2006.

[JRJ94] Guy-Vincent Jourdan, Jean-Xavier Rampon, and Claude Jard. Computing on-line the lattice of maximal antichains of posets. *Order*, 11:197–210, 1994.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. of the ACM*, 21(7):558–565, July 1978.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms: Proc. of the Intnatl. Workshop on Parallel and Distributed Algorithms*, pages 215–226. Elsevier Science Publishers B.V. (North-Holland), 1989.

[MG01] N. Mittal and V. K. Garg. On detecting global predicates in distributed computations. In *21st Intnatl. Conf. on Distributed Computing Systems (ICDCS' 01)*, pages 3–10, Washington - Brussels - Tokyo, April 2001. IEEE.

[NR99] Lhouari Nourine and Olivier Raynaud. A fast algorithm for building lattices. *Inf. Process. Lett.*, 71(5-6):199–204, 1999.

[NR02] Lhouari Nourine and Olivier Raynaud. A fast incremental algorithm for building lattices. *J. Exp. Theor. Artif. Intell.*, 14(2-3):217–227, 2002.

[Reu91] Klaus Reuter. The jump number and the lattice of maximal antichains. *Discrete Mathematics*, 88(23):289 – 307, 1991.

[TG93] A. I. Tomlinson and V. K. Garg. Detecting relational global predicates in distributed systems. In *Proc. of the Workshop on Parallel and Distributed Debugging*, pages 21–31, San Diego, CA, May 1993.