Solved and Unsolved Problems in 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
- Assumptions made on the environment may not hold

Software Quality Assurance:

- Can I trust the results of the computation? Does it satisfy all required properties?
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 \(e \rightarrow f\).

[Lamport 78]
Talk Outline

1. Online Chain Decomposition

2. Global Predicate Detection
   - Distributed Trigger Counting
   - Enumerating Consistent Global States
   - Detecting Special Classes of Predicates

3. Monitoring for Temporal Logic Formulas
   - Computation Slicing
   - Detecting temporal logic formulas

4. Controlled Reexecution
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
Dynamic Chain Clocks

Problem with vector clocks: scalability, dynamic process structure

Idea: Computing the “chains” in an online fashion [Aggarwal and Garg PODC 05] for relevant events

Figure: (a) A computation with 4 processes (b) The relevant subcomputation
Optimal Offline Chain Decomposition

**Antichain**: Set of pairwise concurrent elements

**Width** ($w$): Maximum size of an antichain

**Dilworth’s Chain Partition Theorem** [Dilworth 50]: A poset of width $w$ can be partitioned into $w$ chains and cannot be partitioned into fewer than $w$ chains.

![Diagram of poset]

Requires knowledge of complete poset
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

Up-growing online posets: new element has to be a maximal element of the order presented so far
Online Chain Decomposition for Up-growing Orders

Theorem (Felsner 97): The value of the on-line chain partition game for up-growing orders of width $w$ is $\binom{w+1}{2}$.

Theorem: There exists an efficient online algorithm for online chain decomposition of up-growing orders that uses at most $w^2$ chains with at most $O(w^2)$ comparisons per event. [Aggarwal and Garg, PODC 05]

- Use $k$ sets of queues $B_1, B_2, ..., B_w$. 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 $B_{i-1}$. 
Online Chain Decomposition for General Posets

Open Problem 1: Give an algorithm for online chain decomposition on a poset that requires at most polynomial number of chains in $w$.

Theorem (Szemerédi, 1982): The value of the on-line chain partition game is at least $\binom{w+1}{2}$.

Theorem (Kierstead, 1981): The value of the on-line chain partition game is at most $(5^w - 1)/4$.

Theorem (Bosek and Krawczyk, 2009): The value of the on-line chain partition game is at most $w^{16\cdot \lg w}$.

On-Line Chain Partitions of Orders: A Survey
Bartlomiej Bosek, Stefan Felsner, Kamil Kloch, Tomasz Krawczyk, Grzegorz Matecki, Piotr Micek, Order, 2012.
Talk Outline

1. Online Chain Decomposition

2. Global Predicate Detection
   - Distributed Trigger Counting
   - Enumerating Consistent Global States
   - Detecting Special Classes of Predicates

3. Monitoring for Temporal Logic Formulas
   - Computation Slicing
   - Detecting temporal logic formulas

4. Controlled Reexecution
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

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

[Chandy and Lamport 85]
Global Snapshot Algorithms

white event: events executed before recording the local state
red event: events executed after recording the local state

Key idea:
Ensuring consistency: A process must be red to act on a red message

State of a channel: Record white messages received by red processes

Chandy-Lamport’s Algorithm for FIFO systems:
   Marker Rule: send a marker on all outgoing channels on turning red

Mattern’s Algorithm for non-FIFO systems [Mattern 89]:
   send the number of white messages sent along each channel

Message complexity: $O(n^2)$ for complete topology
Reducing Message Complexity of Global Snapshot Algorithms

Key idea: Do not send markers
- Use a spanning tree to turn all processes red
- Use the tree to compute the sum of all white messages sent
- Use the Distributed Trigger Counting (DTC) protocol to detect when all white messages have been received

Message Complexity is dominated by DTC protocol [Garg, Garg, Sabharwal, TPDS 10]
Distributed Trigger Counting (DTC) Problem

**System:** Completely connected topology

- \( n \) processes
- \( w \) triggers that arrive at these processes, \( w \gg n \)

**DTC Problem:** Raise an alarm when all triggers have arrived

- **No Fake Alarms:** Alarm is raised only when at least \( w \) triggers received.
- **No Dead State:** If \( w \) triggers are received, then an alarm is raised

**Naive Centralized Algorithm:**

Send a message to a master node whenever a trigger arrives

- **Message Complexity:** \( w \)
- **Maximum Receive Load:** \( w \)
Centralized Algorithm

**Idea:** send a message after $B$ triggers

Use rounds with repeated halving.

$\hat{w} =$ triggers not yet received

$B := \lceil \hat{w}/(2n) \rceil$

Master starts the end of round when the count reaches $\lfloor \hat{w}/2 \rfloor$.

- compute $w' :=$ all unreported triggers
- recompute $\hat{w} := \hat{w} - w'$ and $B$ for the next round.

**Claim:** The algorithm does not have any dead state.

at most $B - 1$ unreported triggers per process.

$\Rightarrow$ at most $\lceil \hat{w}/2 \rceil - 1$ total unreported triggers.

Reported triggers at Master $\leq \lfloor \hat{w}/2 \rfloor - 1$.

**Message Complexity:** $O(n \log w)$

**Maximum Receive Load:** $O(n \log w)$
LayeredRand Algorithm

\[ n = (2^L - 1) \text{ processors arranged in } L \text{ layers} \]
\[ i^{th} \text{ layer has } 2^i \text{ processors, } i = 0 \text{ to } L - 1. \]
Threshold for layer \( i \), \( \tau(i) = \lceil \hat{w} / 4 \cdot 2^i \cdot \log(n + 1) \rceil \)
\[ C(x) : \text{sum of triggers received by } x \text{ and some processors in layers below.} \]
For non-root processor \( x \) at layer \( i \):
if a trigger is received: \( C(x) + +; \)
  if \( C(x) \geq \tau(i) \)
    pick a processor \( y \) from level \( i - 1 \) at random and send a coin to \( y \).
    \( C(x) := C(x) - \tau(i); \)
if a coin is received from level \( i + 1 \):
    \( C(x) := C(x) + \tau(i + 1). \)
Root \( r \): maintains \( C(r) \)
  If \( C(r) > \lceil \hat{w} / 2 \rceil \), initiate end-of-round procedure
Message Complexity: \( O(n \log n \log w) \)
Maximum Receive Load: \( O(\log n \log w) \)
# DTC Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message Complexity</th>
<th>MaxRecvLoad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>$O(n \cdot (\log n + \log w))$</td>
<td>$O(n \cdot (\log n + \log w))$</td>
</tr>
<tr>
<td>LayeredRand</td>
<td>$O(n \cdot \log n \cdot \log w)$</td>
<td>$O(\log n \cdot \log w)$</td>
</tr>
<tr>
<td>CoinRand</td>
<td>$O(n \cdot (\log w + \log n))$</td>
<td>$O(\log w + \log n)$</td>
</tr>
<tr>
<td>TreeFillRand</td>
<td>$O(n \cdot \log(w/n))$</td>
<td>$O(\log(w/n))$</td>
</tr>
</tbody>
</table>

[Chakaravarthy, Choudhury, Garg, Sabharwal 12] ,
TreeFill: [Kim, Lee, Park, Cho 13]

**Lower Bound:** [Garg, Garg, Sabharwal 10]  
- **Message Complexity:** $O(n \log(w/n))$  
- **MaxRecvLoad:** $O(\log(w/n))$

**Notation:**  
$n$: Number of processes,  
$w$: Number of triggers
Open Problem 2: Give a deterministic algorithm for distributed trigger counting that requires $O(n \log(w/n))$ messages and has maximum receive load of $O(\log(w/n))$. 
Talk Outline

1. Online Chain Decomposition

2. Global Predicate Detection
   - Distributed Trigger Counting
   - Enumerating Consistent Global States
   - Detecting Special Classes of Predicates

3. Monitoring for Temporal Logic Formulas
   - Computation Slicing
   - Detecting temporal logic formulas

4. Controlled Reexecution
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
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
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*;
    \textit{last} := \textit{current};
    \textit{current} = global states reached from \textit{last} in one step;
until (\textit{current} is empty)

Problem:

Space Complexity: need to store a level of the lattice – exponential in the number of processes
Lexical Enumeration of Consistent Global States

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

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

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

(c)
Algorithm for Lex Order

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

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

No intermediate consistent global nodes stored [Garg PDCS 03]
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. $\text{succ}(\langle 7, 5, 8, 4 \rangle, 2) = \langle 7, 6, 0, 0 \rangle$
  - $\text{succ}(\langle 7, 5, 8, 4 \rangle, 3)$ 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 $nextLex(G)$

**Theorem**

$$nextLex(G) = leastConsistent(succ(G, k))$$

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 $k = 2$, $succ(G, k) = (4, 4, 0)$ Therefore, $nextLex(G) = (4, 4, 1)$. 
Algorithm for Lex Order

\[ \text{nextLex}(G) : \text{next consistent global state in lexical order} \]

\begin{verbatim}
var
    G : consistent global state initially (0, 0, ..., 0);
enumerate(G);
while (G < ⊤)
    \( k := \text{smallest priority process with an event enabled in } G \)
    \( G := \text{leastConsistent}(\text{succ}(G, k)) \)
enumerate(G);
endwhile;
\end{verbatim}

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

[Garg03]
Partition the lattice into multiple interval sublattices
Assume that events arrive in a total order $\sigma$ consistent with $\rightarrow$.
for every event $e$
  - $G_{min}(e) =$ smallest consistent global state that contains $e$
  - $G_{bnd}(e) = \{ f | \sigma(f) \leq \sigma(e) \}$

Theorem [Chang and Garg, PPoPP 14]: Consider the set of all interval lattices, $I(e)$,
$\{ G | G_{min}(e) \subseteq G \subseteq G_{bnd}(e) \}$.
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
## Summary of Enumeration Algorithms

**Problem**: Given a poset $P$, enumerate all its consistent global states.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time per Global State</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit BFS [Cooper, Marzullo 93]</td>
<td>$O(n^3)$</td>
<td>exp. in $n$</td>
</tr>
<tr>
<td>Implicit DFS [Alagar, Venkatesan 01]</td>
<td>$O(n^3)$</td>
<td>$O(</td>
</tr>
<tr>
<td>Gray Code [Pruesse, Ruskey93]</td>
<td>$O(</td>
<td>P</td>
</tr>
<tr>
<td>Ideal Tree [jegou95, habib01]</td>
<td>$O(\Delta(P))$</td>
<td>$O(</td>
</tr>
<tr>
<td>Lexical [Ganter, Reuter 91, Garg 03]</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>QuickLex [Chang, Garg 15]</td>
<td>$O(n \cdot \Delta(P))$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

**Notation**:

$P$: poset,

$n$: width of the poset,

$\Delta(P)$: maximum in-degree of any node in the Hasse Diagram of $P$
CAT Enumeration Problem

**Input:** A distributed computation (a poset)

**Output:** Enumeration of all consistent global states of the computation.

**Open Problem 3:**
Is there an algorithm that takes constant amortized time for enumeration of each consistent global state?
Talk Outline

1. Online Chain Decomposition

2. Global Predicate Detection
   - Distributed Trigger Counting
   - Enumerating Consistent Global States
   - Detecting Special Classes of Predicates

3. Monitoring for Temporal Logic Formulas
   - Computation Slicing
   - Detecting temporal logic formulas

4. Controlled Reexecution
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
Special Classes of Predicates

(i) \{e_1, f_3\} \{e_2, f_3\} \{e_1, f_2\} \{e_3, f_3\} \{e_2, f_1\} \\
\{f_3\} \{e_1, f_1\} \{f_2\} \{f_1\} \\
\{\}\n
(ii) \{e_1, f_3\} \{e_2, f_3\} \{e_1, f_2\} \{e_3, f_3\} \{e_2, f_1\} \\
\{f_3\} \{e_1, f_1\} \{f_2\} \{f_1\} \\
\{\}\n
(iii) \{e_3, f_3\} \{e_2, f_3\} \{e_1, f_3\} \{e_2, f_2\} \{e_3, f_2\} \\
\{f_3\} \{e_1, f_1\} \{f_2\} \{f_1\} \\
\{\}\n
- \text{meet closed predicate}
- \text{join closed predicate}
- \text{regular predicate}
- \text{stable predicate}
**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 $G \subseteq E$,

$$\neg B(G) \Rightarrow \exists e \in E - G : crucial(G, e, B).$$

**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).
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)
- **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.
Detecting Linear Predicates

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

**Theorem:** [Chase and Garg 95] 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 \( P_i \).
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]
Relational Predicate Algorithm

Using Dilworth’s Chain Partition Theorem: $k$ queues of vector clocks can be merged into $k - 1$ queues iff there is no antichain of size $k$.

**Theorem:** Let the poset be presented as $N$ 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]
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
Efficient Distributed Online Detection Algorithms for Relational Predicates

**Input:** An online computation \((E, \to)\)  
A relational predicate \(B\)

**Open Problem:**  
Design an efficient distributed online algorithm to detect if there exists a consistent global state \(G\) in the computation such that \(G\) satisfies the given relational predicate \(B\).
Talk Outline

1. **Online Chain Decomposition**

2. **Global Predicate Detection**
   - Distributed Trigger Counting
   - Enumerating Consistent Global States
   - Detecting Special Classes of Predicates

3. **Monitoring for Temporal Logic Formulas**
   - Computation Slicing
   - Detecting temporal logic formulas

4. **Controlled Reexecution**
Computation Abstraction: Computation Slicing

- Computation
- Slicing
- Slice
- Retain all red consistent cuts

FRIDA’15 (Garg)
Computation Slice: Definition

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

Diagram:

- Computation: e.g. {a, b, c, d}
- Slicer
- Sub-computation: e.g. {c, {a, d}, {b}}
- Predicate: e.g. all sent messages have been received
Slicing Example

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

\[ \{e\} \quad \{d, f\} \]
Slicing Example (Contd.)

\[ f = \Rightarrow \]

\[
\{a, b, c, d, e, f\} \\
\{a, c, d, e, f\} \\
\{a, b, c, d, e\} \\
\{a, c, d, e\} \\
\{a, b, c, d\} \\
\{a, b, e\} \\
\{a, c, d\} \\
\{a, c, e\} \\
\{a, b, c\} \\
\{a, b\} \\
\{a, c\} \\
\{a, e\} \\
\{a\} \\
\{e\} \\
\{\} \\
\]
Join-irreducible Elements

join-irreducible element: cannot be represented as join of two other elements

A join-irreducible element has exactly one incoming edge
Birkhoff’s Representation Theorem

**Theorem**

A distributive lattice can be recovered exactly from the set of its join-irreducible elements.

All elements can be represented as join of some join-irreducible elements.
Representing a Sublattice

**Theorem**

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

A sublattice has a succinct representation.
Computing the Slice

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

Can we find the join-irreducible elements without computing the sublattice?
Computing the Slice for Regular Predicate

Algorithm: For every event \( e \in E \), 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)

\[ B = \text{“no messages in transit”} \]
Slicing Example

\[
J(u) = J(w) \quad J(u) = \{u, w\}
\]

\[
J(v) = \{u, v, w\}
\]

\[
J(w) = \{u, w\} \text{ (duplicate)} \quad J(w) = \{u, w\}
\]

\[
J(x) = \{u, w, x, y, z\}
\]

\[
J(y) = \{y\}
\]

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

- **computation**
  - detect $b_1 \land b_2$
- **slicing**
- **slice for $b_1$**
  - detect $b_2$
- **retain all consistent cuts that satisfy $b_1$**
- **satisfy $b_1$**
Results on Slicing

Efficient polynomial-time algorithms for computing the slice for:

- **general predicate:**
  
  **Theorem:** Given a computation, if a predicate $b$ can be detected efficiently then the slice for $b$ can also be computed efficiently. [Mittal, Sen and Garg TPDS 07]

- **Temporal Logic Operators:** EF, AG, EG [Sen and Garg OPODIS 03]

- **Approximate slice:** For arbitrary boolean expression [Mittal and Garg DC 05]

- **Distributed Abstraction Algorithm:** Online Slicing Algorithm [Chauhan, Garg, Natarajan, Mittal SRDS13]
Efficient Computation Abstraction

**Input:** an online distributed computation \((E, \rightarrow)\)
A predicate \(B\)

**Open Problem:** Give an efficient algorithm to compute an abstraction of \((E, \rightarrow)\) with respect to \(B\) when \(B\) is not a regular predicate.
Monitoring Temporal Logic Formulas

Motivation: Detect formulas with polynomial time complexity polynomial in the size of the computation, 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: $B = \neg \Diamond (\land red_i) \land token_0$

[Ogale and Garg, DISC 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 \(C \models Q\)
- \((C, L, \lambda) \models \neg P \iff \neg (C \models P)\)
- \((C, L, \lambda) \models \Box P \iff \exists C' \in L : (C \subseteq C'\) and \(C' \models P)\)

There exists a future consistent global state in which \(P\) is true.
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**

- **Order Ideal Predicate**: $P$ is true in $G$ iff $G \subseteq W$
  Sufficient to keep the largest CGS $G$ that satisfies $P$

- **Regular Predicate**: Sufficient to keep the slice (or join-irreducibles) of $(E, \rightarrow)$ with respect to $P$
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.
Properties of Semiregular Predicates

- A semiregular predicate is join-closed
  regular and stable predicates are join-closed
- if $P$ and $Q$ are semiregular then so is $P \land Q$.
  both regular and stable predicates are closed under conjunction
- If $P$ is a semiregular predicate then $\Diamond P$ and $\Box P$ are 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 formulas

[Ogale and Garg 2007]

Key Idea: Recursively compute basis for the given BTL formula

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.
Monitoring for Temporal Logic Formula

Input: An online distributed computation \((E, \rightarrow)\)
A temporal logic formula \(\Phi\)

Open Problem 6:
Give an online algorithm to detect violation of \(\Phi\) in \((E, \rightarrow)\).

Sample Related Work:
[Fromentin, Raynal, Tomlinson, Garg ICPP 94]:
  Regular Expressions, LRDAG
[Sen, Vardhan, Agha, Rosu ICSE 04]:
  Past-Time Distributed Temporal Logic
[Ogale, Garg DISC 07]:
  Basis Temporal Logic
[Mostafa, Bonakdarpour, IPDPS 15]:
  3-valued LTL
Talk Outline

1. Online Chain Decomposition

2. Global Predicate Detection
   - Distributed Trigger Counting
   - Enumerating Consistent Global States
   - Detecting Special Classes of Predicates

3. Monitoring for Temporal Logic Formulas
   - Computation Slicing
   - Detecting temporal logic formulas

4. Controlled Reexecution
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
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 vs Choosing Events
  supervisor simply adds delay between events
  supervisor changes order of events
  supervisor chooses an event to execute
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)
Delaying events: Offline control

Problem:
Instance: Given a computation and a boolean expression $B$ of local predicates
Question: Is there a non-interfering control relation that maintains $B$?
This problem is NP-complete [Tarafdar and Garg DISC 97]
Controlled Re-execution

Add the control necessary to maintain correctness properties
e.g., mutual exclusion

Efficient algorithms for computing the synchronization for:

- Locks [Tarafdar, Garg DISC98]
  - time-complexity: \( O(nm) \)
- disjunctive predicate [Mittal, Garg PODC00]
  - e.g., \((n - 1)\)-mutual exclusion
    - time-complexity: \( O(m^2) \)
    - minimizes the number of synchronization arrows
- region predicate [Mittal, Garg PODC00]
  - e.g., virtual clocks of processes are “approximately” synchronized
Choosing Events at Runtime

Assume that the languages supports the construct or.

**Semantics:** the program is correct irrespective of which choice is made

**Examples:**

A.quicksort() or A.insertsort();
A.foo-version1(size) or A.foo-version0(size);
(item := Queue1.remove() or (item := Queue2.remove()));
Controlling Distributed Computation

Input: An online distributed computation $(E, \rightarrow)$
a desired temporal logic predicate $\Phi$

Open Problem 7: Synthesize control such that the controlled computation satisfies $\Phi$
Summary

Constructing computation
  • Online Chain Decomposition

Global Predicate Detection
  • Distributed Trigger Counting
  • Enumerating Consistent Global States
  • Online Detection of Relational Predicates

Monitoring for Temporal Logic Formulas
  • Computation Abstraction Algorithms
  • Runtime monitoring for temporal logic formulas

Efficient Control
  • Runtime control to ensure temporal logic formulas
Background Information

- Elements of Distributed Computing, Wiley & Sons 2002