Observation and Control for Debugging Distributed Computations

Vijay K. Garg
Electrical and Computer Engineering Department
University of Texas at Austin,
Austin, TX 78712

http://maple.ece.utexas.edu/~vijay/

Acknowledgments

- Collaborators on various ideas C. M. Chase, E. Fromentin, R. Kilgore, R. Kumar, J. R. Mitchell, V. V. Murty, M. T. Raghunath, M. Raynal, A. Tarafdar, A. I. Tomlinson, and B. Waldecker

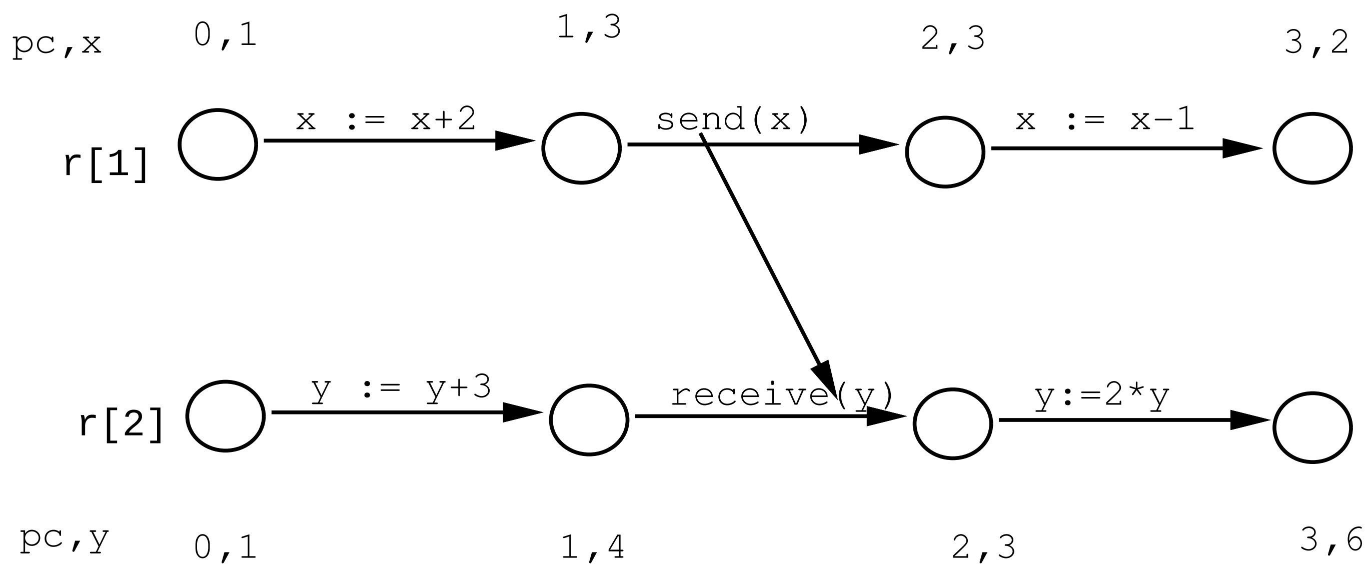
Outline of the talk

- Introduction: our model
- Observation: Main ideas
  - Lack of shared clock
  - Lack of shared memory
  - Combinatorial Explosion
- Observation: Algorithms
  - WCP algorithm, Channel predicates
  - Detecting regular expressions
- Control
  - Delaying events: offline
  - Delaying events: online
  - Controlling order: offline
  - Controlling order: online

Characteristics of Distributed Systems

- Lack of shared clock
  - order of events partial
- Lack of shared memory
  - meaning of global state
  - need messages for observing "global state"
- Multiple processes
  - Combinatorial explosion
  - non-determinism

Model of a Distributed Program



- messages: asynchronous, reliable, no FIFO assumption
- no shared clock or memory
- local states
- Lamport's causally precede relation, concurrency relation

Motivation for Observation

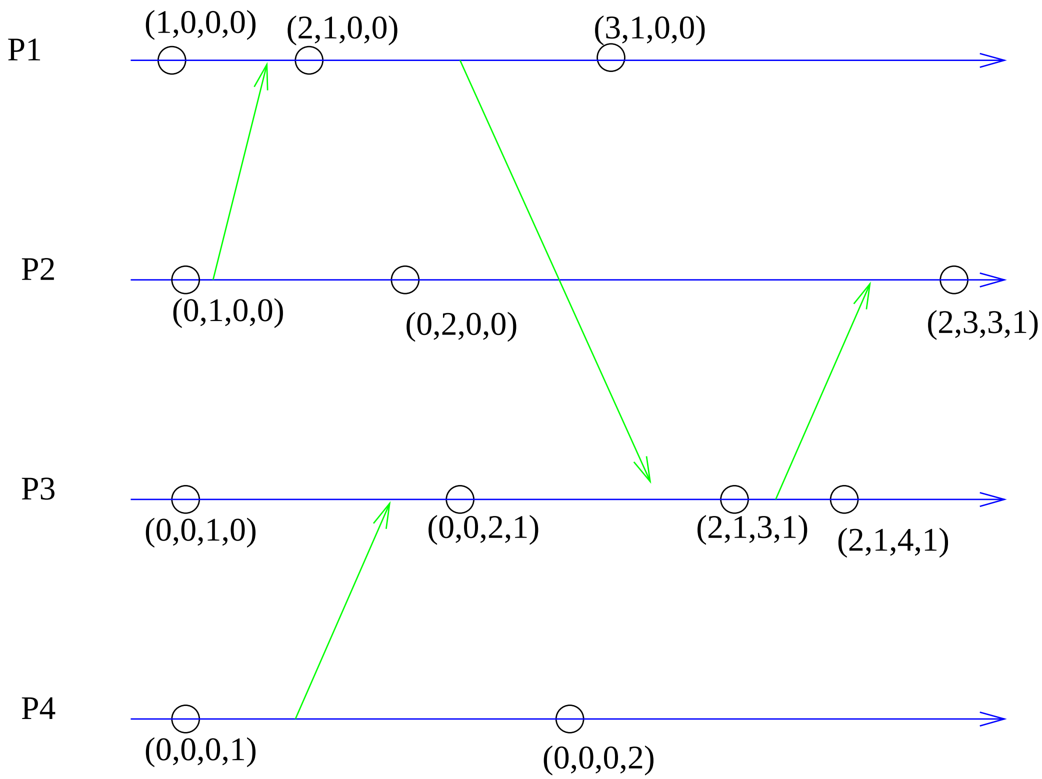
Dear Watson, you see but you do not observe...

- Distributed Debugging, Testing
  - stop when the predicate q is true
    - predicate q = (P1 is in critical section) and (P2 is in critical section).
  - Detect if the program violates any invariant
- Fault-tolerance
  - Monitoring while the program is operational
- Distributed Active Rules
  - On global condition p, trigger rule a
- General paradigm for observing Distributed Algorithms
  - Termination detection, deadlock detection, loss of token

Lack of shared clock

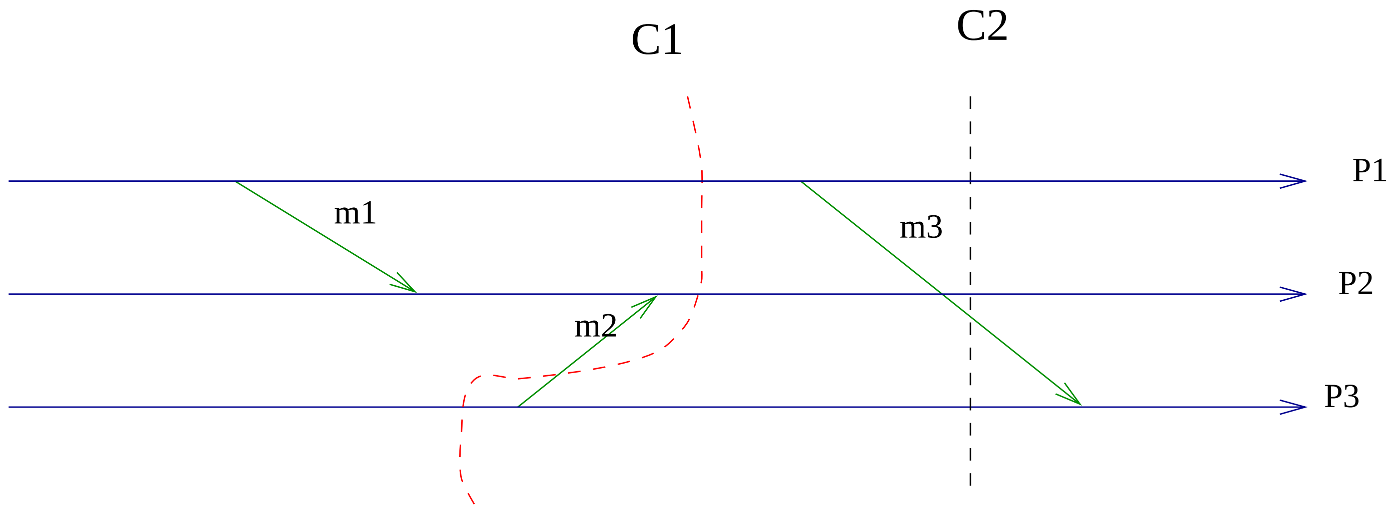
- Problem: define truthness of the predicate $CS_1 \wedge CS_2$
  - based on real time
  - based on causality
- Real-time considered harmful in distributed system.
  - My clock synchronization algorithm achieves 10 ms
  - programs should work independent of processor speeds
- Reject linear time, accept vector time
  - Lamport 78, Fidge 89, Mattern 89
  - Simultaneity vs Concurrency

Clock in a Distributed System



- Property: $s \to t$ iff $s.v < t.v$.

Lack of shared state

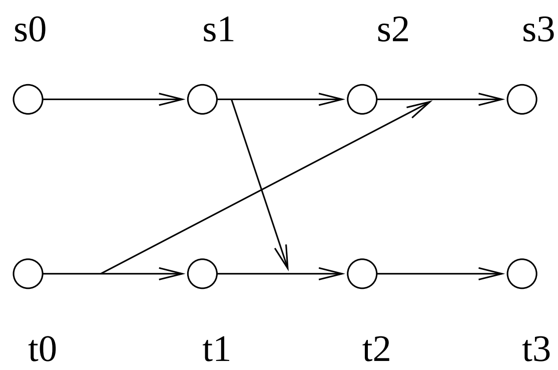


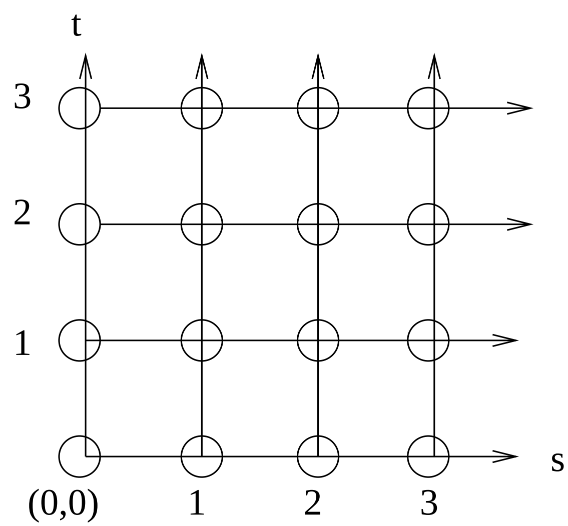
- consistent global state
  - if the receive of an event is recorded, then send must be recorded

Camera: Chandy and Lamport's Algorithm

- Algorithm to compute a snapshot of a computation: S_*
  - S_* is a possible global state in the computation
- Stable predicate: once true stays true
  - e.g. termination detection, deadlock detection
- To monitor stable predicates: repeatedly take the snapshots
- Disadvantages of CL Algorithm for predicate detection
  - Not useful for unstable predicates
  - Does not return the first cut
  - How often should the snapshot be taken ?
  - Assumes FIFO

Unstable Predicates





- Multiple timed executions consistent with one run

Two interpretations of predicates

- Two modalities: [Cooper and Marzullo 91], [Garg and Waldecker 91]
  - Possibly:q (also called weak predicates)
    - exists a path from the initial state to the final state along which q is true on some state
  - Definitely:q (also called strong predicates)
    - for all paths from the initial state to the final state q is true on some state

Communication Complexity

- Consider evaluation of the predicate $q(x_1, x_2)$
  - only P_1 knows all the values taken by x_1
  - only P_2 knows values taken by x_2
  - Is $q(x_1, x_2)$ true for some value of x_1 and x_2
- Key question: number of values that need to be communicated
  - one value per internal event, or
  - one value per external event

Monotonicity

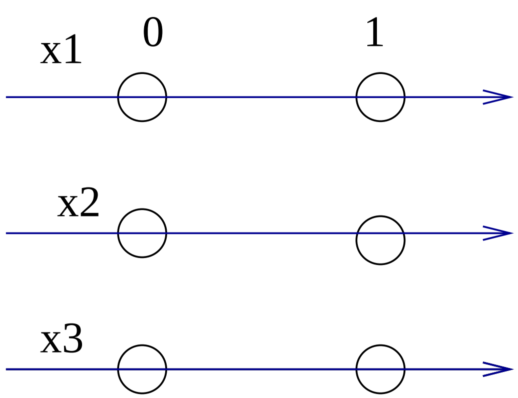
- Definition
  - Assume x_1 takes values from a totally ordered set
  - q is monotone w.r.t. first argument if

$$\forall a, b, x_2 : (a < b) \Rightarrow (q(a, x_2) \Rightarrow q(b, x_2))$$

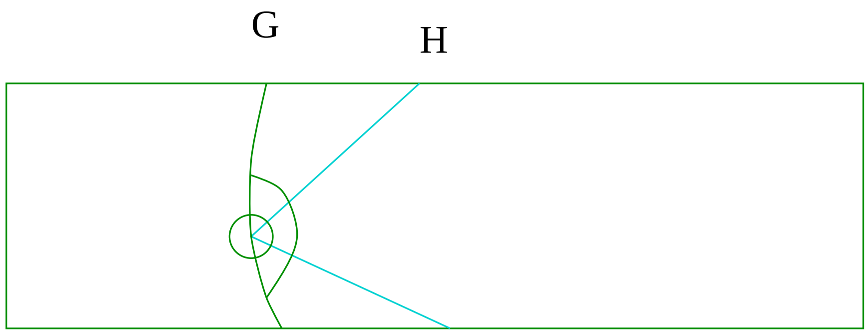
- Examples
  - $q = (x_1 > x_2)$: monotonic w.r.t x_1 and x_2
  - $q = l_1 \wedge l_2$: monotonic
  - $q = (x_1 = x_2)$: not monotonic.

Multiple Processes

- Intractability of the Global Predicate Detection Problem
  - **Given**: an execution S of N processes, N variables $x_1, \ldots, x_N$, and a predicate q defined on x.
  - Is there a consistent cut $G \in S$ such that $q(G)$ is true.
- Theorem [Chase and Garg 95]: The predicate detection problem is NP-Complete.
  - Proof: By reduction from SAT $((x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2) \wedge \ldots)$



Linearity



- Forbidden predicate: forbidden(G,i) iff
  - $\forall H : G \leq H : (G[i] = H[i]) \Rightarrow \neg q(H)$
- Predicate q is linear w.r.t. a computation S if
  - $\forall G : \neg q(G) \Rightarrow \exists i : \text{forbidden}(G, i)$
- Examples
  - $l_1 \wedge l_2 \wedge \ldots \wedge l_n$
  - $x + y \geq k$, x is non-increasing
  - channel is empty

Summary of Observation: Problems and Solutions

•

Characteristic	Problem	Idea	Bonus
No shared clock	ordering events	causality	avoid race errors
No shared memory	message/state change	monotonicity	extremal functior
multiple processes	combinatorial explosion	linearity	first cut

Cooper and Marzullo's Algorithm

- Possibly:p
  - construct the lattice of global states, check each global state for truthness of p
- Definitely:p
  - for all paths from the initial state to the final state p is true on some state
  - construct the lattice of global states
  - remove states satisfying p
  - Is last state reachable from the initial state
- Complexity: $O(k^n)$ where
  - k: Number of local states per process
  - n: Number of processes

Weak Conjunctive Predicates

- WCP $\equiv$ Possibly:$l_1 \wedge l_2 \wedge \ldots \wedge l_n$
- useful for bad or undesirable predicates
  - Example: the classical mutual exclusion problem.
  - Example: (John is sleeping) and (Mary is sleeping) and (Robert is sleeping)
- detect errors that may be hidden in some run due to race conditions.

Importance of Weak Conjunctive Predicates

- Sufficient for detection of any boolean expression
  - which can be expressed as a disjunction of a small number of conjunctions.
  - Example x, y and z are in three different processes. Then,

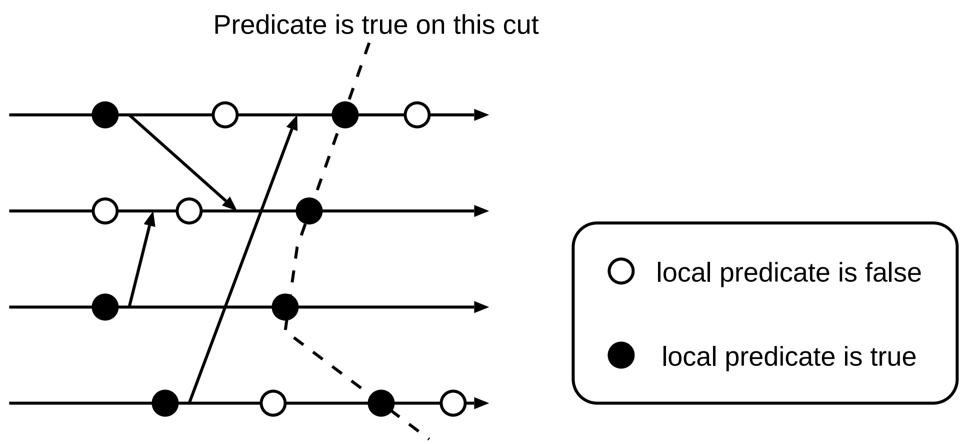
$$even(x) \wedge ((y < 0) \vee (z > 6))$$

$$\equiv$$

$$(even(x) \wedge (y < 0)) \vee (even(x) \wedge (z > 6))$$

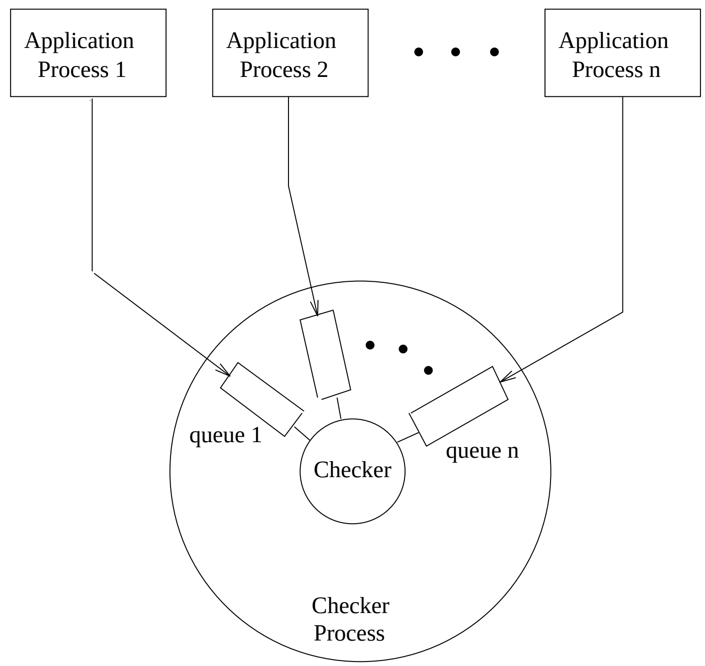
- the global predicate is satisfied by only a finite number of possible global states.
  - Example, x and y are in different processes.
    - $(x = y)$ is not a *local* predicate

Conditions for Weak Conjunctive Predicates



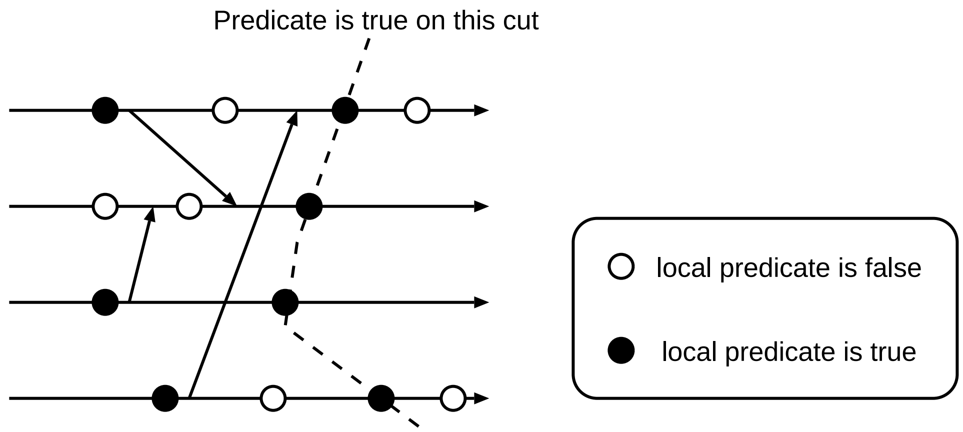
- Possibly $(l_1 \wedge l_2 \wedge \ldots l_n)$ is true **iff** there exist s_i in P_i such that l_i is true in state s_i, and s_i and s_j are incomparable for distinct i, j.
- Key problems and solutions
  - number of states satisfying local predicates may be large: Use monotonicity (at most one state per message)
  - combinatorial explosion when combining them together: Use Linearity

Weak Conjunctive Predicates: Centralized Algorithm



- Each non-checker process maintains its local *vector*
  - send to the checker process the vector clock whenever
    - local predicate is true
    - at most once in each message interval.
  - Optimization: Sufficient to send the vector once after each message is sent

Checker Process



- Steps
  - Begin with the initial global state
  - Eliminate any state that happened before any other state along the current cut.
- Predicate true for the first time
  - if no states can be eliminated.
- Predicate false
  - if we eliminate the final state from any process

Overhead: Non-checker processes

- Space complexity
  - the array $vector$: $O(n)$.
- message complexity
  - $O(m_s)$ where m_s is the number of program messages sent.
  - In addition, program messages have to include time vectors.
- Time complexity
  - detection of local predicates
  - maintain vector clock $(O(n)$/message).

Overhead: Checker processes

- Space complexity
  - n queues, each containing at most m vectors
- Time complexity
  - The algorithm for checker requires at most $O(n^2m)$ 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.

[Garg and Waldecker 94]

Disadvantages of above algorithm

- Centralized
  - Checker process may become a bottleneck
- Space requirements
  - Queues at the checker process may grow large
- Message complexity
  - may result in too many additional messages

Other WCP algorithms

- token based algorithm [Garg and Chase 95]
  - eliminate centralized checker process
- Completely distributed algorithm [Garg and Chase 95]
  - Uses Scholten and Dijkstra's termination detection
- Distributed Offline-algorithm [Venkatesan and Dathan 92]
  - assume FIFO and off-line
- 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

Channel Predicates: Observing hallways

- Many properties require channels
  - termination detection: all processes are idle and all channels are empty

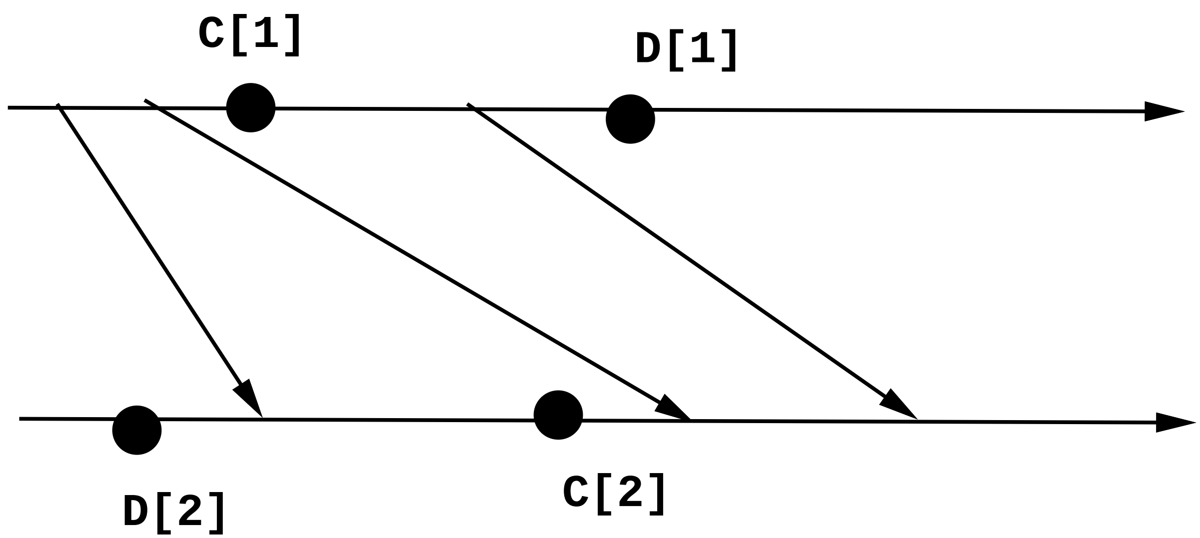
- A channel predicate: a boolean function on the state of the channel
  - uni-directional
  - memoryless. i.e. channel state = sequence of messages sent - set of messages received
  - Linearity: Given any channel state in which the predicate is false, then
    - cannot be made true by sending more messages without receiving any messages, or
    - cannot be made true by receiving more messages without sending any messages.

Linear Channel Predicates

- Empty channels
  - If false, then it cannot be made true by sending more messages,
- Channel has exactly three red messages
  - If less than three, then it cannot be made true by receiving more messages,
  - If more than three, then it cannot be made true by sending more messages,

Non-linear Channel Predicates

- Channel has an odd number of messages



- Key result: linearity = first cut is well defined.

Relational Predicates

- k tokens corresponding to k resources in the system
  - x_i: number of token at P_i
  - $\Sigma x_i < k$: loss of tokens
  - $\Sigma x_i > k$: License violation problem
- Predicate, global function
  - $\exists G$: consistent(G) : $\Sigma_{s_i \in G} s_i.x_i < K$
  - $\min G$: consistent(G) : $\Sigma_{s_i \in G} s_i.x_i$
  - Ideas:
    - max-flow technique: [Chase and Garg 95]
    - Matrix clocks: detect predicate of the form $x_1 + x_2 < k$ [Tomlinson and Garg 93]
    - Use Dilworth's theorem: [Tomlinson and Garg 96]

Other Algorithms

- Conjunction of global predicates
  - Example: $(x_1 = x_2) \wedge (x_3 > x_4)$

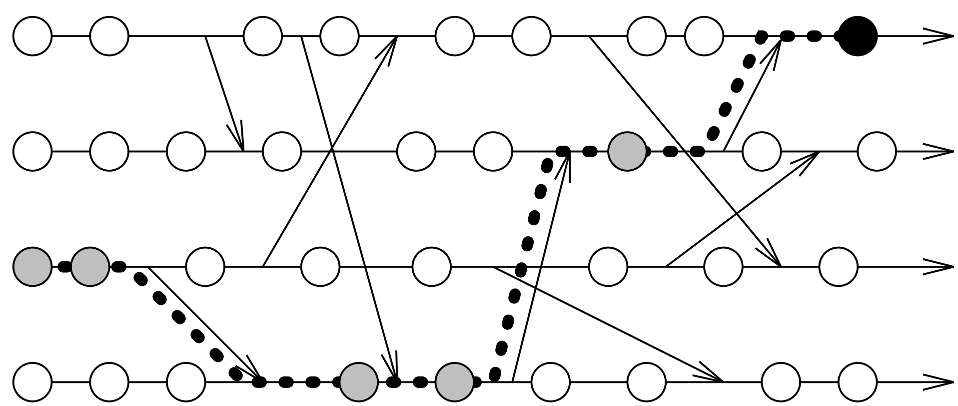
  Stoller and Scneider 95, Garg and Mitchell 96
- Notion of fixed set [Stoller and Scneider 95]
  - set of variables such that on fixing them we get a WCP
  - fix $x_1 = 4$ and $x_4 = 6$, we get $(4 = x_2) \wedge (x_3 > 6)$
  - evaluate all WCP obtained by using all values of fixed-set.
- Definitely True predicates
  - strong conjunctive predicates [Garg and Waldecker 93]

Causal Predicates

- Predicate based on control flow
  - useful for expressing and observing the flow of information.
- Early work
  - sequence of local predicates [Miller and Choi 88]
    - $l_1 \rightarrow l_2 \rightarrow \ldots l_m$.
- regular expression of local predicates [Fromentin, Raynal, Garg, Tomlinson 94]

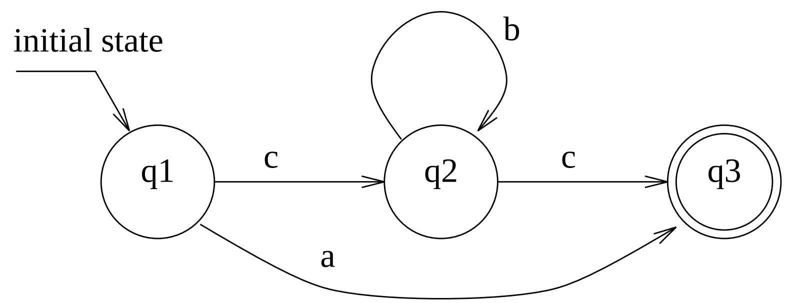
Detection of Regular Expression

- Example of a regular expression ?
  - $a + cb^*c$
- a regular expression is true in a run iff there exists a path in the run (poset) which matches the expression
- Complexity of problem
  - Many states
  - Many paths per state
  - Many strings per path



Algorithm

- Regular expression: $a + cb^*c$
- convert it to *non-deterministic* finite state machine (fsm)
- simulate it during the execution (piggybacking state of the fsm)
  - keep z[1..m] with each process
  - z[i] $=$ 1 iff there exists a causal path that takes the fsm to state i.



- Define one bit for each state

$$
\begin{aligned}
z_1 &:= init \\
z_2 &:= (c \wedge \diamond z_1) \vee (b \wedge \diamond z_2) \\
z_3 &:= (a \wedge \diamond z_1) \vee (c \wedge \diamond z_2)
\end{aligned}
$$

Other Approaches

- DAG patterns of local predicates [Garg, Tomlinson, Fromentin, Raynal 95]
- Atomic Sequences [Hurfin, Plouzeau, Raynal 93]
  - $l_i[r_i]l_{i+1}$
  - r_i does not occur between l_i and l_{i+1}
- Dynamic Properties [Babaoglu and Raynal 95]
  - Generalization of atomic sequences
- Event Normal Form [Chiou and Korfhage 94]
  - sequences of conjunctive predicates
- Recursive Poset Logic [Tomlinson and Garg 95]
  - Recursive combination of sequencing, conjunction, and linear predicates

Motivation for Control

Who controls the past controls the future, who controls the present controls the past...

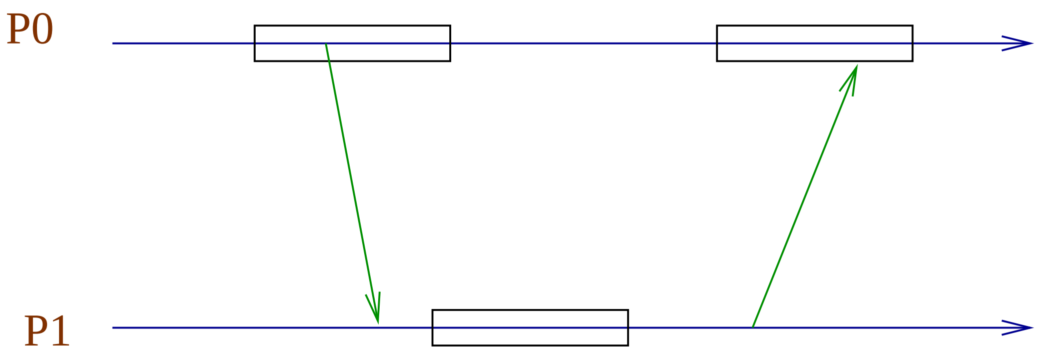
George Orwell, Nineteen Eighty-Four.

- maintain global invariants or proper order of events
- Examples: Distributed Debugging
  - ensure that $busy_1 \vee busy_2$ is always true
  - ensure that m_1 is delivered before m_2
- Resource Allocation
  - 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

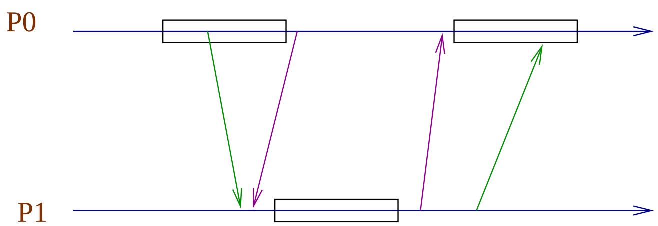
- Is the future known ?
  - Yes: offline control
    - applications in distributed debugging, recovery, fault tolerance..
  - No: online control
    - applications: global synchronization, resource allocation
- Delaying events vs Changing order of events
  - supervisor simply adds delay between events
  - supervisor changes order of events

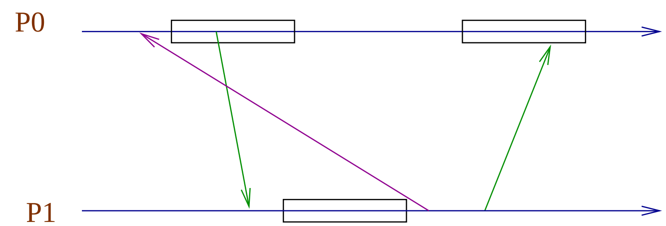
Delaying events: Offline control



- Maintain at least one of the process is not red
- Can add additional arrows in the diagram
- the control relation should not interfere with existing causality relation
  - otherwise, the system deadlocks

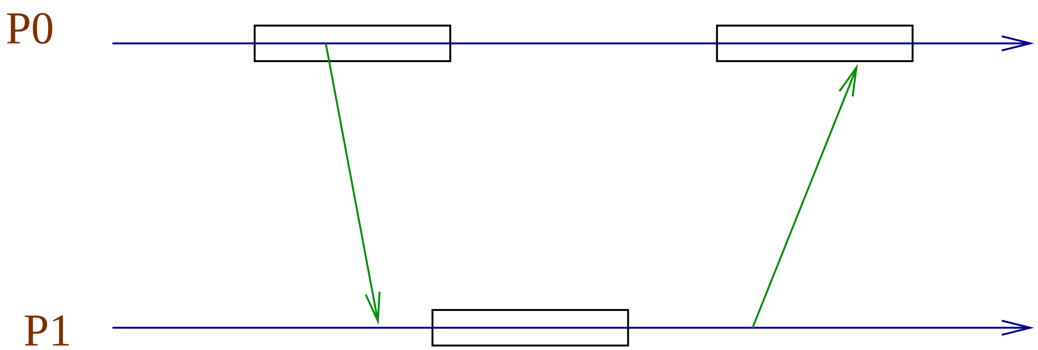
Delaying events: Offline control





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

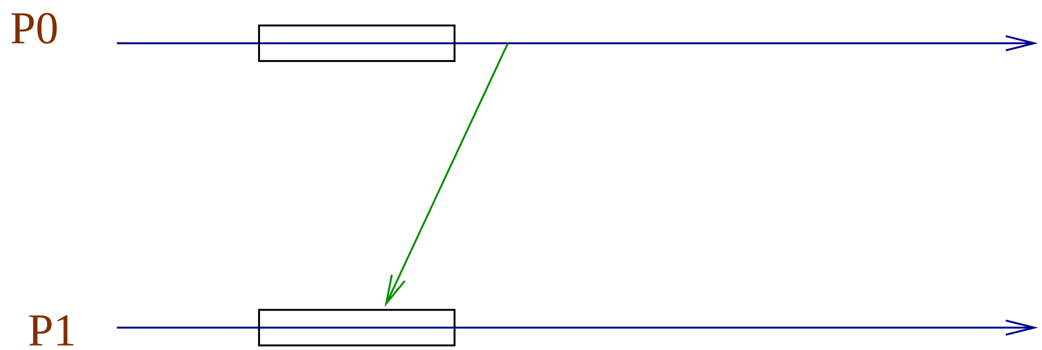
Delaying events: disjunctive predicates



- 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
    - $overlap(I_1, I_2) = (I_1.lo \rightarrow I_2.hi) \wedge (I_2.lo \rightarrow I_1.hi)$
  - Result: There exists an $O(n^2 m)$ algorithm to determine the strategy
    - n = number of processes
    - m = number of states per process

Delaying events: Online control

- Only the past is known
  - deadlock is impossible to avoid



- Assume: a process cannot block when its local predicate is false
  - maintaining $l_1 \vee l_2 \vee ... \vee l_n$ is equivalent to $n-1$ mutual exclusion problem
    - in CS = local predicate false
  - i.e., all n processes cannot be in the CS
  - can be solved using token which is a liability rather than privilege

Controlling order: Offline control

- Problem: Given a computation enforce an order of messages in a repeated run
  - Same order
    - Replay of distributed execution (distributed debugging)
    - need to store messages or message order
  - Different order
    - Testing of a distributed program [Kilgore, Chase 97]
    - Recovery of a distributed program
    - can change the order of two independent messages
    - the computation may change after first reorder

Controlling order: Online control

- Simple example: FIFO ordering of messages
- External events:
  - invocation of a message
  - send of a message
  - receive of a message
  - delivery of a message
- constraints on supervisor
  - invocation and receive events are uncontrollable
  - liveness requirement
    - if only events possible are send and delivery then at least one must be enabled.

Limitations of Online Supervision

- Specification: set of computation possible with a fixed set of messages
  - Question: Is there a control strategy to meet the specification ?
- Assumption: Supervisor can send control messages and tag user messages
  - Control possible iff specs include all synchronously order computations [Murty and Garg 97]
- Assumption: Supervisor can only tag user messages
  - control possible iff specs include all causally ordered computations [Murty and Garg 97]

Online supervision: Algorithms

- Forbidden predicate [Murty and Garg 97]
  - sub-structure that is not allowed in the computation
  - Example 1: Causal ordering
    - $\exists x, y : (x.s \rightarrow y.s) \wedge (y.r \rightarrow x.r)$
  - Example 2: Local forward flush channels
    - $(process(x.s) = process(y.s)) \wedge (process(x.r) = process(y.r) \wedge (color(x) = red) \wedge (x.s \rightarrow y.s) \wedge (y.r \rightarrow x.r)$
- There exists an algorithm with
  - input: a forbidden predicate
  - output: either "not possible", or a protocol to meet specs

Applications to Distributed Debugging

- Additional command
  - do *action* when *condition*
  - Also assume *run* and *rerun*
- Conditions
  - boolean predicate on the global state
    - requirement of (semi)-linearity
  - regular expression
- Actions
  - stop pids
  - print expressions
  - maintain boolean predicate
  - maintain order-expression

Summary

- Observation
  - Use causality instead of time to define "and"
  - Use monotonicity to reduce communication complexity
  - Global observation is quite efficient for many practical cases
    - linearity for boolean predicates
    - regular expressions of local predicates
- Control
  - desirable for many applications
  - offline vs online has implications on limitations
  - delay vs change of order model

Future Work

- Predicate detection under faulty environment
  - processes, channels or messages may fail
  - messages from different incarnations
- More complex model of control
  - plant variables vs control variables
  - unobservable events, uncontrollable events