A Lattice-Theoretic Approach to Monitoring Distributed Computations

Vijay K. Garg    Neeraj Mittal

Parallel and Distributed Systems Lab,
Department of Electrical and Computer Engineering,
The University of Texas at Austin,

Advanced Networking and Dependable Systems Laboratory
Computer Science Department
The University of Texas at Dallas
Motivation

Debugging and Testing Distributed Programs:
- Global Breakpoints: stop the program when $x_1 + x_2 > x_3$
- Traces need to be analyzed to locate bugs.

Software Fault-Tolerance:
- Distributed programs are prone to errors.
  - Concurrency, nondeterminism, process and channel failures
- Software faults are dominant reasons for system outages
- Need to take corrective action when the current computation violates a safety invariant

Software Quality Assurance:
- Can I trust the results of the computation? Does it satisfy all required properties?
What is a Distributed Computation?

- **Distributed Program:** a computer program that runs on a distributed system

- **Distributed Computation:** A single execution of a distributed program

- **Assumptions:**
  - No shared memory,
  - No shared clock,
  - Asynchrony in communication
Modeling a Distributed Computation

A computation is $(E, \rightarrow)$ where $E$ is the set of events and $\rightarrow$ (happened-before) is the smallest relation that includes:

- $e$ occurred before $f$ in the same process implies $e \rightarrow f$.
- $e$ is a send event and $f$ the corresponding receive implies $e \rightarrow f$.
- if there exists $g$ such that $e \rightarrow g$ and $g \rightarrow f$, then $e \rightarrow f$.

[Diagram showing a sequence of events $e_1, e_2, e_3, e_4, e_5$ and corresponding $f_1, f_2, f_3, f_4, f_5$ with arrows indicating the happened-before relation.]

[Lamport 78] RV’14 Tutorial (Garg and Mittal) Monitoring Distributed Computations
Modeling a computation as a Poset

\[(E, \to)\] is an irreflexive poset (\(\to\) is an irreflexive and transitive binary relation on \(E\))

Can we exploit the theory of ordered sets?

- join/meet of elements, width of a poset, dimension of a poset, order ideals

Example: Order ideal of a poset corresponds to a consistent global state. The set of all order ideals form a distributive lattice under set containment relation.

Can we exploit the theory of distributive lattices for analyzing consistent global states?

- representing sublattices, lattice congruences
Talk Outline

1 Motivation

2 Background: Posets and Lattices
   - Cooper and Marzullo’s Algorithm
   - Alagar and Venkatesan’s Algorithm
   - Lexical Enumeration of Consistent Global States

3 Global Predicate Detection Problem
   - Linear Predicates
   - Relational Predicates

4 Predicate Detection for Special Classes

5 Slicing

6 Basis Temporal Logic
   - Syntax and Semantics
   - Semiregular Predicates
   - Algorithm to detect BTL
A poset (partially ordered set) is a tuple \((X, \leq)\) where \(X\) is any set and \(\leq\) is a binary relation on \(X\) with the following properties:

- reflexive,
- antisymmetric and
- transitive

\((X, <)\) is an irreflexive poset when \(<\) is irreflexive and transitive.

Examples:

1. \((\mathbb{N}, <)\):
   - set of natural numbers under usual less than relation

2. \((\mathbb{N}^k, <)\):
   - set of \(k\)-dimensional vectors under component-wise comparison
     - \((2, 3, 0) < (3, 3, 1)\)
     - \((2, 3, 0) \not< (1, 4, 2)\)

3. \((E, \rightarrow)\):
   - set of events of a distributed computation under the happened-before relation
Background: Poset Terminology

- $x \parallel y$ (\textit{x incomparable with y}): $\neg (x < y) \land \neg (y < x)$
- \textbf{chain}: $Y \subseteq X$ is a chain if every distinct pair of elements from $Y$ is comparable
- \textbf{antichain}: $Y \subseteq X$ is an antichain if every distinct pair of elements from $Y$ is incomparable
- \textbf{height of a poset}: size of the longest chain in the poset
- \textbf{width of a poset}: size of the longest antichain in the poset
- \textbf{width antichain}: antichains of size equal to the width
Background: Lattices

For any $z \in X$, $z$ is the join of $x$ and $y$, i.e., $z = x \sqcup y$ iff

- $x \leq z$ and $y \leq z$
- $\forall z' \in X, (x \leq z' \land y \leq z') \Rightarrow z \leq z'$.

The meet of two elements $z = x \sqcap y$ is defined dually.

A poset $(X, \leq)$ is a lattice iff it is closed under meets and joins.

$\forall x, y \in X, x \sqcup y \in X$ and $x \sqcap y \in X$. 
Background: Sublattices

Which subsets form sublattices?

(i) (ii)

(iii) (iv)

RV’14 Tutorial (Garg and Mittal)
Background: Distributive Lattices

A lattice \((L, \leq)\) is a distributive lattice iff
\[
\forall x, y, z \in L : x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z).
\]

Fact

- A lattice is distributive iff it does not have a pentagon or a diamond as a sublattice.
Let \((X, <)\) be any poset. A subset \(Y \subseteq X\) an order ideal (or a downset) if 

\[ z \in Y \land y < z \Rightarrow y \in Y. \]

Are these order ideals?

\(Y_1 = \{a_1, b_1\}\)
\(Y_2 = \{a_1, a_3, b_1\}\)
\(Y_3 = \{}\)
\(Y_4 = X\)

\(Y \subseteq X\) an order filter if 

\[ z \in Y \land z < y \Rightarrow y \in Y \]
Example: Order Ideals

Consistent Global State (CGS) of a Distributed System

Consistent global state = subset of events executed so far. A subset $G$ of $E$ is a consistent global state (also called a consistent cut) if

$$\forall e, f \in E : (f \in G) \land (e \rightarrow f) \Rightarrow (e \in G)$$
Background: Lattice of order ideals

**Theorem**

The set of all order ideals of any poset forms a distributive lattice under the set containment relation.

The set of ideals forms a lattice

- if $X$ and $Y$ are ideals then so are $X \cap Y$ and $X \cup Y$
  
  meet $\rightarrow$ intersection $\rightarrow$ join

  $\rightarrow$ union

$Y_1 = \{a_1, a_3, b_1\}$

$Y_2 = \{a_1, a_2, b_2\}$

$Y_1 \cup Y_2 = \{a_1, a_2, a_3, b_1, b_2\}$

$Y_1 \cap Y_2 = \{a_1\}$
Ideal Lattice

The lattice of ideals is distributive

- union distributes over intersection

which of the following graphs are possible CGS lattices?

Corollary: The set of all CGS of a computation forms a distributive lattice.
Modeling using States vs Events

One can model a computation using states rather than events.

\[ x := x + 2 \quad \text{send}(x) \quad x := x - 1 \]

\[ y := y + 3 \quad \text{receive}(y) \quad y := 2 \times y \]

---

Equivalent state based model

\[ pc, x \]

\[ x := x - 1 \quad x := x + 2 \]

\[ y := y + 3 \quad y := 2 \times y \]

\[ 1, 3 \quad 2, 3 \quad 3, 2 \]
Consistent Global States in the State based Model

(a) Event Based Model

(b) State Based Model

(c) CGS

(d) CGS
Talk Outline

1. Motivation
2. Background: Posets and Lattices
   3. Global Predicate Detection Problem
      • Cooper and Marzullo’s Algorithm
      • Alagar and Venkatesan’s Algorithm
      • Lexical Enumeration of Consistent Global States
3. Predicate Detection for Special Classes
   • Linear Predicates
   • Relational Predicates
4. Slicing
5. Basis Temporal Logic
   • Syntax and Semantics
   • Semiregular Predicates
   • Algorithm to detect BTL

RV’14 Tutorial (Garg and Mittal)
Global Predicate Detection

**Predicate:** A global condition expressed using variables on processes e.g., more than one process is in critical section, there is no token in the system

**Problem:** find a consistent cut that satisfies the given predicate

The global predicate may express: a software fault or a global breakpoint
Two interpretations of predicates

Possibly: $\Phi$: exists a path from the initial state to the final state along which $\Phi$ is true on some state

Definitely: $\Phi$: for all paths from the initial state to the final state $\Phi$ is true on some state
Detecting Possibly: $\Phi$

- Centralized Checker Process
- Send relevant events to the checker process
- Include dependency information for events
- Checker process enumerates consistent global states
**Tracking Dependency**

**Problem:** Given \((E, \rightarrow)\), assign timestamps \(v\) to events in \(E\) such that 
\[\forall e, f \in E : e \rightarrow f \iff v(e) < v(f)\]

**Online Timestamps:** Vector Clocks [Fidge 89, Mattern 89]:
- **all events:** increment \(v[i]\) after each event
- **send events:** piggyback \(v\) with the outgoing message
- **receive events:** compute the max with the received timestamp
Detecting Possibly : $B$ — Enumeration of Consistent Global States

(a)

BFS: 00, 01, 10, 11, 20, 12, 21, 13, 22, 23, 33

DFS: 00, 10, 20, 21, 22, 23, 33, 11, 12, 13, 01

(b)

(c)
Challenges for Lattice Enumeration

- The number of CGS is exponential in the number of processes
- The lattice cannot be stored in the main memory
- What if the poset is infinite?
Cooper and Marzullo’s Algorithm

[Cooper and Marzullo 91]
Implicit BFS Traversal

*current*: list of the global states at the current level.
Initially, *current* has only one global state, the initial global state

```
repeat
    enumerate current;
    last := current;
    current = global states reached from last in one step;
until (current is empty)
```
Cooper and Marzullo’s Algorithm

[Cooper and Marzullo 91]
Implicit BFS Traversal

`current`: list of the global states at the current level. Initially, `current` has only one global state, the initial global state

repeat
    enumerate `current`;
    `last` := `current`;
    `current` = global states reached from `last` in one step;
until (`current` is empty)

Problems:
Repeated Enumeration: a CGS can be reached from multiple global states.
Space Complexity: need to store a level of the lattice – exponential in the number of processes
Avoiding Repeated Enumeration

Idea: explore events only in a sorted order
an event $e$ is explored from a global state $G$ iff $e$ is bigger than all the events in $G$
Revised BFS Algorithm

$Q$: set of CGS at the current level initially $\{(0, 0, \ldots, 0)\}$;
$\sigma$: a topological sort of all events in $(E, \rightarrow)$

while $(Q \neq \emptyset)$ do
    $G := \text{remove_first}(Q)$;
    for all events $e$ enabled in $G$ do // generate CGS at the next level
        if $(\forall f \in \text{maximal}(G) : \sigma(f) < \sigma(e))$ then
            $H := G \cup \{e\}$;
            append($Q$, $H$);
        endif;
    endfor;
endwhile;

Time: $O(n^2 M)$ \quad Space complexity: $O(nw_L)$

$n$: number of processes \quad $M$: number of CGS

$w_L$: width of the lattice $L$.

Problem: $w_L$ is exponential in $n$ in the worst case.
Implicit Depth First Search

Idea: Instead of storing width, store the height of the lattice
Use implicit Depth-First-Search [Alagar and Venkaesan 94]

\[ G: \text{array}[1..n] \text{ of integer}; \quad // \text{current global state} \]
\[ pred: \text{array}[1..n] \text{ of integer}; \quad // \text{predecessor info for the next event} \]

Function dfsTraversal(int \( k \)) // event \( e \) at \( P_k \) enabled in the current state
- \( G[k]++ \);
- enumerate(\( G \));
- compute \( pred[k] \) using the vector clock for \( e \);
- forall \((j \neq k)\): if \((e_j \text{ depends on } e)\) then \( pred[j]-- \);
-forall \((j)\) with next event \( e_j \)
  - if \((pred[j] = 0)\) and \(\sigma(e) < \sigma(e_j)\)
    - dfsTraversal(\( j \));
- restore values of \( G \) and \( pred \);
end;

Problem: Stack can grow to \( O(E) \) the number of events in the computation
Lexical Enumeration of Consistent Global States

BFS: 00, 01, 10, 11, 20, 12, 21, 13, 22, 23, 33

DFS: 00, 10, 20, 21, 22, 23, 33, 11, 12, 13, 01

Lexical: 00, 01, 10, 11, 12, 13, 20, 21, 22, 23, 33
Lexical Order

$G \prec_l H$ iff

$$\exists k : (\forall i : 1 \leq i \leq k - 1 : G[i] = H[i]) \land (G[k] < H[k]).$$

Lemma

$$\forall G, H : G \subseteq H \Rightarrow G \leq_l H.$$
Algorithm for Lex Order

\textit{nextLex}\((G)\): next consistent global state in lexical order

\begin{verbatim}
var
  G : consistent global state initially (0, 0, ..., 0);
enumerate(G);
while \((G < \top)\)
  \(G := \text{nextLex}(G)\);
enumerate(G);
endwhile;
\end{verbatim}

No intermediate consistent global nodes stored
Computing next consistent global state in lexical order

Lemma

Given any global state \( K \) (possibly inconsistent), the set of all consistent global states that are greater than or equal to \( K \) in the CGS lattice is a sublattice.

Corollary

- There exists a minimum consistent global state \( H \) that is greater than or equal to a given global state \( K \).

Notation

- \( \text{succ}(G, k) \): advance along \( P_k \) and reset components for \( P_i \) (\( i > k \)) to 0.
  
  e.g. \( \text{succ}(<7, 5, 8, 4>, 2) = <7, 6, 0, 0> \)
  
  \( \text{succ}(<7, 5, 8, 4>, 3) \) is \( <7, 5, 9, 0> \).

- \( \text{leastConsistent}(K) \): the least consistent global state greater than or equal to a given global state \( K \) in the \( \subseteq \) order.
Computation of $\text{nextLex}(G)$

**Theorem**

\[ \text{nextLex}(G) = \text{leastConsistent}(\text{succ}(G, k)) \]

where $k$ is the index of the process with the smallest priority which has an event enabled in $G$.

**Example:** Let $G = (4, 3, 3)$. Then $k = 2$, $\text{succ}(G, k) = (4, 4, 0)$.
Therefore, $\text{nextLex}(G) = (4, 4, 1)$. 
Algorithm for Lex Order

\texttt{nextLex}(G): next consistent global state in lexical order

\texttt{var} \\
\hspace{1em}G : consistent global state initially (0, 0, ..., 0); \\
\hspace{1em}enumerate(G); \\
\hspace{1em}while (G < \top) \\
\hspace{2em}k := smallest priority process with an event enabled in G \\
\hspace{2em}G := leastConsistent(succ(G, k)) \\
\hspace{2em}enumerate(G); \\
endwhile \\

\(k, succ(G, k)\) and \texttt{leastConsistent()} can be computed in \(O(n^2)\) time using vector clocks.

\[\text{[Garg03]}\]
Parallel and Online Algorithms

Partition the lattice into multiple interval sublattices
Assume that events arrive in a total order $\sigma$ consistent with $\rightarrow$.
for every event $e$
- $G_{\text{min}}(e) =$ smallest consistent global state that contains $e$
- $G_{\text{bnd}}(e) = \{ f | \sigma(f) \leq \sigma(e) \}$

Theorem[Chang and Garg 14]: Consider the set of all interval lattices, $I(e)$,
\{ $G | G_{\text{min}}(e) \subseteq G \subseteq G_{\text{bnd}}(e)$ \}. These interval lattices are mutually disjoint and cover the entire lattice of all consistent global states.

ParaMount: A parallel implementation for detecting predicates in concurrent systems [Chang and Garg 14]
Talk Outline

1. Motivation
2. Background: Posets and Lattices
3. Global Predicate Detection Problem
   - Cooper and Marzullo’s Algorithm
   - Alagar and Venkatesan’s Algorithm
   - Lexical Enumeration of Consistent Global States
4. Predicate Detection for Special Classes
   - Linear Predicates
   - Relational Predicates
5. Slicing
6. Basis Temporal Logic
   - Syntax and Semantics
   - Semiregular Predicates
   - Algorithm to detect BTL

RV'14 Tutorial (Garg and Mittal)
Exploit the structure/properties of the predicate

- **stable predicate** [Chandy and Lamport 85]
  - once the predicate becomes true, it stays true
  - e.g., deadlock

- **observer independent predicate** [Charron-Bost et al 95]
  - occurs in one interleaving $\implies$ occurs in all interleavings
  - e.g., stable predicates, disjunction of local predicates

- **linear predicate** [Chase and Garg 95]
  - closed under meet, e.g., there is no leader in the system
  - **relational predicate**: $x_1 + x_2 + \cdots + x_n \geq k$ [Chase and Garg 95] [Tomlinson and Garg 96]
  - e.g., violation of $k$-mutual exclusion

RV’14 Tutorial (Garg and Mittal) Monitoring Distributed Computations 37
Linearity

Crucial Element $\text{crucial}(G, e, B)$
For a consistent cut $G \subseteq^\neq E$ and a predicate $B$, $e \in E - G$ is crucial for $G$ if:

$$\forall H \supseteq G : (e \in H) \lor \neg B(H).$$

Linear Predicates
A predicate $B$ is linear if for all consistent cuts $G \subseteq^\neq E$,

$$\neg B(G) \Rightarrow \exists e \in E - G : \text{crucial}(G, e, B).$$
Examples of Linear Predicates: Conjunctive Predicates

- **mutual exclusion problem**: (P1 in CS) and (P2 in CS)
- **missing primary**: (P1 is secondary) and (P2 is secondary) and (P3 is secondary)
Channel Predicates: Observing hallways

Many properties require channels

Example: termination detection – all processes are idle and all channels are empty

Channel predicate: boolean function on the state of the unidirectional channel

channel state : sequence of messages sent - set of messages received

Linearity: Given any channel state in which the predicate is false, either the next event at the receiver is crucial, or the next event at the sender is crucial
Linear Channel Predicates

- **Empty channels**
  If false, then it cannot be made true by sending more messages.
  The next event at the receiver is crucial.

- **Channel has more than three red messages**
  The next event at the sender is crucial.

- **Channel has exactly three red messages**
  If less than three, the next event at the sender is crucial,
  If more than three, the next event at the receiver is crucial
Non-linear Channel Predicates

\[ B \equiv \text{Channel has an odd number of messages} \]

The set of cuts satisfying the predicate is not linear.
Theorem: [Chase and Garg 95] A predicate $B$ is linear if and only if it is meet-closed (in the lattice of all consistent cuts).
- A predicate $P$ is **meet-closed** if all the cuts that satisfy the predicate are closed under intersection. $(C_1 \models P \land C_2 \models P) \Rightarrow (C_1 \cap C_2) \models P$.

- A predicate $P$ is **join-closed** if all cuts that satisfy the predicate are closed under union. i.e., $(C_1 \models P \land C_2 \models P) \Rightarrow (C_1 \cup C_2) \models P$.

- A predicate $P$ is **regular** if it is join-closed and meet-closed.

- A predicate $P$ is **stable**, if $\forall C_1, C_2 \in L: C_1 \models P \land C_1 \subseteq C_2 \Rightarrow C_2 \models P$. 

---

**Special Classes of Predicates**
Example: Special Classes of Predicates

(i) meet closed predicate
(ii) join closed predicate
(iii) regular predicate
Detecting Linear Predicates

(Advancement Property) There exists an efficient (polynomial time) function to determine the crucial event.

Theorem: Any linear predicate that satisfies advancement property can be detected efficiently.

Example: A conjunctive predicate, \( l_1 \land l_2 \land \ldots \land l_n \), where \( l_i \) is local to \( P_i \).
Importance of Conjunctive Predicates

Sufficient for detection of the following global predicates

- **boolean expression of local predicates** which can be expressed as a disjunction of a small number of conjunctions.
  
  *Example:* \( x, y \) and \( z \) are in three different processes. Then,

  \[
  \text{even}(x) \land ((y < 0) \lor (z > 6))
  \]

  \[
  \equiv \\
  (\text{even}(x) \land (y < 0)) \lor (\text{even}(x) \land (z > 6))
  \]

- **predicate satisfied by only a small number of values**

  *Example:* \( x \) and \( y \) are in different processes.

  \( x = y \) is not a *local* predicate but \( x \) and \( y \) are binary.
Conditions for Conjunctive Predicates

\[(l_1 \land l_2 \land \ldots l_n)\] is true iff there exist \(s_i\) in \(P_i\) such that \(l_i\) is true in state \(s_i\), and \(s_i\) and \(s_j\) are incomparable for distinct \(i, j\).
Weak Conjunctive Predicates: Centralized Algorithm

Each non-checker process maintains its local vector and sends the vector clock to the checker process whenever:

- local predicate is true
- at most once in each message interval.

Optimization: Sufficient to send the vector once after any message is sent.

Space complexity: $O(n)$

Message complexity: $O(m_s)$, $m_s =$ number of program messages sent.

Time complexity: detection of local predicates, maintain vector clock $O(n)$

[Garg and Waldecker 94]
Checker Process

\( n \) queues of vectors

Steps

1. Begin with the initial global state
2. Eliminate any vector that happened before any other vector along the current global state.

Predicate is true for the first time

- all vectors are pairwise concurrent

Predicate is false

- if we eliminate the final vector from any process
Overhead: Checker processes

Space complexity
- \( n \) queues, each containing at most \( m \) vectors

Time complexity
- The algorithm for checker requires at most \( O(n^2 m) \) comparisons.
- Any algorithm which determines whether there exists a set of incomparable vectors of size \( n \) in \( n \) chains of size at most \( m \), makes at least \( mn(n - 1)/2 \) comparisons.
Disadvantages of above algorithm

Centralized
- Checker process may become a bottleneck

Space requirements
- Queues at the checker process may grow large

Message complexity
- many additional messages to the checker process
Token-based Algorithm

A monitor process is active only if it has the token. Token consists of two vectors $G$ and $color$.

$G$: global state vector

- $G[i] = k$ indicates that state $(i, k)$ is part of the current cut.

$color$: indicates which states have been eliminated.

- If $color[i] = red$ then state $(i, G[i])$ has been eliminated and can never satisfy the global predicate.
- If $color[i] = green$, then there is no state in $G$ such that $(i, G[i])$ happened before that state.
Monitor Process Algorithm: $M_i$

```plaintext
var candidate:array[1..n] of integer;

on receiving the token (G,color)
    while (color[i] = red) do
        receive candidate from application process $P_i$
        if (candidate.vclock[i] > G[i]) then
            G[i] := candidate.vclock[i]; color[i] := green;
        endwhile
    for $j \neq i$
        if (candidate.vclock[j] > G[j]) then
            G[j] := candidate.vclock[j];
            color[j] := red;
        endif
    endfor
    if (∃ j: color[j] = red) then send token to $P_j$
    else detect := true;
```
Analysis of Single-Token WCP Algorithm

**Theorem**

For any computation \((E, \rightarrow)\) and any conjunctive predicate \(B\), if \(B\) holds in \((E, \rightarrow)\), then the Single-Token WCP algorithm returns the least CGS that satisfies \(B\). If \(B\) is false, then the algorithm returns false.

**Work complexity:** \(O(n^2m)\)

Every time a state is eliminated, \(O(n)\) work is performed. There are at most \(mn\) states.

**Message complexity:** \(O(mn)\).

**Communication bit complexity:** \(O(n^2m)\).

Size of both the token and the candidate messages is \(O(n)\).

**Space complexity:** \(O(mn)\) space per process.

\(m\): maximum number of vectors per process, \(n\): number of processes
Other WCP algorithms

A completely distributed algorithm [Chase and Garg 94]

Uses Dijkstra and Scholten’s termination detection algorithm

Keeping queues shorter [Chiou and Korfhage 95]

eliminate vectors that are useless

Avoiding control messages [Hurfin, Mizuno et al 96]

piggyback info/token with application messages
Talk Outline

1 Motivation

2 Background: Posets and Lattices

3 Global Predicate Detection Problem
   - Cooper and Marzullo’s Algorithm
   - Alagar and Venkatesan’s Algorithm
   - Lexical Enumeration of Consistent Global States

4 Predicate Detection for Special Classes
   - Linear Predicates
   - Relational Predicates

5 Slicing

6 Basis Temporal Logic
   - Syntax and Semantics
   - Semiregular Predicates
   - Algorithm to detect BTL
Relational Predicates: Binary Variables

Problem: Given \((S, \rightarrow)\)

\[ B \equiv x_1 + x_2 + x_3 \ldots x_n \geq k \]

where \(x_i\) resides on process \(P_i\).

Example:

\(x_i\): \(P_i\) is using the shared resource.

Are there \(k\) or more processes using the resource concurrently?

Equivalent Problem: Is there an antichain \(H \subseteq S\) such that the size of \(H\) is at least \(k\) and \(x\) is true on local states in \(H\).

[Tomlinson and Garg 96]
Using Dilworth’s Theorem

**Dilworth’s Chain Partition Theorem**: For any poset \((X, \leq)\), size of a maximum sized antichain (width) =

the minimum number of chains that covers the poset

\[ b_1 \quad b_2 \quad b_3 \]
\[ a_1 \quad a_2 \quad a_3 \]

\( k \) queues of vector clocks can be merged into \( k - 1 \) queues iff there is no antichain of size \( k \).
Relational Predicate Algorithm

**Input:** $n$ queues of vector clocks;  
**Output:** true iff $\sum_i x_i \geq k$  

for $i := 1$ to $n - k + 1$ do  
    pick smallest $k$ chains and merge them into $k - 1$ chains;  
    if not possible then  
        found an antichain of size $k$;  
        return true; // the antichain = CGS where the predicate holds  
endfor;  
return false; // only $k - 1$ chains left
Generalized Merging

Theorem: Let the poset be presented as \( k \) queues of vector clocks. There exists an efficient algorithm that can merge \( N \) queues into \( N - 1 \) queues in an online fashion whenever possible.  

[Tomlinson and Garg 96]
How to merge queues of vectors?

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$(1,0,0)$</td>
<td>d: $(0,1,0)$</td>
<td>f: $(2,0,0)$</td>
</tr>
<tr>
<td>b</td>
<td>$(1,1,0)$</td>
<td>e: $(2,2,0)$</td>
<td>g: $(2,3,0)$</td>
</tr>
<tr>
<td>c</td>
<td>$(1,2,0)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Naive Strategy

Move a minimal element into any output queue in which it can be inserted.
After insertion of \(a, d, b, c\):

\[
\begin{array}{c|c}
Q_1 & Q_2 \\
\hline
a: (1,0,0) & d: (0,1,0) \\
b: (1,1,0) \\
c: (1,2,0) \\
\end{array}
\]

\[
\begin{array}{c|c|c}
P_1 & P_2 & P_3 \\
\hline
 & f: (2,0,0) & \\
e: (2,2,0) & g: (2,3,0) & \\
\end{array}
\]
Merge is Possible

\[ Q_1 \quad Q_2 \]
\[
a: (1,0,0) \quad d: (0,1,0) \\
b: (2,0,0) \quad b: (1,1,0) \\
e: (2,2,0) \quad c: (1,2,0) \\
g: (2,3,0) \\
\]

\( b: (1,1,0) \) is inserted in \( Q_2 \) and not \( Q_1 \).
Queue Insert Graph

\( G = (V, E) \): undirected graph called queue insert graph

- \( V \): set of \( k \) input queues
- \( E \): undirected edges on \( V \)

**Invariant 1:** \( G \) is a spanning tree
\( \Rightarrow \) there are exactly \( k - 1 \) edges in \( G \). Each edge is labeled with a unique output queue

**Invariant 2:** Let \((P_i, P_j)\) be labeled with \( Q_k \)
All elements of \( P_i \) and \( P_j \) are bigger than all elements of \( Q_k \) \( \Rightarrow \) Any element from \( P_i \) or \( P_j \) can be inserted at the tail of \( Q_k \).

![Diagram of queue insert graph]
\[
\begin{align*}
Q_1 & : (1,0,0) \\
Q_2 & : (0,1,0)
\end{align*}
\]
Using Queue Insert Graph

\[ b : (1, 1, 0) \in P_1 < e : (2, 2, 0) \in P_2 \]
delete \( b : (1, 1, 0) \) from \( P_1 \) and insert in an output queue.

Which one?

1. Add an edge between \( P_i \) and \( P_j \) in the spanning tree.
2. A unique cycle is formed. Let \((P_i, P_k)\) be the other edge incident on \( P_i \) in that cycle.
3. Remove \((P_i, P_k)\). Transfer its label to \((P_i, P_j)\) and insert the vector in the corresponding output queue.

Verify: Queue Insert Graph invariant is preserved.
Relational Predicates: Nonbinary Variables

Let $x_i$: number of tokens at $P_i$  
$\Sigma x_i < k$: loss of tokens

Algorithm: max-flow technique [Groselj 93, Chase and Garg 95], Consistent cut with minimum value = min cut in the flow graph

$$p_1 \quad x_1 = 8 \bullet 1 \bullet 2 \bullet 5$$

$$p_2 \quad x_2 = 9 \bullet 4 \bullet 9 \bullet 2$$

max-flow conversion

$$c_{e1} = 8 \quad c_{e2} = 9 \quad c_{e1} = 9 \quad c_{e2} = 24$$
Talk Outline

1 Motivation

2 Background: Posets and Lattices

3 Global Predicate Detection Problem
   - Cooper and Marzullo’s Algorithm
   - Alagar and Venkatesan’s Algorithm
   - Lexical Enumeration of Consistent Global States

4 Predicate Detection for Special Classes
   - Linear Predicates
   - Relational Predicates

5 Slicing

6 Basis Temporal Logic
   - Syntax and Semantics
   - Semiregular Predicates
   - Algorithm to detect BTL
Summary

- Space efficient algorithms for general predicates
- Time efficient algorithms for special classes of predicates

**Problem:** What if the predicate does not belong to one of the special classes?
Talk Outline

1. Motivation
2. Background: Posets and Lattices
3. Global Predicate Detection Problem
   - Cooper and Marzullo’s Algorithm
   - Alagar and Venkatesan’s Algorithm
   - Lexical Enumeration of Consistent Global States
4. Predicate Detection for Special Classes
   - Linear Predicates
   - Relational Predicates
5. Slicing
6. Basis Temporal Logic
   - Syntax and Semantics
   - Semiregular Predicates
   - Algorithm to detect BTL

RV’14 Tutorial (Garg and Mittal)
Basis Temporal Logic: Motivation

RCTL can handle only regular predicates. Even a simple formula such as $p \lor q$ is not regular.

Need for a logic:

- Sufficiently expressive
- Easy to write formulas in that logic
- Can detect them with polynomial time complexity polynomial in the number of processes, not the size of the formula
Basis Temporal Logic: Syntax

AP: Set of Atomic Propositions
Atomic Propositions are evaluated on a single global state.
A predicate in BTL is defined recursively as follows:

1. \( \forall l \in AP, \ l \) is a BTL predicate
2. If \( P \) and \( Q \) are BTL predicates then \( P \lor Q, P \land Q, \diamond P \) and \( \neg P \) are also BTL predicates

Example: \( B = \neg \diamond (\bigwedge red_i) \land token_0 \)
[Ogale and Garg 07]
Basis Temporal Logic: Semantics

$E$, $\rightarrow$: Poset (distributed computation)
$L$: Lattice of consistent global states of $(E, \rightarrow)$
$C$: A consistent global state of $(E, \rightarrow)$
$\lambda : L \rightarrow 2^{AP}$ set of atomic propositions true in any consistent global state

- $(C, L, \lambda) \models l \iff l \in \lambda(C)$ for an atomic proposition $l$
- $(C, L, \lambda) \models P \land Q \iff C \models P$ and $C \models Q$
- $(C, L, \lambda) \models P \lor Q \iff C \models P$ or $C \models Q$
- $(C, L, \lambda) \models \neg P \iff \neg(C \models P)$
- $(C, L, \lambda) \models \Diamond P \iff \exists C' \in L : (C \subseteq C' \text{ and } C' \models P)$

There exists a future consistent global state in which $P$ is true.
Special Classes of Predicates

(i) meet closed predicate

(ii) join closed predicate

(iii) regular predicate

{e3, f3} {e2, f3} {e1, f3} {f3}

{e2, f2} {e3, f2} {e1, f2} {f2}

{e1, f1} {f1}

{}
Basis of a Predicate

Given a computational lattice $L$, corresponding to a computation $E$, and a predicate $P$, a subset $S[P]$ of $L$ is a basis of $P$ if

1. **Compactness**: The size of $S[P]$ is polynomial in the size of computation $E$.

2. **Efficient Membership**: Given any consistent global state $C \in L$, there exists a polynomial time algorithm that takes $S[P]$, $E$ and $C$ as input and determines whether $(C, L) \models P$.

Examples

- **Predicate for an Order Ideal**: Sufficient to keep the largest CGS that satisfies $P$
- **Regular Predicate**: Sufficient to keep the slice (or join-irreducibles) of $(E, \rightarrow)$ with respect to $P$
Given a stable predicate $P$ and the computational lattice $L$, a stable structure is the set of ideals $\mathcal{I}$ such that a cut satisfies $P$ iff it does not belong to any of the ideals in $\mathcal{I}$. Therefore, $C \models P \iff \neg (C \in \bigcup_{I \in \mathcal{I}} I)$.
Talk Outline

1. Motivation

2. Background: Posets and Lattices

3. Global Predicate Detection Problem
   - Cooper and Marzullo’s Algorithm
   - Alagar and Venkatesan’s Algorithm
   - Lexical Enumeration of Consistent Global States

4. Predicate Detection for Special Classes
   - Linear Predicates
   - Relational Predicates

5. Slicing

6. Basis Temporal Logic
   - Syntax and Semantics
   - Semiregular Predicates
   - Algorithm to detect BTL
Semiregular Predicates

$P$ is a **semiregular** predicate if it can be expressed as a conjunction of a regular predicate with a stable predicate.

**Examples:**
- All processes are never *red* concurrently at any future state and process $P_0$ has the token. That is, $P = \neg \Diamond (\bigwedge red_i) \land \text{token}_0$.
- At least one process is beyond phase $k$ (stable) and all the processes are red.

**Claim:** All regular predicates and stable predicates are semiregular.
A semiregular predicate is join-closed.

If $P$ and $Q$ are semiregular then so is $P \land Q$. Both regular and stable predicates are closed under conjunction.
Properties of Semiregular Predicates

If \( P \) is a semiregular predicate then \( \Diamond P \) and \( \Box P \) are semiregular.

- If \( P \) is semiregular, \( P \) has a unique maximal cut, say \( C_{\text{max}} \) and \( \Diamond P \) is an ideal of the lattice that contains all cuts less than or equal to \( C_{\text{max}} \).
- \( \Box P \) is a stable predicate for any \( P \), and therefore it is also semiregular.
A semiregular structure, $g$, is a tuple $(\langle \text{slice}, \mathcal{I} \rangle)$ consisting of a slice and a stable structure, such that the predicate is true in cuts that belong to their intersection.

$$C \in g \iff (C \in \text{slice}) \land \neg (C \in \bigcup_{I \in \mathcal{I}} I).$$
Algorithm to Detect BTL: Base Case

/*The input predicate $P_{in}$ has all negations pushed
- inside to the $\Diamond$ operator or to the atomic propositions */
/* each semiregular structure is represented as a tuple $\langle \text{slice}, \text{maxCuts} \rangle$
- where $\text{maxCuts}$ is the set of maximal cuts
- of the ideals $\mathcal{I}$ representing the stable structure */

definition getBasis(Predicate $P_{in}$)
    output: $S[P_{in}]$, a set of semiregular structures
    Case 1. (Base case: local predicates) : $P_{in} = l$ or $P_{in} = \neg l$
        $S[P_{in}] := \{ \langle \text{slice}(P), \{\} \rangle \}$
    Case 2. $P_{in} = P \lor Q$
    Case 3. $P_{in} = P \land Q$
    Case 4. $P_{in} = \Diamond P$
    Case 5. $P_{in} = \neg \Diamond P$
    return $S[P_{in}]$
Algorithm to Detect BTL: Conjunctions and Disjunctions

```plaintext
function getBasis(Predicate P_in)
output: S[P_in], a set of semiregular structures

Case 1. (Base case: local predicates) : P_in = l or P_in = ¬l
    S[P_in] := {⟨slice(P), {}⟩}

Case 2. P_in = P ∨ Q
    S[P] := getBasis(P); S[Q] := getBasis(Q);
    S[P_in] := S[P] ∪ S[Q];

Case 3. P_in = P ∧ Q
    S[P] := getBasis(P); S[Q] := getBasis(Q);
    S[P_in] := \bigcup_{g_p \in S[P], g_q \in S[Q]} \{⟨g_p.slice \land g_q.slice,
                                           g_p.maxCuts \cup g_q.maxCuts⟩\};

Case 4. P_in = ♦P

Case 5. P_in = ¬♦P

return S[P_in]
```
Algorithm to Detect BTL: Modalities

function getBasis(Predicate $P_{in}$)
output: $S[P_{in}]$, a set of semiregular structures

Case 1. (Base case: local predicates) : $P_{in} = l$ or $P_{in} = \neg l$

Case 2. $P_{in} = P \lor Q$

Case 3. $P_{in} = P \land Q$

Case 4. $P_{in} = \Diamond P$
    
    $S[P] := \text{getBasis}(P)$;
    $S[P_{in}] := \bigcup_{g \in S[P]} \{ \langle \Diamond (g \cdot \text{slice}) , \{\} \rangle \}$;

Case 5. $P_{in} = \neg \Diamond P$
    
    $S[P] := \text{getBasis}(P)$;
    */ slice$_{\text{orig}}$ is the original computation */
    $S[P_{in}] := \{ \langle \text{slice}_{\text{orig}} , \bigcup_{g \in S[P]} \{ \maxCutIn(g \cdot \text{slice}) \} \rangle \}$;

Remove all empty semiregular structures from $S[P_{in}]$;

return $S[P_{in}]$
Complexity Analysis

**Theorem**

The total number of ideals $|I|$ in the basis computed by the algorithm to detect a BTL predicate $P$ with $k$ operators is at most $2^k$.

**Theorem**

The time complexity of the algorithm to detect a BTL formula is polynomial in the number of events ($|E|$) and the number of processes ($n$) in the computation.
Conclusions

- Lattice properties are crucial in monitoring distributed computations
Additional Tutorial

- Elements of Distributed Computing Wiley & Sons 2002

- Introduction to Lattice Theory with Computer Science Applications (Expected December 2014)
Acknowledgements

- Brian Waldecker: Weak and Strong Conjunctive Predicates
- Alexander Tomlinson: Relational Predicates
- Richard Kilgore, Roger Mitchell: Channel Predicates
- Craig Chase: Distributed Algorithm for Conjunctive Predicate
- Michel Raynal, Eddy Fromentin: Control Flow Predicates
- Ashis Tarafdar: Controlling Computations
- Neeraj Mittal: Slicing
- Alper Sen: Regular CTL
- Anurag Agarwal: Online Chain Decomposition
- Selma Ikiz: Incremental Chain Decomposition
- Arindam Chakraborty: Lattice Congruences
- Vinit Ogale: Basis Temporal Logic
- Yen-Jung Chang: Parallel and Online CGS Enumeration
- Himanshu Chauhan, Aravind Natarajan: Distributed Slicing Algorithms

Blue: Graduate Students at PDSL, UT Austin  Green: Graduate Students elsewhere  Red: Faculty Members
Additional Topics

- Monitoring for liveness properties: Infinite (periodic posets)
- Quotient Construction for Distributive Lattices
  Collapsing sublattices such that temporal logic formula is true in the original lattice iff it holds in the reduced lattice
- Online Chain Partition
  Events arrive in an online fashion. Insert them into as few chains as possible
Online Chain Decomposition

- Elements of a poset presented in a total order consistent with the poset
- Assign elements to chains as they arrive
- Can be viewed as a game between
  - Bob: present elements
  - Alice: assign them to chains
- For a poset of width $k$, Bob can force Alice to use $k(k + 1)/2$ chains. Any online algorithm can be forced to use $k^2$ chains [Felsner 97].

![Diagram of a poset with elements x, u, y, z, and u connected to x and y, y connected to z, and x connected to u.]
Online Chain Decomposition

An efficient online algorithm that uses at most $k^2$ chains with at most $O(k^2)$ comparisons per event. [Aggarwal and Garg 05]

- Use $k$ sets of queues $B_1, B_2, \ldots, B_k$. The set $B_i$ has $i$ queues with the invariant that no head of any queue is comparable to the head of any other queue.
- For a new element $z$, insert it into the first queue $q$ in $B_i$ with its head less than $z$.
- Swap remaining queues in $B_i$ with queues in $B_{i-1}$.
Motivation for Control

*Who controls the past controls the future, who controls the present controls the past...*  
George Orwell, *Nineteen Eighty-Four.*

- maintain global invariants or proper order of events
  - Examples: Distributed Debugging
    - ensure that $\text{busy}_1 \lor \text{busy}_2$ is always true
    - ensure that $m_1$ is delivered before $m_2$
    - maintain $\neg \text{CS}_1 \lor \neg \text{CS}_2$
- Fault tolerance
  - On fault, rollback and execute under control
- Adaptive policies
  - procedure A (B) better under light (heavy) load
Is the future known?

Yes: offline control
  applications in distributed debugging, recovery, fault tolerance..

No: online control
  applications: global synchronization, resource allocation

Delaying events vs Changing order of events
  supervisor simply adds delay between events
  supervisor changes order of events
Delaying events: Offline control

Maintain at least one of the process is not red
Can add additional arrows in the diagram such that the control relation should not interfere with existing causality relation (otherwise, the system deadlocks)
Problem: Instance: Given a computation and a boolean expression $q$ of local predicates

Question: Is there a non-interfering control relation that maintains $q$? This problem is NP-complete [Tarafdar and Garg 97]
Delaying events: disjunctive predicates

Efficient algorithm for disjunctive predicates

Example: at least one of the philosopher does not have a fork

Result: a control strategy exists iff there is no set of overlapping false intervals

\[
\text{overlap}(I_1, I_2) = (I_1.\text{lo} \rightarrow I_2.\text{hi}) \land (I_2.\text{lo} \rightarrow I_1.\text{hi})
\]

Result: There exists an $O(n^2 m)$ algorithm to determine the strategy $n =$ number of processes $m =$ number of states per process