Formal Methods for Monitoring Distributed Computations

Vijay K. Garg

Parallel and Distributed Systems Lab,
Department of Electrical and Computer Engineering,
The University of Texas at Austin,
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?
Why worry about a single Distributed Computation?

The current computation affects the results.
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 \(e \rightarrow f\).

\[\text{[Lamport 78]}\]
Tracking Dependency

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

**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
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, y, z, and u connected by arrows]

**FRIDA’15 (Garg)** Forma
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 $B_1, B_2, ..., B_k$. 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 $B_{i-1}$.
Open Problem 1: Online Chain Decomposition
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
\[ \forall e, f \in E : (f \in G) \land (e \rightarrow f) \Rightarrow (e \in G) \]
Global Snapshot Algorithm

Chandy-Lamport’s Algorithm for FIFO systems: white event red event
State of a channel: all messages sent by a white process received by a red process. marker along each channel:
Message complexity: $O(n^2)$ for dense graphs
Reducing Message Complexity

- Assume Spanning Tree

```plaintext
initiate() // enabled only if (color=white)
take local checkpoint;
color = red;
send “init” to all processes connected by tree edges;

On receiving “init” message or a red message on edge e
if (color = white)
take local checkpoint;
color = red;
send “init” to all tree edges except e
```

**Figure**: Common Initiation Component

Compute the total number of white messages sent by processes: $w$ By using convergecast on the spanning tree.
Distributed Trigger Counting Problem

$n$ processes $w$ triggers that arrive at these processes DTC Problem: Raise an alarm when all triggers have arrived
The set of all Consistent Global States 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
- join $\rightarrow$ union

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

union distributes over intersection
Talk Outline

1 Motivation

2 Global Predicate Detection Problem
   - Cooper and Marzullo’s Algorithm

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

\[ X \quad Y \]

\[ p_1 \]

\[ p_2 \]

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
Detecting *Possibly* : $B$ — Enumeration of Consistent Global States

(a) $e_1 \rightarrow e_2 \rightarrow e_3$

(b) $f_1 \rightarrow f_2 \rightarrow f_3$

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

DFS: 00, 10, 20, 21, 22, 23, 33, 11, 12, 13, 01
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 \( G \)
Lexical Enumeration of Consistent Global States

(b) 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

(c)
Lexical Order

\[ G <_l H \text{ iff } \]

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

Lemma

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

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

\begin{verbatim}
var
  G : consistent global state initially (0, 0, ..., 0);
enumerate(G);
while (G < \top)
  G := nextLex(G);
enumerate(G);
endwhile
No intermediate consistent global nodes stored
\end{verbatim}
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.* $\text{succ}(\langle 7, 5, 8, 4 \rangle, 2) = \langle 7, 6, 0, 0 \rangle$
  
  $\text{succ}\langle 7, 5, 8, 4 \rangle, 3\rangle$ is $\langle 7, 5, 9, 0 \rangle$.

- $\text{leastConsistent}(K)$: 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

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

\textbf{var}

\ \ G \ : \ consistent \ global \ state \ initially \ (0, 0, ..., 0);

\textbf{enumerate}(G);

\textbf{while} \ (G < \top)

\ \ k := \ smallest \ priority \ process \ with \ an \ event \ enabled \ in \ G

\ \ G := \ leastConsistent(succ(G, k))

\textbf{enumerate}(G);

\textbf{endwhile} ;

\textit{k}, \ \ succ(G, k) \ \text{and} \ \textit{leastConsistent}() \ \text{can be computed in} \ O(n^2) \ \text{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]
Open: A CAT algorithm for enumeration of Consistent Global States

Is there an algorithm that takes constant amortized time for enumeration of each consistent global state?
Talk Outline

1 Motivation

2 Global Predicate Detection Problem
   - Cooper and Marzullo's Algorithm

3 Predicate Detection for Special Classes
   - Linear Predicates
   - Relational Predicates
Predicate Detection for Special Cases

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 $\Rightarrow$ 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
Linearity

Crucial Element $crucial(G, e, B)$
For a consistent cut $G \subseteq E$ 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 $G \subsetneq E$,

$$\neg B(G) \Rightarrow \exists e \in E - G : crucial(G, e, B).$$
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)
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.
Linearity of Predicates and Meet-Closure

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).
Special Classes of Predicates

- 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 is **regular** if it is join-closed and meet-closed.

- A predicate $P$ is **stable**, if
  \[
  \forall C_1, C_2 \in L : C_1 \models P \land C_1 \subseteq C_2 \Rightarrow C_2 \models P.
  \]
Example: Special Classes of Predicates

(i) 

meet closed predicate

join closed predicate

regular predicate

(ii) 

(iii) 

stable 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 \(i, j\).
Talk Outline

1. Motivation

2. Global Predicate Detection Problem
   - Cooper and Marzullo’s Algorithm

3. Predicate Detection for Special Classes
   - Linear Predicates
   - Relational Predicates
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 \(H\) it 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]

\[
P_1 \quad C_1 \quad C_2 \quad C_3
\]

\[
P_2
\]

\[
P_3
\]

\[
C_1 \quad C_2
\]
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 \(a, d, b, c\):

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

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

\[
\begin{array}{ll}
Q_1 & Q_2 \\
a: (1,0,0) & d: (0,1,0) \\
f: (2,0,0) & b: (1,1,0) \\
e: (2,2,0) & c: (1,2,0) \\
g: (2,3,0) & \\
\end{array}
\]

\(b: (1,1,0)\) 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 \( Q_k \).
\begin{align*}
\text{Q}_1 & \quad \text{Q}_2 \\
a & : (1,0,0) & d & : (0,1,0) \\
\end{align*}

\begin{align*}
P_1 & \quad P_2 & \quad P_3 \\
b & : (1,1,0) & e & : (2,2,0) & f & : (2,0,0) \\
c & : (1,2,0) & g & : (2,3,0) \\
\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 $P_i$

$\sum x_i < k$: loss of tokens

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

\[
\begin{array}{ccc}
  a & b & c \\
p_1 & x_1 = 8 & 1 & 2 & 5 \\
p_2 & x_2 = 9 & 4 & 9 & 2 \\
\end{array}
\]

max-flow conversion

\[
\begin{array}{ccc}
  a & 1 & b & 2 & c \\
  d & e & f \\
  9 & 4 & 9 \\
\end{array}
\]

\[
\begin{array}{ccc}
  8 & 5 \\
  \infty \\
  9 & 2 \\
\end{array}
\]
Open problem: Efficient Distributed Online Detection Algorithms for Relational Predicates

Problem: Design an efficient monitor process at each process $P_i$ to detect if there exists a CGS $G$ in the computation such that $G$ satisfies the given relational predicate $B$. 
Talk Outline

1. Motivation

2. Global Predicate Detection Problem
   - Cooper and Marzullo’s Algorithm

3. Predicate Detection for Special Classes
   - Linear Predicates
   - Relational Predicates
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?
The Main Idea of Computation Slicing

- computation
- slicing
- slice
- retain all red consistent cuts
Computation Slice

Computation slice: a sub-computation such that:

1. it contains all consistent cuts of the computation satisfying the given predicate, and
2. it contains the least number of consistent cuts

- computation
  - e.g.
  - \( a, b \)
  - \( c, d \)

- predicate
  - e.g.
  - all sent messages have been received

Slicer

- sub-computation
  - e.g.
  - \( \{ c \} \{ a, d \} \{ b \} \)
Slicing Example

\[ P_1 \]
\[ a \quad b \]
\[ P_2 \]
\[ c \quad d \]
\[ P_3 \]
\[ e \quad f \]

\{a, c\} \quad \{b\}
\{e\} \quad \{d, f\}

slicing

no messages in transit
Slicing Example (Contd.)

\[
\{a, b, c, d, e, f\} \\
\{a, b, c, d, e\} \quad \{a, c, d, e, f\}
\]

\[
\begin{array}{c}
P_1 \quad a \\ b \\
\end{array} \\
\{a, b, c, d\} \quad \{a, b, c\} \quad \{a, c, d, e\}
\]

\[
\begin{array}{c}
P_2 \quad c \\ d \\
\end{array} \Rightarrow \quad \{a, b, c\} \quad \{a, b, e\} \quad \{a, c, d\} \quad \{a, c, e\}
\]

\[
\begin{array}{c}
P_3 \quad e \\ f \\
\end{array} \\
\{a, b\} \quad \{a, c\} \quad \{a, e\}
\]

\[
\begin{array}{c}
{a} \\ {e} \\
\end{array} \\
\{\}
\]
Characterization of Consistent Cuts

The set of consistent cuts of a computation forms a distributive lattice

1. The set forms a lattice
   - if $X$ and $Y$ are consistent cuts then so are $X \cap Y$ and $X \cup Y$
   - meet (infimum, greatest lower bound) $\rightarrow$ intersection
   - join (supremum, least upper bound) $\rightarrow$ union

2. The lattice is distributive
   - meet distributes over join
Characterization of Consistent Cuts

The set of consistent cuts of a distributed computation forms a distributive lattice.

1. meet operator: set intersection
2. join operator: set union
3. meet distributes over join
Basis Elements

Basis element: cannot be represented as join of two other elements

\{a, b, c, d, e, f\}

\{a, b, c, d, e\} \quad \{a, c, d, e, f\}

\{a, b, c, d\} \quad \{a, b, c\} \quad \{a, c, d\} \quad \{a, c, e\}

\{a, b\} \quad \{a, b, e\} \quad \{a, c, d\} \quad \{a, c, e\}

\{a\} \quad \{a, c\} \quad \{a, e\}

\{e\}

\{}

A basis element has exactly one incoming edge
Birkhoff’s Representation Theorem

Theorem

A distributive lattice can be recovered exactly from the set of its basis elements.

\[
\{a, b, c, d, e, f\} \\
\{a, b, c, d, e\} \quad \{a, c, d, e, f\} \\
\{a, b, c, d\} \quad \{a, b, c, e\} \quad \{a, c, d, e\} \\
\{a, b, c\} \quad \{a, b, e\} \quad \{a, c, d\} \quad \{a, c, e\} \\
\{a, b\} \quad \{a, c\} \quad \{a, e\} \\
\{a\} \quad \{e\} \\
\{\} \\
\]

All elements can be represented as join of some subset of its basis elements.
What about a Subset of Consistent Cuts?

\[ \{a, b, c, d, e, f\} \]

\[ \{a, b, c, d, e\} \quad \{a, c, d, e, f\} \]

\[ \text{X in subset and } \ Y \text{ in subset} \]

\[ \text{X} \cap \ Y \text{ in subset and } \ X \cup \ Y \text{ in subset} \]

\[ \{a\} \quad \{e\} \]

\[ \{\} \]

Sublattice: subset of consistent cuts closed under intersection and union
Representing a Sublattice

Theorem

A sublattice of a distributive lattice is also a distributive lattice.

A sublattice has a succinct representation.

\[{a, b, c, d, e, f}\]

\[Z\]

\[\{a, b, c, e\}\]

\[\{a, c, d, e, f\}\]

\[W\]

\[Y\]

\[\{a, b, c\}\]

\[\{a, c, e\}\]

\[\{a, c\}\]

\[\{b\}\]

\[\{e\}\]

\[\{d, f\}\]

\[X\]

\[Z\]

\[\{e\}\]

\[\{a, c, d, e, f\}\]

\[\{a, b, c, e\}\]

\[\{e\}\]

\[{a, c, e}\]

\[{a, c, d, e, f}\]

\[\{a, c\}\]

\[{b}\]

\[{d, f}\]

\[\{e\}\]

\[\{a, c, d, e, f\}\]

\[{a, b, c, e}\]

\[{a, c, d, e, f}\]

\[\{a, c\}\]

\[{b}\]

\[{d, f}\]

\[{e}\]

\[{a, c, d, e, f}\]

\[{a, b, c, e}\]

\[{a, c, d, e, f}\]
What if the Subset is not a Sublattice?

Add consistent cuts to complete the sublattice
Computing the Slice

Algorithm:
1. Find all consistent cuts that satisfy the predicate
2. Add consistent cuts to complete the sublattice
3. Find the basis elements of the sublattice

Can we find the basis elements without computing the sublattice?
Regular Predicate

Regular predicate: the set of consistent cuts satisfying the predicate is closed under intersection and union

\[ (X \text{ satisfies } b) \text{ and } (Y \text{ satisfies } b) \implies (X \cap Y \text{ satisfies } b) \text{ and } (X \cup Y \text{ satisfies } b) \]

Examples:
- conjunctive predicate—conjunction of local predicates
- there are at most (or at least) \( k \) messages in transit from process \( P_i \) to process \( P_j \)
- every “request” message has been “acknowledged” in the system
Properties of Regular Predicates

- The set of consistent cuts satisfying a regular predicate forms a sublattice of the set of all consistent cuts.

- The class of regular predicates is closed under conjunction:
  
  If $b_1$ and $b_2$ are regular predicates then so is $b_1 \land b_2$

- The class of regular predicates is a subset of the class of linear predicates.
Computing the Slice for Regular Predicate

\[
\begin{align*}
P_1 & \quad u & v \\
\text{ } & \quad w & x \\
\text{ } & \quad y & z \\

b = \text{“no messages in transit”}
\end{align*}
\]

Algorithm:

Step 1: Compute the least consistent cut \( L \) that satisfies \( b \)
\[ L = \{ \} \]

Step 2: Compute the greatest consistent cut \( G \) that satisfies \( b \)
\[ G = \{ u, v, w, x, y, z \} \]
Computing the Slice for Regular Predicate

Algorithm:
Step 3: For every event \( e \in G - L \), compute \( J(e) \) defined as:
  1. \( J(e) \) contains \( e \)
  2. \( J(e) \) satisfies \( b \)
  3. \( J(e) \) is the least consistent cut satisfying (1) and (2)

- \( J(e) \) is a basis element of the sublattice.
- \( J(e) \) is a regular predicate.
Slicing Example

\[ J(u) = J(w) \]
\[ u \quad v \]
\[ P_1 \]
\[ w \quad x \]
\[ P_2 \]
\[ y \quad z \]
\[ P_3 \]

\[ J(u) = \{ u, w \} \]
\[ J(v) = \{ u, v, w \} \]
\[ J(w) = \{ u, w \} \text{ (duplicate)} \]
\[ J(x) = \{ u, w, x, y, z \} \]
\[ J(y) = \{ y \} \]
\[ J(z) = \{ u, w, x, y, z \} \text{ (duplicate)} \]
How does Computation Slicing Help?

satisfy $b_1$

computation

detect $b_1 \land b_2$

slicing

retain all consistent cuts that satisfy $b_1$
	slice for $b_1$

detect $b_2$
Composing Two Slices: Conjunction

\[ \text{slice for } b_1 \land \text{slice for } b_2 = \text{slice for } b_1 \land b_2 \]

\[ \bigcap \quad = \bigcap \]
Composing Two Slices: Disjunction

\[ \text{slice for } b_1 \quad \lor \quad \text{slice for } b_2 \quad = \quad \text{slice for } b_1 \lor b_2 \]
Computing a Slice using Composition

Example: \((x_1 \lor x_2) (x_3 \lor x_4)\)

\[
\begin{array}{cccc}
  x_1 & x_2 & x_3 & x_4 \\
\end{array}
\]

compose: \(\lor\)

\[
\begin{array}{c}
  x_1 \lor x_2 \\
  x_3 \lor x_4 \\
\end{array}
\]

compose: \(\land\)

slice

\[
(x_1 \lor x_2) \land (x_3 \lor x_4)
\]
Path Based Properties

Some system properties are defined on paths rather than states.

Examples:

- Starvation freedom: Every request is eventually fulfilled.
- Deadlock freedom: If there is one or more request in the system, then some request is eventually fulfilled.
Temporal Logic Predicates

A path is a sequence of consistent cuts ending at the final consistent cut such that a successor of a cut is obtained by addition of a single vertex.

Temporal Operators: $\Diamond$, $\land$, $\Box$, $\text{and}$. 
$X$ satisfies $\Diamond b$. 
Temporal Operator:

\[ X \text{ satisfies } b. \]
Temporal Operator:

$X$ satisfies $b$. 
$X$ satisfies $\square b$. 
RCTL Predicates

Examples:

- Violation of mutual exclusion: Processes $P_1$ and $P_2$ are not in their critical sections simultaneously.
  \[ \Diamond CS_1 \land CS_2 \]

- Starvation Freedom: Every request is eventually fulfilled.
  \[ \square request \implies granted \]

RCTL: subset of CTL (Computation Tree Logic) where atomic propositions are regular and the operators are \[\Diamond, \land, \Box, \text{and} \]. [?]

An RCTL predicate is a regular predicate.
Computing the Slice for $b$

Observation 1: Any consistent cut that satisfies $b$ also satisfies $b$.

Idea 1: To compute $\text{Slice}(b)$, first compute $\text{Slice}(b)$ and then add edges to it.

Observation 2:
- Let $C$ denote a cycle in $\text{Slice}(b)$.
- If a consistent cut $X$ satisfies $b$, then $X$ includes all the vertices from $C$.

Idea 2: Eliminate all consistent cuts that do not contain all vertices of a cycle.
Computing the Slice for $b$

$b = \text{no messages in transit}$
Computing the Slice for $b$  

$b = \text{no messages in transit}$  

Consistent cuts that satisfy $b$: $\{a, c, d, e, f\}$ and $\{a, b, c, d, e, f\}$. 
Computing the Slice for $b$

All consistent cuts that do not satisfy $b$ have been eliminated.
Computing the Slice for $b$

$2.25\text{inRCTL/rctl4.pstex}_t$

All consistent cuts that do not include $a$ or $c$ are eliminated.
Computing the Slice for $b$

$2.25\text{inRCTL}\!/\text{rcrl5.pstex}_t$

All consistent cuts that do not include $d$ or $f$ are eliminated.
Efficient slicing algorithms have been developed for other predicates in RCTL.

Time complexity: $O(|b|n^2|E|)$, where
- $|b|$: number of boolean and temporal operators in $b$
- $n$: number of processes
- $|E|$: number of events
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 July 2015)
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
- **Sujatha Kashyap**: Applications to Model Checking
- **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
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 $busy_1 \lor busy_2$ is always true
    - ensure that $m_1$ is delivered before $m_2$
    - maintain $\neg CS_1 \lor \neg CS_2$

- Fault tolerance
  - On fault, rollback and execute under control

- Adaptive policies
  - procedure A (B) better under light (heavy) load
Models for Control

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 [Taraﬁdar 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

\[
\text{overlap}(I_1, I_2) = (I_1.\text{lo} \rightarrow l_2.\text{hi}) \land (l_2.\text{lo} \rightarrow I_1.\text{hi})
\]

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