Lattice Completion Algorithms for Distributed Computations

Vijay K. Garg

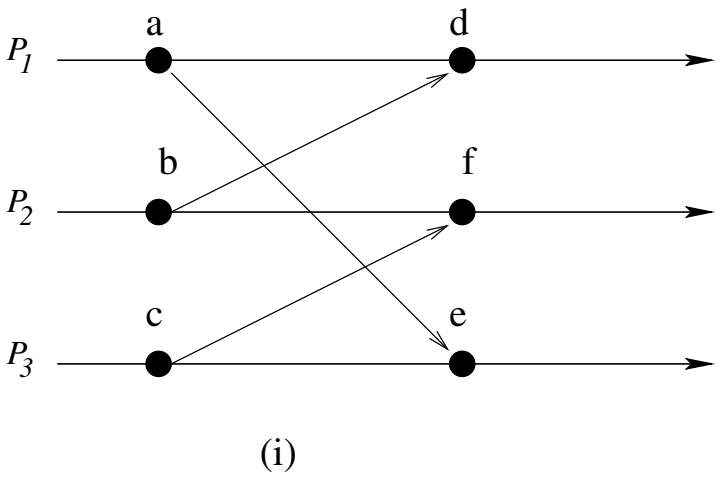
Parallel and Distributed Systems Lab,
Department of Electrical and Computer Engineering,
The University of Texas at Austin,
Austin, TX 78712
http://www.ece.utexas.edu/~garg

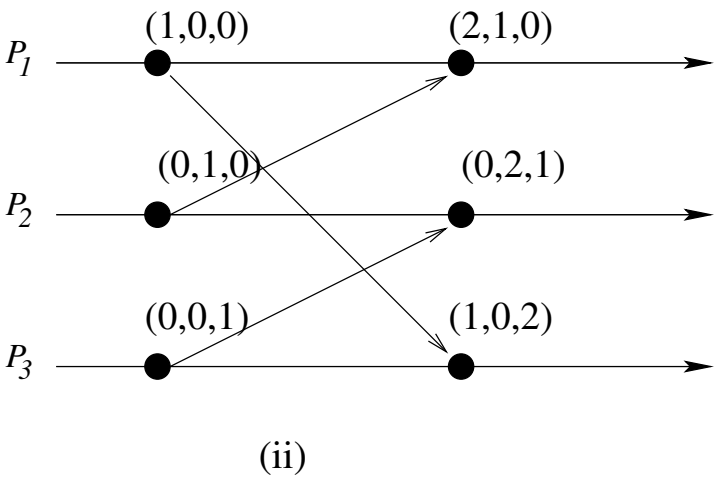
Outline of the Talk

- What is lattice completion?
- Motivation
- Normal Cuts
- Incremental Lattice Completion Algorithms
- Lattice Traversal Algorithms
- Conclusions and Future Work

Model of a Distributed Computation: Poset

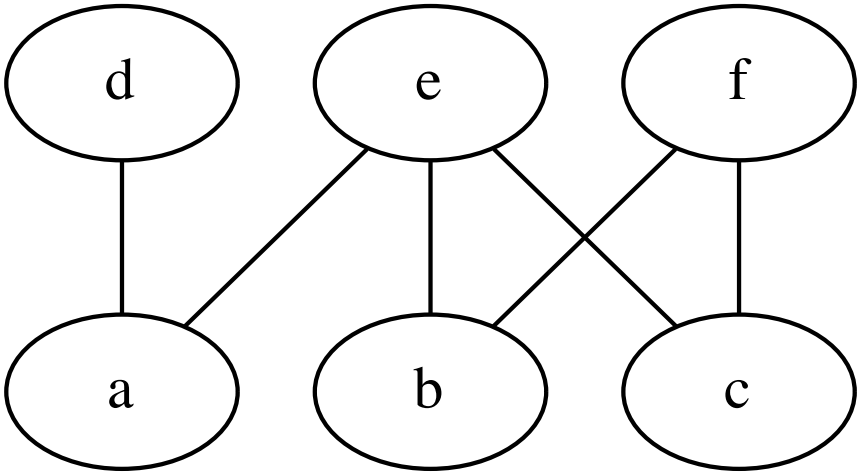
distributed computation = poset (partially ordered set)
$(E, \rightarrow)$ where
E = is the set of events, and
$\rightarrow$ is (happened-before) relation.





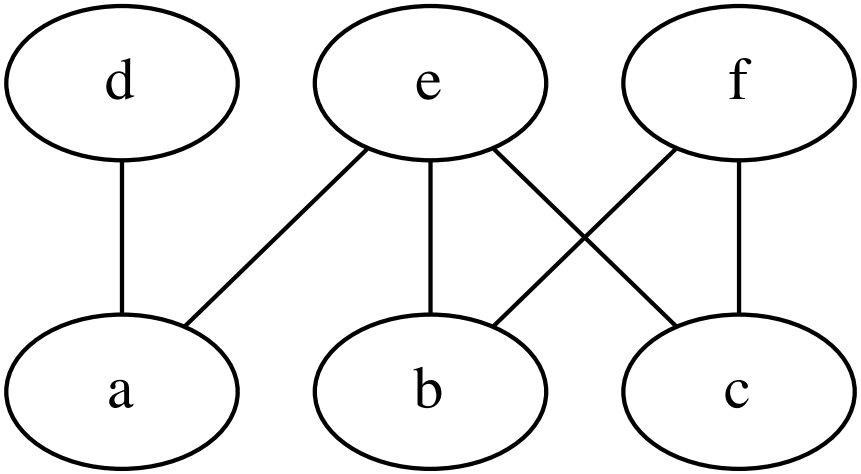
Events can be timestamped in an online fashion using Vector Clocks such that $e \rightarrow f \equiv V(e) < V(f)$.

Motivation: Computing Meets and Joins



- Meet (greatest lower bound) of a subset of events
  Interpretation: most recent common cause
  meet of $\{d, e\} = d \sqcap e = \{a\}$
  meet of $\{a, b\}$ does not exist
  meet of $\{e, f\}$
- Join (least upper bound) of a subset of events ($\sqcup$)
  Interpretation: least common consequence
- Lattice: a poset in which all finite subsets have meets and joins.

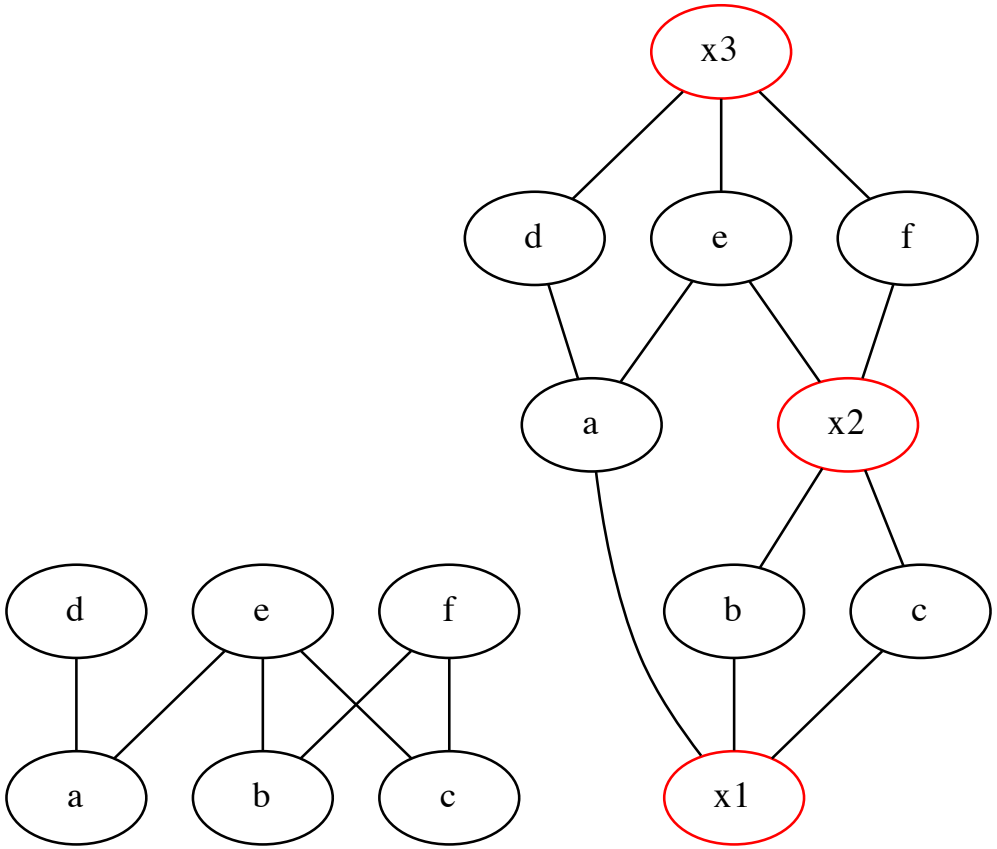
Motivation: Computing Meets and Joins



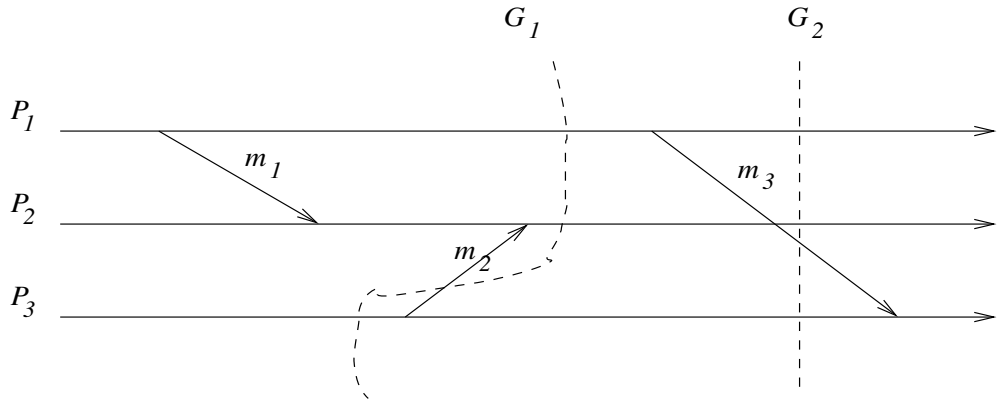
- Meet (greatest lower bound) of a subset of events
  Interpretation: most recent common cause
  meet of $\{d, e\} = d \sqcap e = \{a\}$
  meet of $\{a, b\}$ does not exist
  meet of $\{e, f\}$ does not exist
- Join (least upper bound) of a subset of events ($\sqcup$)
  Interpretation: least common consequence
- Lattice: a poset in which all finite subsets have meets and joins.

Smallest Lattice Completion

Problem Statement: Given a poset (a computation), find the smallest lattice that contains P as a subposet.



Consistent Cut of a Distributed System



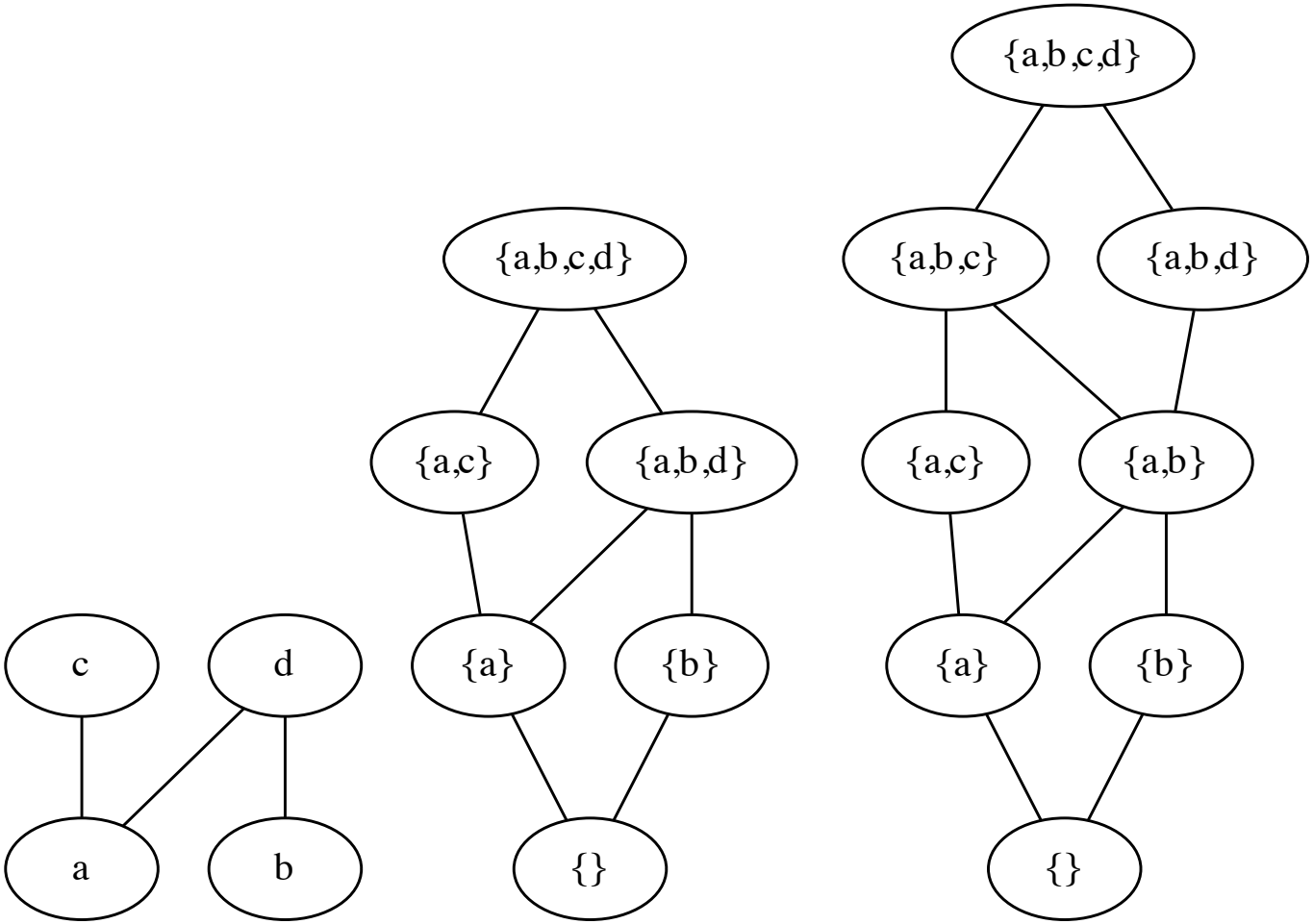
Consistent cut = set of events executed so far

A subset G of E is a consistent cut (consistent global state) if

$$\forall e, f \in E : (f \in G) \wedge (e \rightarrow f) \Rightarrow (e \in G)$$

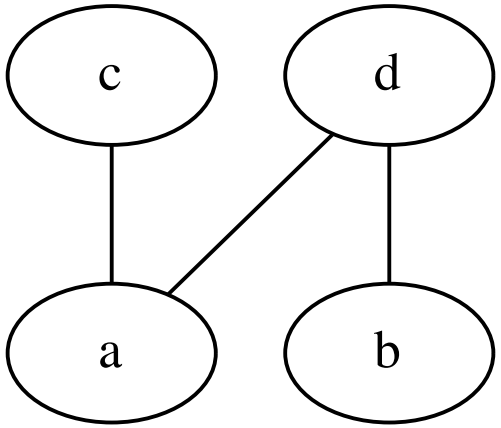
Motivation 2: Detecting Global Conditions

Problem: Given a global predicate find a consistent cut that satisfies the given predicate



(i) Poset (ii) Lattice Completion (Normal Cuts) (iii) Consistent Cuts

Normal Cuts of a Poset



Q^l: Lower Bounds of a set

Example: $\{c, d\}^l = \{a\}$

$\{d\}^l = \{a, b, d\}$

Q^u:Upper Bounds of a set

Example: $\{a, b, d\}^u = \{d\}$

$(\{a, b, d\}^u)^l = \{a, b, d\}$

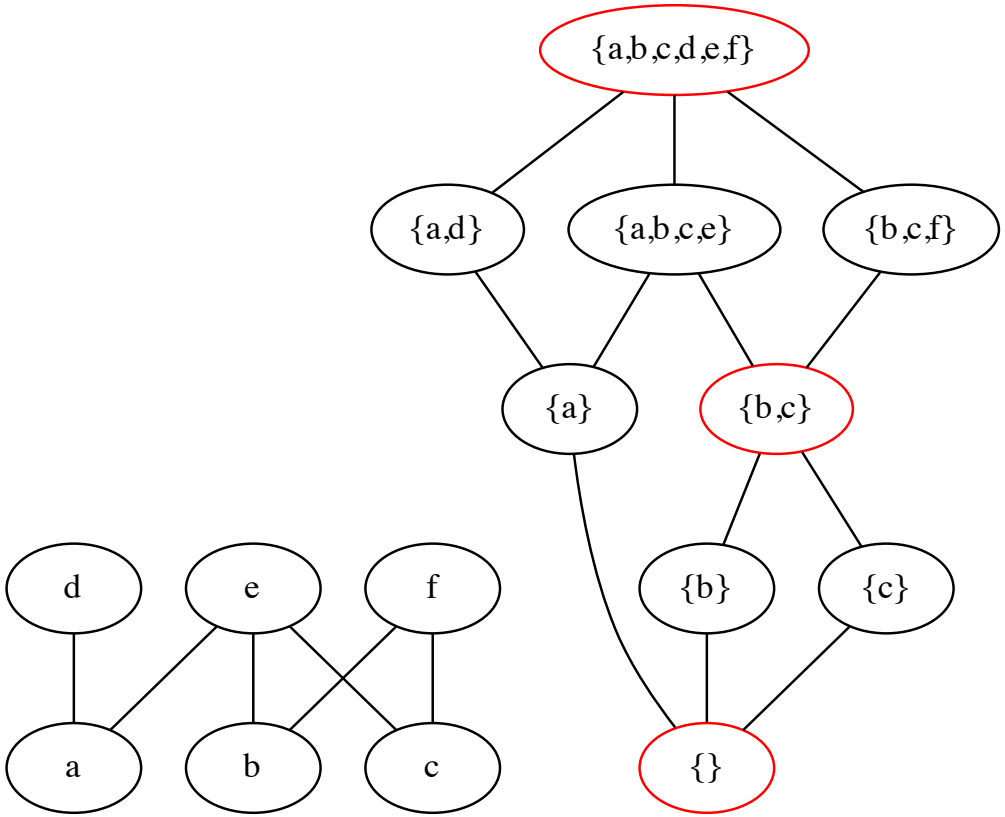
$\{a, b\}^{ul} = \{d\}^l = \{a, b, d\}$

A set $Q \subseteq P$ is a normal cut if $Q^{ul} = Q$.

Dedekind–MacNeille Completion of a Poset

Dedekind–MacNeille completion of $P = (X, \leq)$ is the poset formed with the set of all the normal cuts of P under the set inclusion.

$$DM(P) = (\{A \subseteq X : A^{ul} = A\}, \subseteq).$$



Outline of the Talk

- What is lattice completion?
- Motivation
- Normal Cuts
- Incremental Lattice Completion Algorithms
- Lattice Traversal Algorithms
- Conclusions and Future Work

Related Work: Incremental Algorithms

Elements of the poset arrive in a order preserving $\rightarrow$

Input: poset P, its DM-completion L, element x

Output: $L' :=$ DM-completion of $P \cup \{x\}$

Algorithm	Time Complexity	Space
Ganter and Kuznetsov 98	$O(mn^3)$	$O(mn \log n)$
Nourine and Raynaud 99, 02	$O(mn^2)$	$O(mn \log n)$
Algorithm IDML [this paper]	$O(rwm \log m)$	$O(mw \log n)$

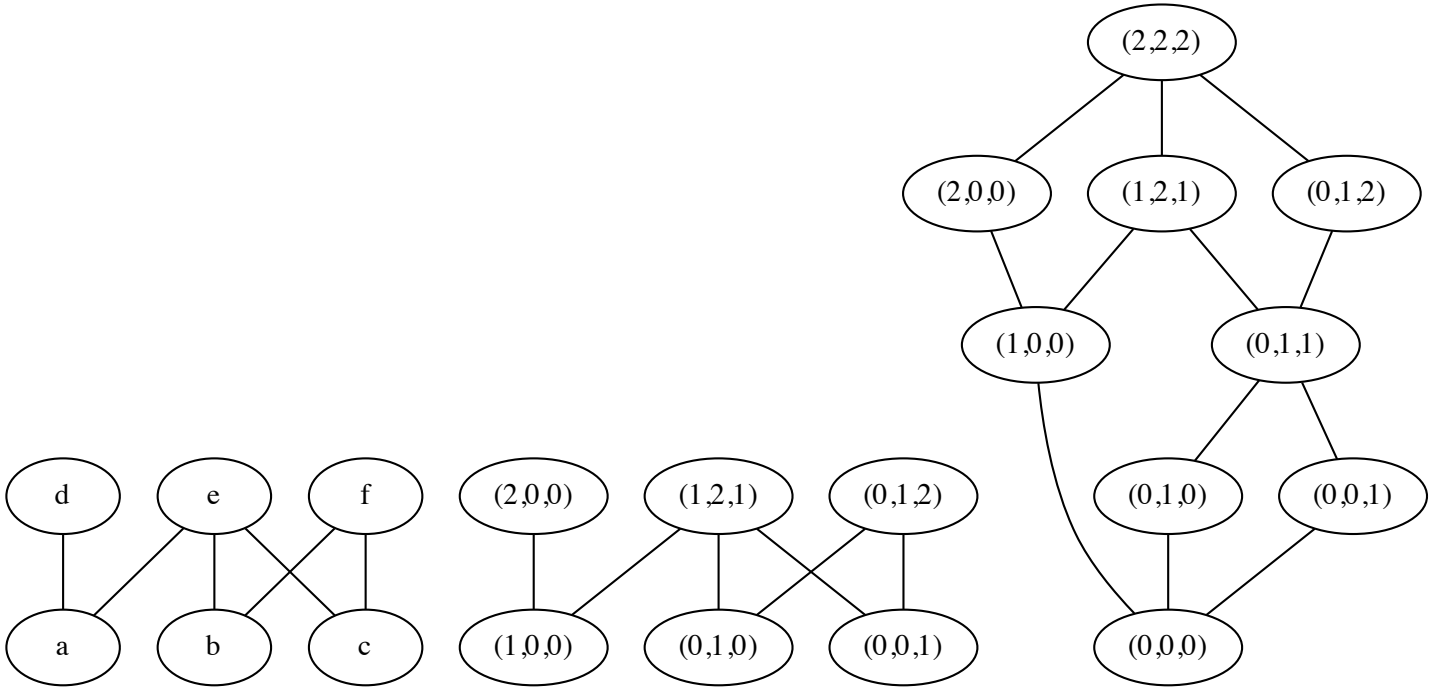
The parameters are:

n	size of the poset P	m	size of the lattice L of normal cuts
w	width of the poset P	r	elements with > 1 lower cover

Ideas in Our Incremental Algorithm

- Use vector clocks to represent cuts
- For any $x \in P = (X, \leq)$, let $D[x] = \{y \in X | y \leq x\}$
  $D[x]$ is always a normal cut.
- A finite poset is a lattice iff it has the top element and all meets are defined.
  $\text{join}(Q) = \text{meet}\ (Q^u)$
- Whenever a new element arrives, ensure that
  (1) there is a top element, and
  (2) all meets are defined.
- If an element x covers a single element, then it is sufficient to add $D[x]$.

Using Vector Clocks



Normal Cuts represented using vector clocks

Incremental Algorithm for DM-construction

Input: a nonempty finite poset P, its DM-completion L, element x
Output: $L' :=$ DM-completion of $P \cup \{x\}$

$D[x] :=$ the vector clock for x;
$Y := top(L)$;
$newTop := max(D[x], Y)$;
// **Step 1**: Ensure that L' has a top element

// **Step 2**: Ensure that $D[x]$ is in L'

// **Step 3**: Ensure that all meets are defined

Incremental Algorithm for DM-construction

Input: a nonempty finite poset P, its DM-completion L, element x
Output: $L' :=$ DM-completion of $P \cup \{x\}$

$D[x] :=$ the vector clock for x;
$Y := top(L)$;
$newTop := max(D[x], Y)$;
// **Step 1**: Ensure that L' has a top element
  if $Y \in P$ then $L' := L \cup \{newTop\}$;
  else $L' := (L - Y) \cup \{newTop\}$;
// **Step 2**: Ensure that $D[x]$ is in L'

// **Step 3**: Ensure that all meets are defined

Incremental Algorithm for DM-construction

Input: a nonempty finite poset P, its DM-completion L, element x
Output: $L' :=$ DM-completion of $P \cup \{x\}$

$D[x] :=$ the vector clock for x;
$Y := top(L)$;
$newTop := max(D[x], Y)$;
// **Step 1**: Ensure that L' has a top element

// **Step 2**: Ensure that $D[x]$ is in L'
$L' := L' \cup \{D[x]\}$;
// **Step 3**: Ensure that all meets are defined

Incremental Algorithm for DM-construction

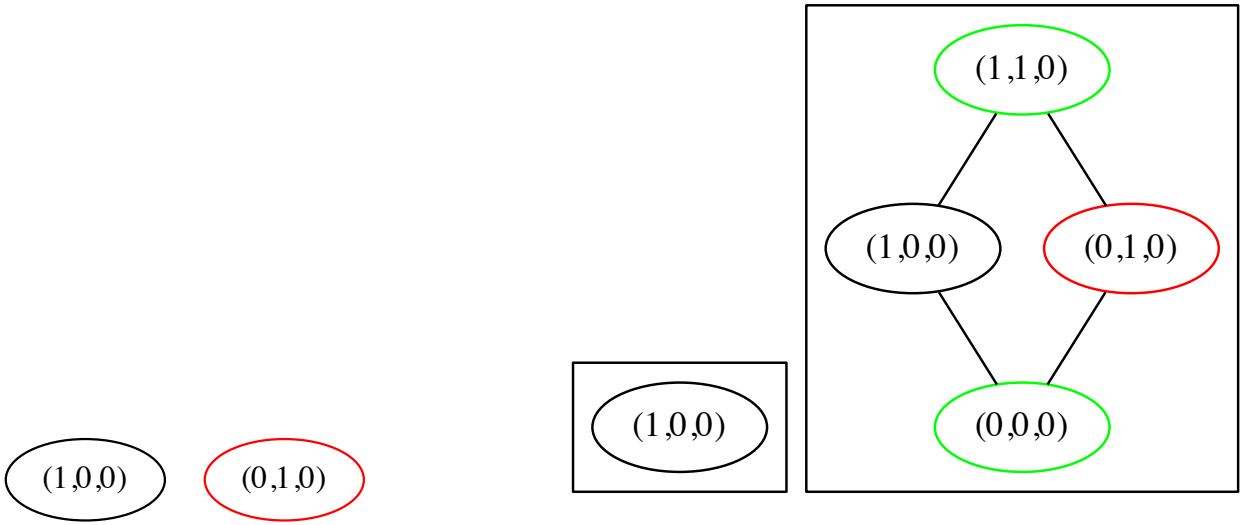
Input: a nonempty finite poset P, its DM-completion L, element x
Output: $L' :=$ DM-completion of $P \cup \{x\}$

$D[x] :=$ the vector clock for x;
$Y := top(L)$;
$newTop := max(D[x], Y)$;
// **Step 1**: Ensure that L' has a top element

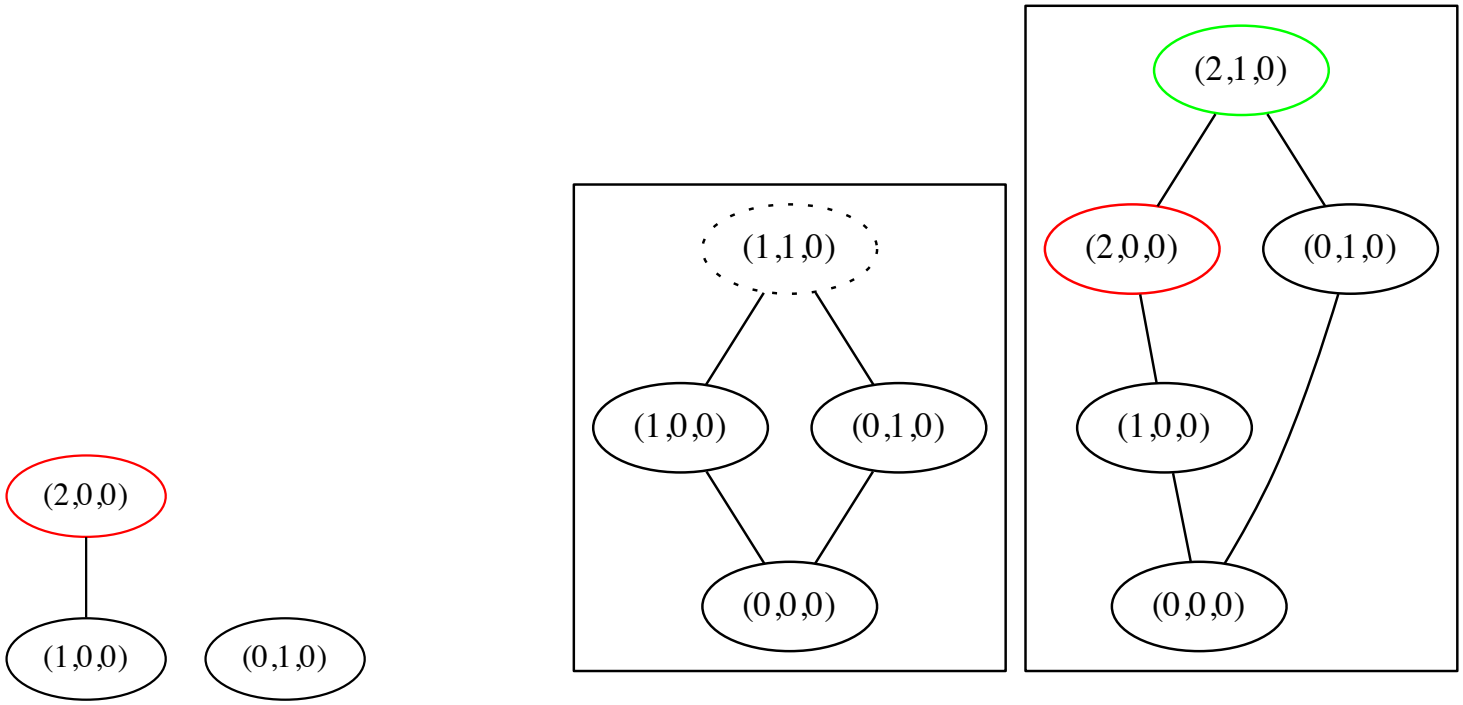
// **Step 2**: Ensure that $D[x]$ is in L'

// **Step 3**: Ensure that all meets are defined
if x covers more than one element in P then
for all normal cuts $W \in L$ do
if $min(W, D[x]) \notin L'$ then $L' := L' \cup min(W, D[x])$;

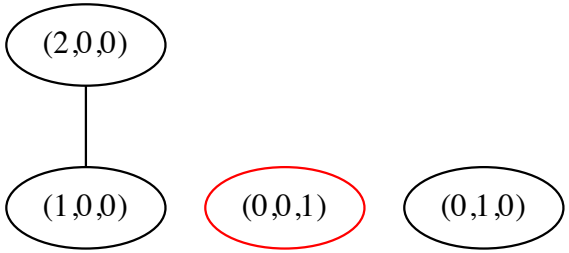
IDML Example

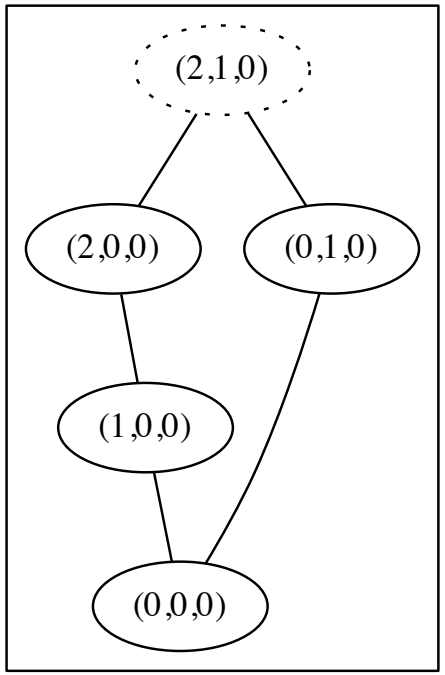


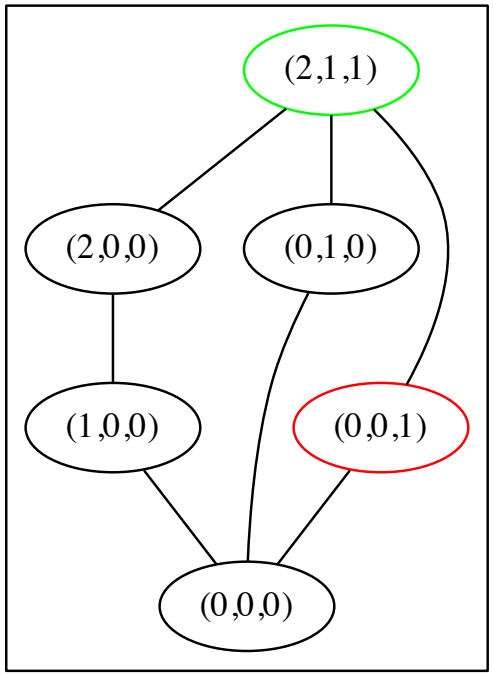
IDML Example



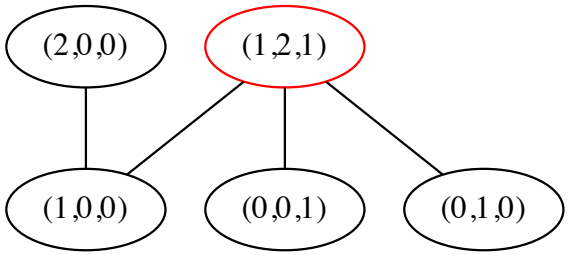
IDML Example

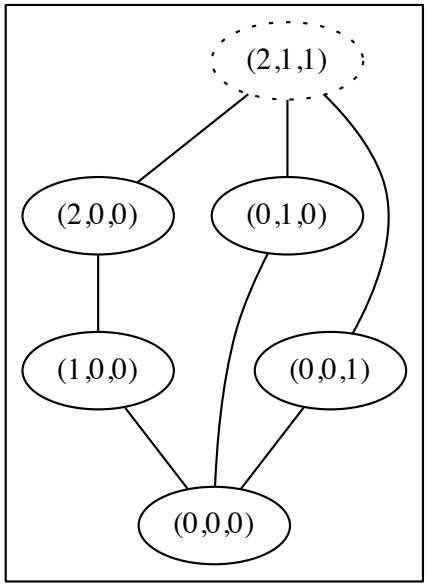


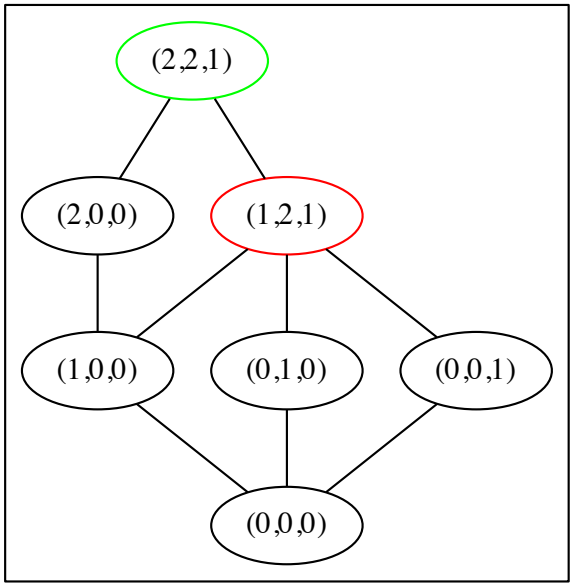




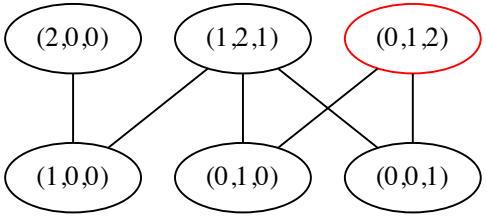
IDML Example

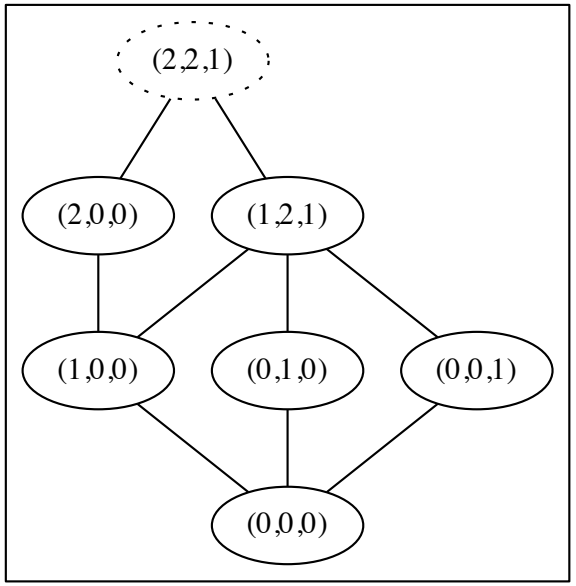


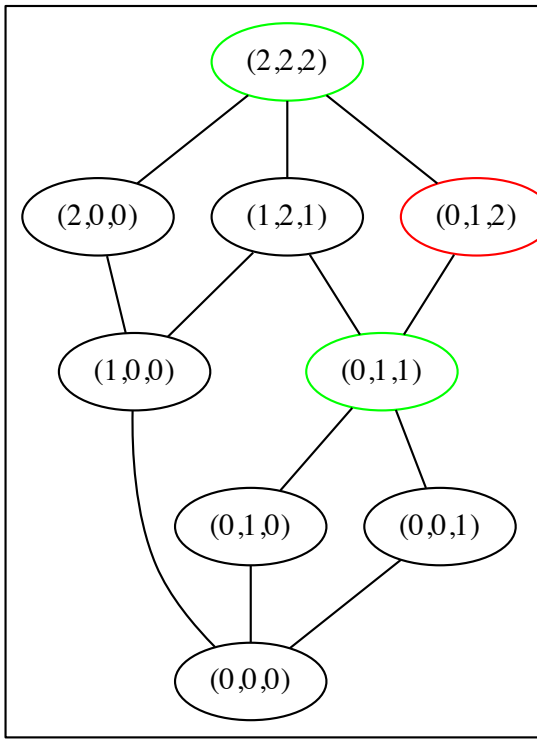




IDML Example







Outline of the Talk

- What is lattice completion?
- Motivation
- Normal Cuts
- Incremental Lattice Completion Algorithms
- Lattice Traversal Algorithms
- Conclusions and Future Work

Related Work: Enumeration Algorithms

Input: a nonempty finite poset P
Output: enumerate all elements of $L :=$ DM-completion of P

Algorithm	Time	Space
Lexical (by Ganter84)	$O(mn^3)$	$O(n \log n)$
BFS [this paper]	$O(mw^2(w + \log w_L))$	$O(w_L w \log n)$
DFS [this paper]	$O(mw^3)$	$O(h_L w \log n)$

The parameters are:

n	size of the poset P	m	size of the lattice L of normal cuts
w	width of the poset P	w_L	width of the lattice L
h_L	height of the lattice L		

Enumeration Algorithms

- Enumerate all normal cuts without storing them
- Traversal can be done in BFS, DFS, or lexical
- Challenge: Not allowed to store the lattice (graph)

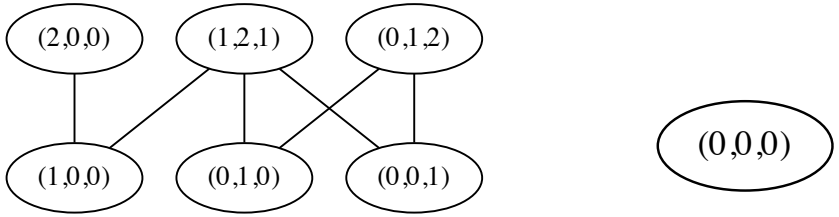
BFS Traversal Algorithm

Input: a finite poset P
Output: BFS Enumeration of $DM(P)$
$G :=$ bottom element ;
$\mathcal{S} :=$ Ordered Set of VectorClocks initially $\{G\}$;
(1) **while** ($\mathcal{S}$ is notEmpty)
(2) $\quad H :=$ remove the smallest element from $\mathcal{S}$;
(3) $\quad$ output(H);
(4) $\quad$ **foreach** event e enabled in H do;
(5) $\quad\quad K :=$ the smallest normal cut containing $Q := H \cup \{e\}$;
(6) $\quad\quad$ if K is not in $\mathcal{S}$, then add K to $\mathcal{S}$;

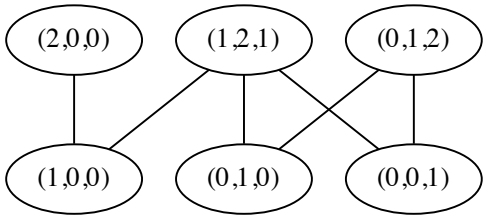
BFS: Maintaining $\mathcal{S}$

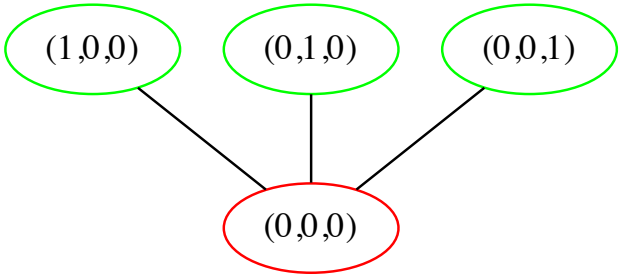
- Keeping $\mathcal{S}$ as a queue does not work
- Need the guarantee that if K has been enumerated then $K \in \mathcal{S}$
- The order on $\mathcal{S}$ must preserve the order defined by the size of the cut

BFS Example

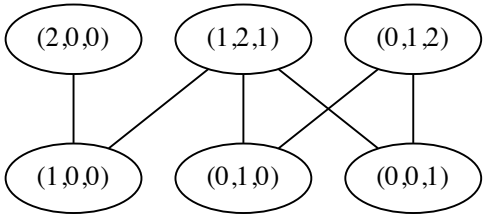


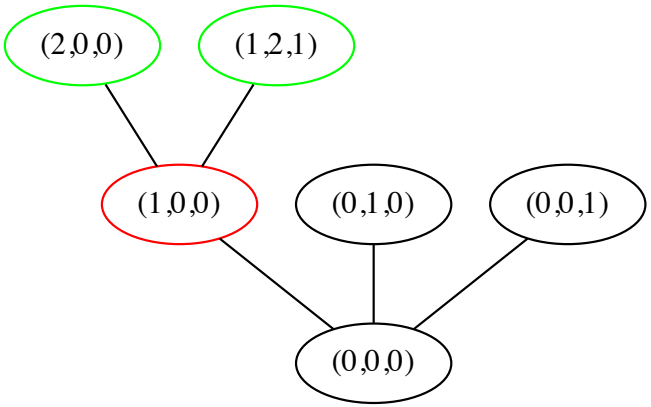
BFS Example



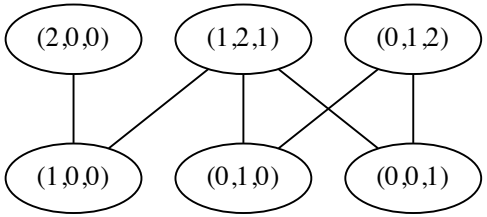


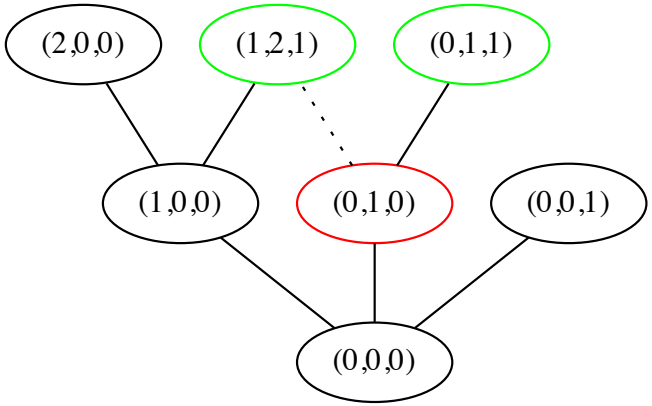
BFS Example





BFS Example





Depth First Search Enumeration of Normal Cuts

- BFS requires space $O(width(DM(P))$
- Width of the lattice of normal cuts can be large
- DFS requires space $O(height(DM(P))$
- $height(DM(P)) \leq |P|$

Depth First Search Enumeration of Normal Cuts

Input: a finite poset P, starting state G

Output: DFS Enumeration of elements of DM-completion of P

```
output(G);
foreach event e enabled in G do
    K := smallest normal cut containing Q := G ∪ {e};
    if K has not been visited before then
        DFS-NormalCuts(K);
```

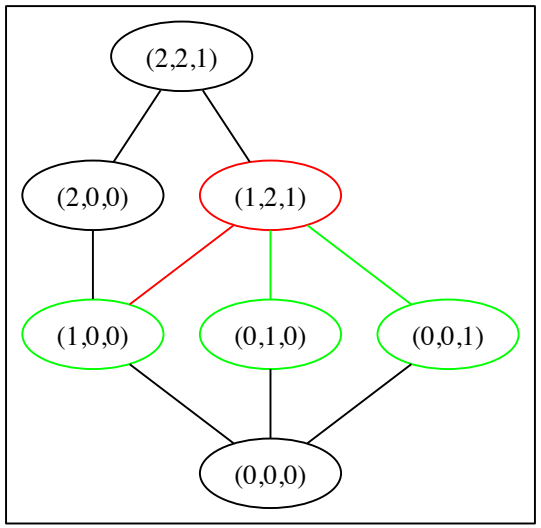
How to avoid revisiting cuts?

Visit a state only from the maximum predecessor.

(4) $M := \textit{get-Max-predecessor}(K)$;

(5) if $M = G$ then

(6) DFS-NormalCuts(K);



Application to Global Predicate Detection

Every one knows predicate B in the consistent cut G, if there exists a consistent cut H such that

- H satisfies B and
- for every process i there exists an event e in $G[i]$ such that all events in H happened before e.

Everyone Knows B can be detected in the lattice of normal cuts instead of consistent cuts.

Conclusions and Future Work

- An Incremental Algorithm IDML to generate completion lattice
- BFS and DFS enumeration of normal cuts. DFS has $O(mw^3)$ time complexity.
- Applications to global predicate detection

Question: Is there a space-efficient algorithm with time complexity $O(mw \log n)$?

Thank You

Any questions?

Complexity of the Incremental Algorithm

Input: a nonempty finite poset P, its DM-completion L, element x
Output: $L' :=$ DM-completion of $P \cup \{x\}$

// **Step 3**: Ensure that all meets are defined
if x covers more than one element in P then
for all normal cuts $W \in L$ do
if $min(W, D[x]) \notin L'$ then $L' := L' \cup min(W, D[x])$;

Time dominated by step 3: Use balanced binary trees to store L
Time: $O(w \log m)$ to check if a vector in L
Due to for loop, we get $O(mw \log m)$ for elements that cover more than one element
If x does not cover more than one element, then $O(w \log m)$
Building the lattice for the entire poset:
$O(rmw \log m + (n - r)w \log m)$. $= O(rmw \log m)$ for $r > 1$.

Using Closures

- Given any subset Q of the poset:
  Q^{ul} = the smallest normal cut that contains Q
- Computing Q^{ul} is a closure operator, i.e.,
  1. $Q \subseteq Q^{ul}$ (it is extensive)
  2. $Q_1 \subseteq Q_2 \Rightarrow Q_1^{ul} \subseteq Q_2^{ul}$ (it is monotone)
  3. $(Q^{ul})^{ul} = Q^{ul}$ (it is idempotent)
- lattices $\equiv$ family of closed sets $\equiv$ topped intersection-closed family of sets.

How to get the maximum predecessor?

Expand K in the dual poset. Choose the biggest successor.

function VectorClock *get-Max-predecessor*(K) {

//returns K's maximum predecessor normal cut

(1) H = MinimalUpperBounds(K); // $H := K^u$

(2) **foreach** event f enabled in the cut H in P^d do

(3) $temp_f := H - \{f\}$; // advance on f in P^d

(4) // get the set of lower bounds on $temp_f$

(5) $pred$:= MaximalLowerBounds($temp_f$) using H^l;

(6) if (levelCompare($pred$, $maxPred$) = 1) $maxPred = pred$;

(7) return $maxPred$;

Application to Global Predicate Detection

every one knows the predicate B in the consistent cut G, if there exists a consistent cut H such that

- H satisfies B and
- for every process i there exists an event e in $G[i]$ such that all events in H happened before e.

Theorem

Let B be any global predicate and G be a consistent cut such that $E(B, G)$. Then, there exists a normal cut N such that $E(B, N^u)$ and N^u is less than or equal to G.

The least consistent cut in which Everyone Knows B corresponds to N^u for a normal cut N.