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.
- Assumptions made on the environment may not hold

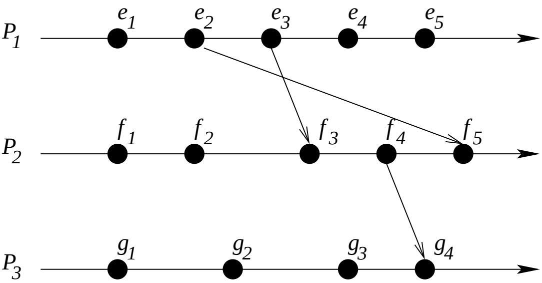
Software Quality Assurance:

- 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

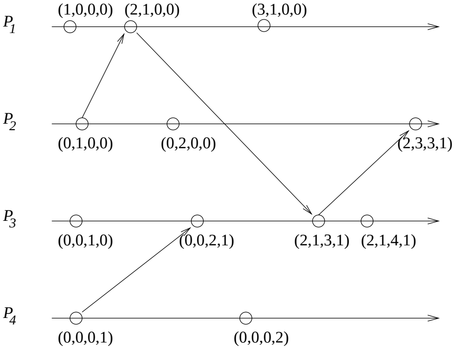
2. Global Predicate Detection
   - Distributed Trigger Counting
   - Enumerating Consistent Global States
   - Detecting Special Classes of Predicates

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]:
Every process maintains a vector v of size n, the number of processes.

Dynamic Chain Clocks

Problem with vector clocks: scalability, dynamic process structure
Idea: Computing the “chains” in an online fashion [Aggarwal and Garg 05] for relevant events

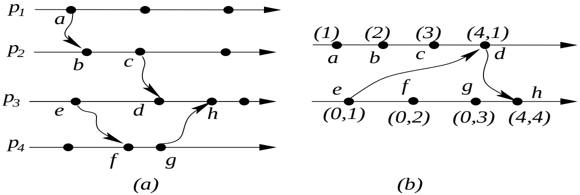
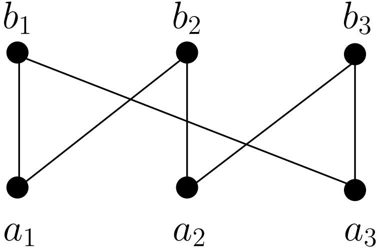


Figure : (a) A computation with 4 processes (b) The relevant subcomputation

Optimal Offline Chain Decomposition

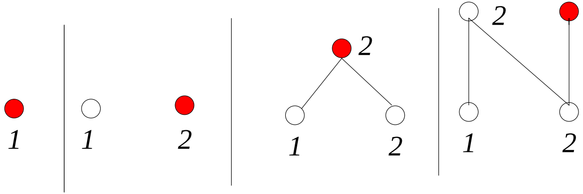
Antichain: Set of pairwise concurrent elements
Width (w): Maximum size of an antichain
Dilworth's Chain Partition Theorem [Dilworth 50]: A poset of width w can be partitioned into w chains and cannot be partitioned into fewer than w chains.



Requires knowledge of complete poset

Online Chain Decomposition for Up-growing Orders

- 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



Online Chain Decomposition for Up-growing Orders

Theorem (Felsner 97): The on-line chain partition for up-growing orders of width w requires $\Omega(w^2)$ chains in the worst case.

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

Open Problem 1: Online Chain Decomposition for Posets

Problem

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*lgw}.

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

Talk Outline

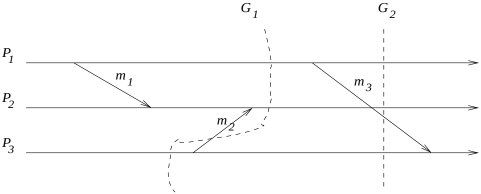
1. Online Chain Decomposition

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 (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

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

[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
Channel State: white messages received by a red process
Key idea:
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 10]

Distributed Trigger Counting (DTC) Problem

System: Completely connected topology

n processes

w triggers that arrive at these processes, $w >> n$

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 τ triggers

Use rounds with repeated halving.

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

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

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

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

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

[Chakaravarthy, Choudhury, Garg, Sabharwal 12]

LayeredRand Algorithm

$n = (2^L - 1)$ processors arranged in L layers
i^{th} layer has 2^i processors, $i = 0$ to $L - 1$.
Threshold for layer i, $\tau(i) = \lceil \hat{w}/4.2^i . \log(n+1) \rceil$

On receiving $\tau(i)$ triggers inform a random node at level $i - 1$.

Message Complexity: $O(n \log n \log w)$
Maximum Receive Load: $O(\log n \log w)$
[Chakaravarthy, Choudhury, Garg, Sabharwal 12]

DTC Algorithms

Algorithm	Message Complexity	MaxRecvLoad
Centralized	$O(n \cdot (\log n + \log w))$	$O(n \cdot (\log n + \log w))$
LayeredRand	$O(n \cdot \log n \cdot \log w)$	$O(\log n \cdot \log w)$
CoinRand	$O(n \cdot (\log w + \log n))$	$O(\log w + \log n)$
TreeFillRand	$O(n \cdot \log(w/n))$	$O(\log(w/n))$

CoinRand: [Chakaravarthy, Choudhury, Garg, Sabharwal 12] ,
TreeFillRand: [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: Distributed Trigger Counting

Problem

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

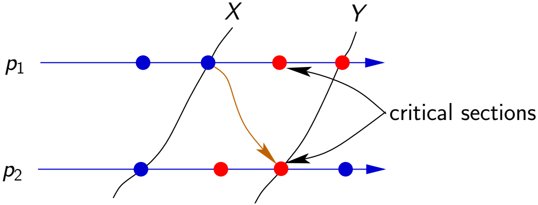
4. Controlled Reexecution

Global Predicate Detection

Predicate: A global condition B expressed using variables on processes

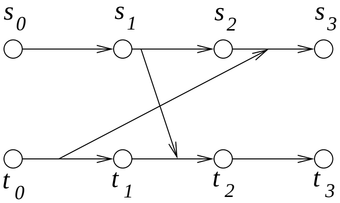
e.g., more than one process is in critical section,
there is no token in the system

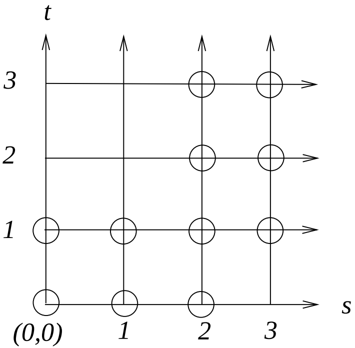
Problem: find a consistent cut that satisfies the given predicate B



The global predicate may express: a software fault or a global breakpoint

Two interpretations of predicates





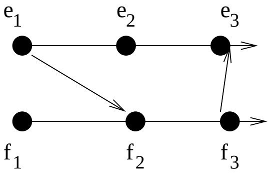
Possibly:B: exists a path from the initial state to the final state along which B is true on some state

Definitely:B : for all paths from the initial state to the final state B is true on some state

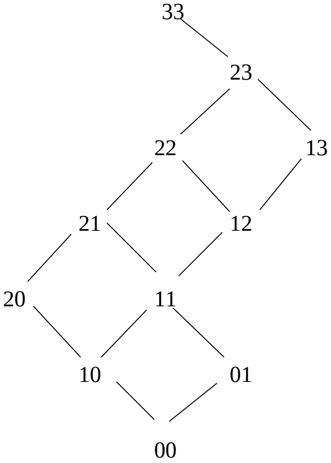
Cooper and Marzullo's Algorithm

[Cooper and Marzullo 91]
Implicit BFS Traversal
Problem:
Space Complexity: need to store a level of the lattice – exponential in the number of processes

Lexical Enumeration of Consistent Global States



(a)



(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)

Algorithm for Lex Order

```
$nextLex(G)$: next consistent global state in lexical order
    var
        $G$ : consistent global state initially $(0, 0, ..., 0)$;
    enumerate($G$);
    while $(G < \top)$
        $k :=$ smallest priority process with an event enabled in $G$
        $G := leastConsistent(succ(G, k))$
        enumerate($G$);
    endwhile ;
```

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

[Garg03]

Summary of Enumeration Algorithms

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

Algorithm	Time per Global State	Space
Implicit BFS [Cooper, Marzullo 93]	$O(n^3)$	exp. in n
Implicit DFS [Alagar, Venkatesan 01]	$O(n^3)$	$O(\|P\|)$
Gray Code [Pruesse, Ruskey93]	$O(\|P\|)$	exp. in $\|P\|$
Ideal Tree [Jegou 95, Habib 01]	$O(\Delta(P))$	$O(\|P\|)$
Lexical [Ganter, Reuter 91, Garg 03]	$O(n^2)$	$O(n)$
QuickLex [Chang, Garg 15]	$O(n \cdot \Delta(P))$	$O(n^2)$

Notation:

P: poset,

n: width of the poset,

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

Open Problem 3: CAT Enumeration of Global States

Input: A distributed computation (a poset)
Output: Enumeration of all consistent global states of the computation.

Problem

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

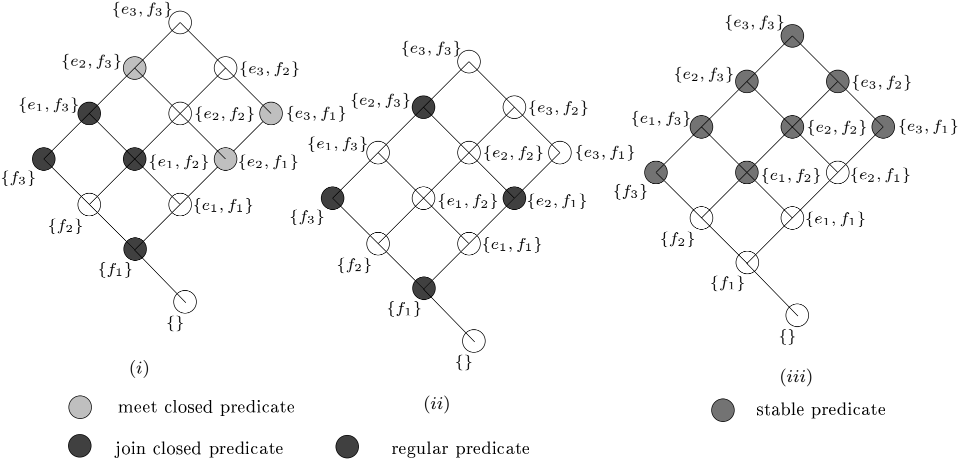
  occurs in one interleaving $\implies$ occurs in all interleavings

  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: $x_1 + x_2 + \cdots + x_n \geqslant k$ Chase and Garg 95 [Tomlinson and Garg 96]

  e.g., violation of k-mutual exclusion

Special Classes of Predicates

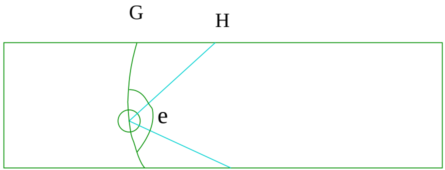


(i)

(ii)

(iii)

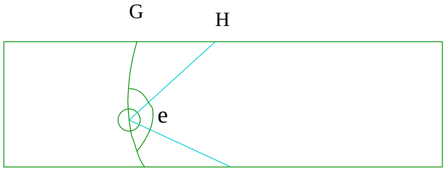
Linearity



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).

Linear Predicates A predicate B is linear if whenever B is false in G, there exists a crucial event e such that unless e is executed the predicate cannot become true.

Examples of Linear Predicates: Conjunctive Predicates



- missing primary: (P1 is secondary) and (P2 is secondary) and (P3 is secondary)
- mutual exclusion problem: (P1 in CS) and (P2 in CS)
- 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, $l_1 \wedge l_2 \wedge \ldots \wedge l_n$, where l_i is local to P_i.

Relational Predicates

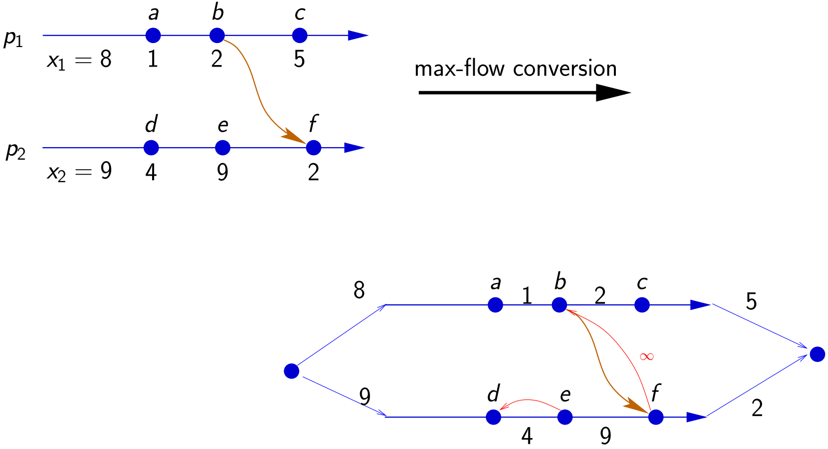
Let x_i: number of tokens at P_i

$\Sigma x_i < k$: loss of tokens

$\Sigma x_i > k$: resource allocation violation

Algorithm: max-flow technique [Groselj 93, Chase and Garg 95],

Consistent cut with minimum value = min cut in the flow graph



Open Problem 4: Efficient Distributed Online Algorithms for Relational Predicates

Input: An online computation $(E, \rightarrow)$
A relational predicate B

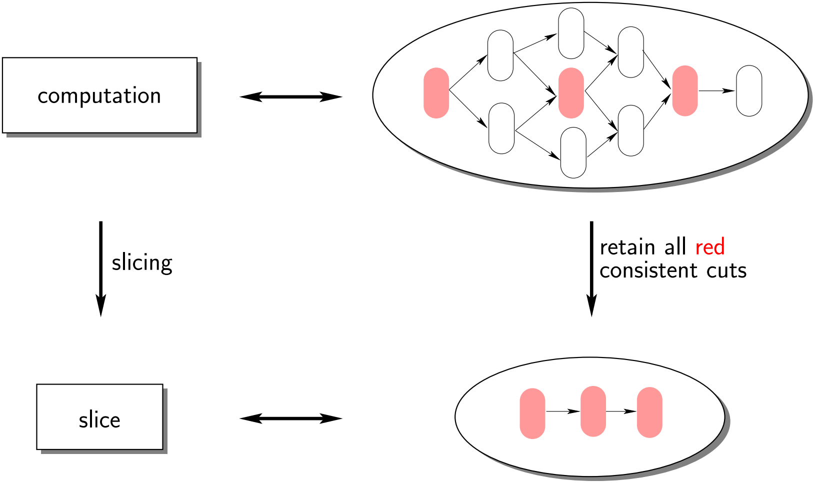
Problem

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

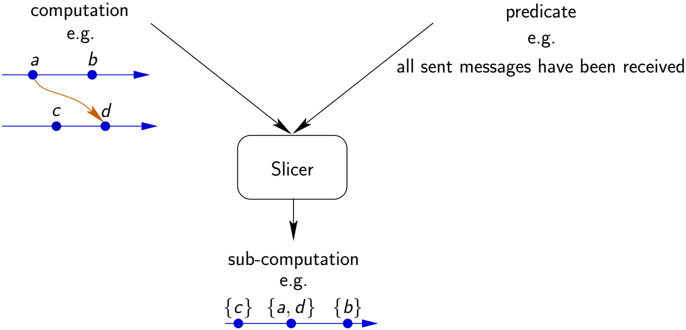
slice

retain all red consistent cuts

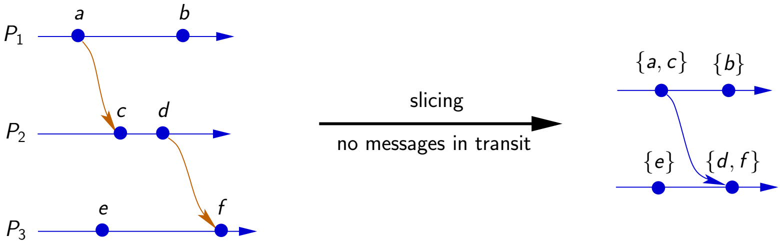
Computation Slice: Definition

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



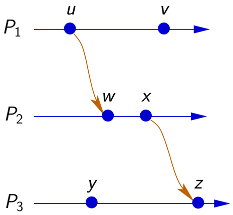
Slicing Example



slicing

no messages in transit

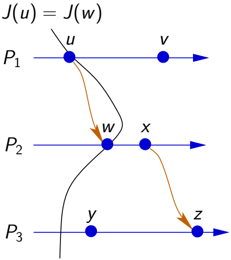
Computing the Slice for Regular Predicate



$B =$ "no messages in transit"

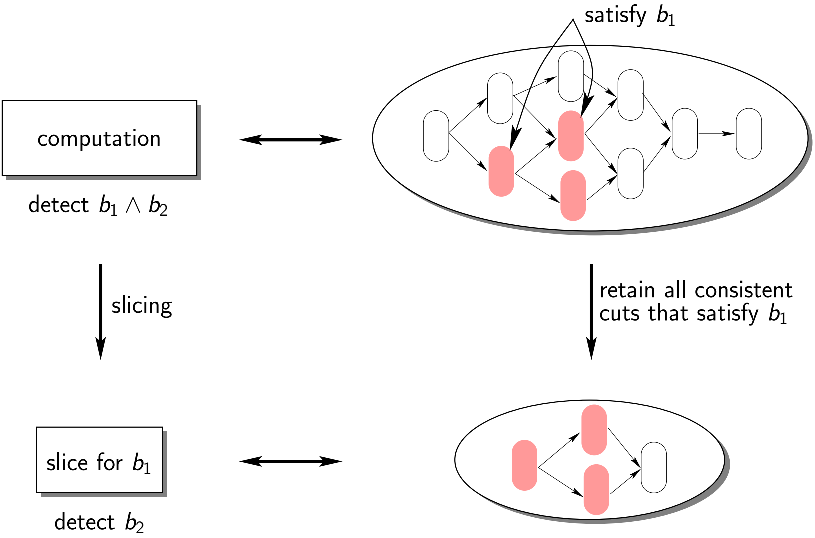
Algorithm: For every event $e \in E$, compute
$J(e)$: the least consistent cut that contains e and satisfies B.

Slicing Example



$J(u) = \{u, w\}$
$J(v) = \{u, v, w\}$
$J(w) = \{u, w\}$ (duplicate)
$J(x) = \{u, w, x, y, z\}$
$J(y) = \{y\}$
$J(z) = \{u, w, x, y, z\}$ (duplicate)

How does Computation Slicing Help?



computation

detect $b_1 \wedge b_2$

satisfy b_1

slicing

retain all consistent cuts that satisfy b_1

slice for b_1

detect b_2

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]

Open Problem 5: Efficient Computation Abstraction

Input: online distributed computation $(E, \rightarrow)$
predicate B

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

Goal: Detect temporal logic formulas with polynomial time complexity
polynomial in the size of the computation, not the size of the formula

Basis Temporal Logic (BTL)

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 $P \vee Q$, $P \wedge Q$, $\Diamond P$ and $\neg P$ are also BTL predicates

Example:$B = \neg \Diamond (\bigwedge red_i) \wedge token_0$

[Ogale and Garg, DISC 07]

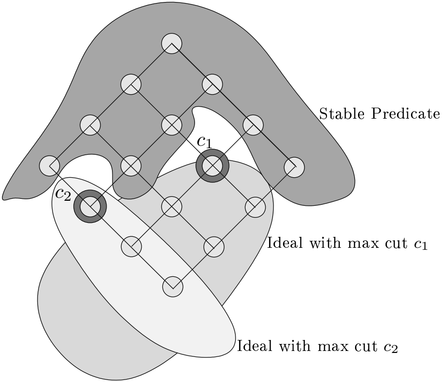
Basis of Predicate

Basis of a predicate: compact representation of all consistent global states that satisfy the predicate.

Examples

- Regular Predicate:
  Sufficient to keep the slice (or join-irreducibles) of $(E, \rightarrow)$ with respect to P
- Order Ideal Predicate: P is true in G iff $G \subseteq W$
  Sufficient to keep the largest CGS G that satisfies P

Stable Predicates



Representing stable predicates

Represent a stable predicate by a set of ideals.

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, $P = \neg\Diamond(\bigwedge red_i) \wedge token_0$.
- At least one process is beyond phase k (stable) and all the processes are red.

All regular predicates and stable predicates are semiregular.

Semiregular Structure

A semiregular structure, g, is a tuple $(\langle \mathit{slice}, \mathcal{I} \rangle)$ consisting of a slice and a stable structure, such that
the predicate is true in cuts that belong to their intersection.

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.

Open Problem 6: Monitoring for Temporal Logic Formulas

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

Problem

Give an online distributed algorithm to detect violation of Φ in $(E, \rightarrow)$.

Sample Related Work:

[Fromentin, Raynal, Tomlinson, Garg 94]:
Regular Expressions, LRDAG

[Sen, Vardhan, Agha, Rosu 04]:
Past-Time Distributed Temporal Logic

[Ogale, Garg 07]:
Basis Temporal Logic

[Mostafa, Bonakdarpour 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

- maintain global invariants or proper order of events
  Examples: Distributed Debugging

  ensure that $busy_1 \vee busy_2$ is always true

  ensure that m_1 is delivered before m_2

  maintain $\neg CS_1 \vee \neg CS_2$
- Fault tolerance

  On fault, rollback and execute under control
- Adaptive policies

  procedure A (B) better under light (heavy) load

Models for Control

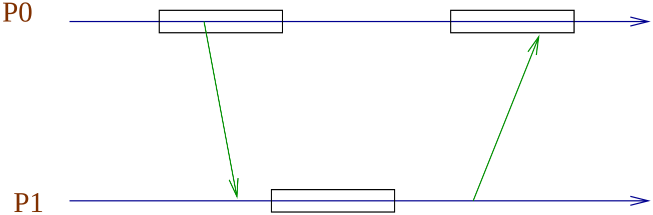
Is the future known ?

- 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 vs Choosing Events

- supervisor simply adds delay between events
- supervisor changes order of events
- supervisor chooses an event to execute

Delaying events: Offline control

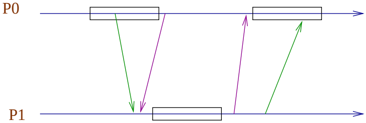


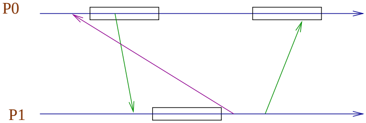
Maintain at least one of the process is not red

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





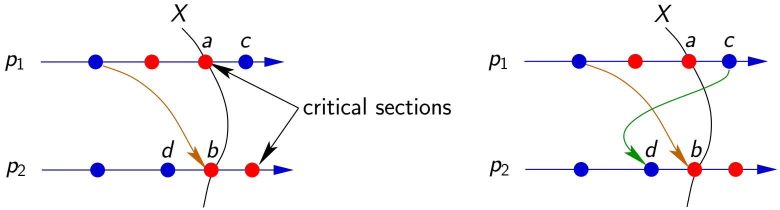
Problem:

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
Efficient algorithms for computing the synchronization for:

- Locks [Tarafdar, Garg 98]
  - *time-complexity:* $O(nm)$
- disjunctive predicate [Mittal, Garg 00]

  e.g., $(n-1)$-mutual exclusion
  - *time-complexity:* $O(m^2)$
  - minimizes the number of synchronization arrows

n: number of processes, m: number of events

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());
```

Open Problem 7: Controlling Distributed Computation

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

Problem

Synthesize control such that the controlled computation satisfies Φ

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, V. K. Garg, Wiley & Sons 2002
- Introduction to Lattice Theory with Computer Science Applications, V. K. Garg, Wiley & Sons 2015
- Modeling and Control of Logical Discrete Event Systems, R. Kumar and V. K. Garg, Springer-Verlag 1995