Mid-term Review:

The Predicate Control Problem

Ashis Tarafdar

Supervising Professor: Vijay K. Garg

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

April 21, 1999

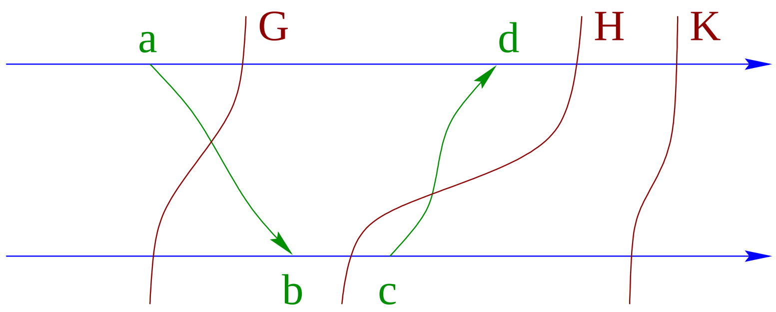
Research Summary

- Pre-Proposal Work:
  - Predicate Control: Disjunctive Predicates [Tarafdar and Garg 98a]
  - False Causality: Predicate Detection [Tarafdar and Garg 98b]
  - False Causality: General [Tarafdar and Garg 98c]
- Post-Proposal Work:
  - Predicate Control: Mutual Exclusion [Tarafdar and Garg 99a]
  - False Causality: Optimistic Recovery [Damani, Tarafdar and Garg 99b]
- Future Work:
  - Predicate Control: Other Predicates
  - Predicate Control: Software Fault Tolerance

Talk Outline: The Predicate Control Problem

- Model and Problem Statement
- Results for Disjunctive Predicates
- Application to Software Fault-Tolerance

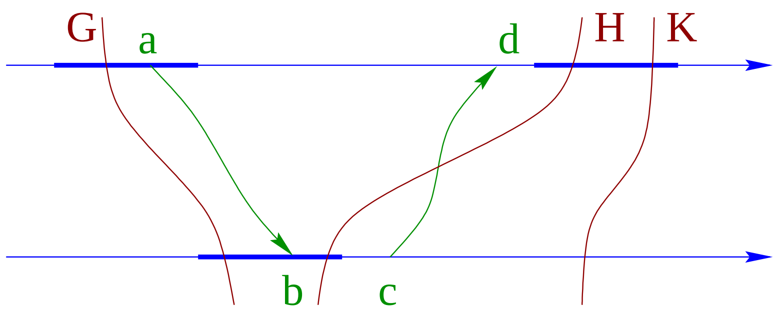
Model: Computations



state, global state, computation ($a \rightarrow c$)
G and K are consistent global states
H is an inconsistent global state

Assumptions: asynchronous, reliable message-passing

Model: Global Predicates



global predicate: boolean function on a global state

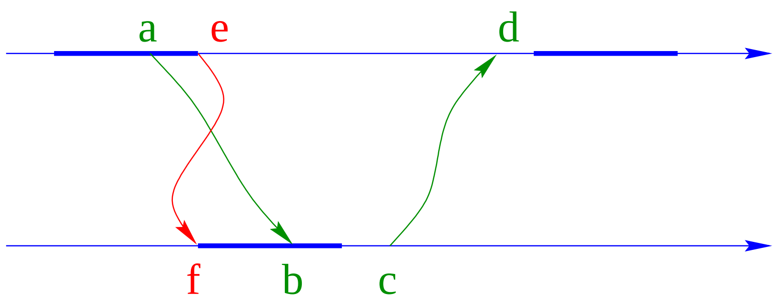
Example: mutual exclusion

$B = \neg(critical_1 \wedge critical_2)$

K satisfies B

G and H do not satisfy B

Model: Controlling Computations



$\rightarrow^c$ is a controlling computation of B in $\rightarrow$, if:

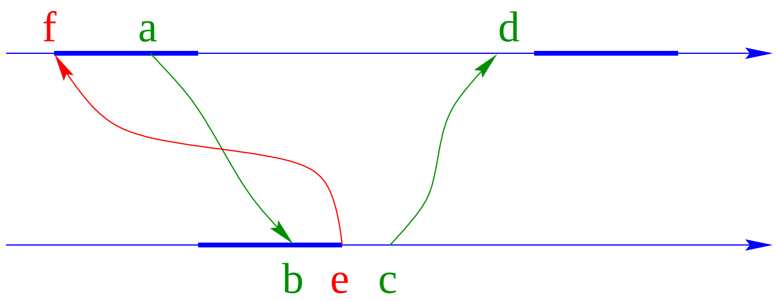
(1) $\rightarrow^c$ is stricter than $\rightarrow$, and
(2) all consistent global states in $\rightarrow^c$ satisfy B

Problem Statement

The Predicate Control Problem:

> Given a computation $\rightarrow$ and a global predicate B,
> find a controlling computation of B in $\rightarrow$

Why is it difficult?



- Cycles must be avoided. (e.g. $a - b - e - f$)
- General Predicates: NP-Hard [Tarafdar and Garg 98a]

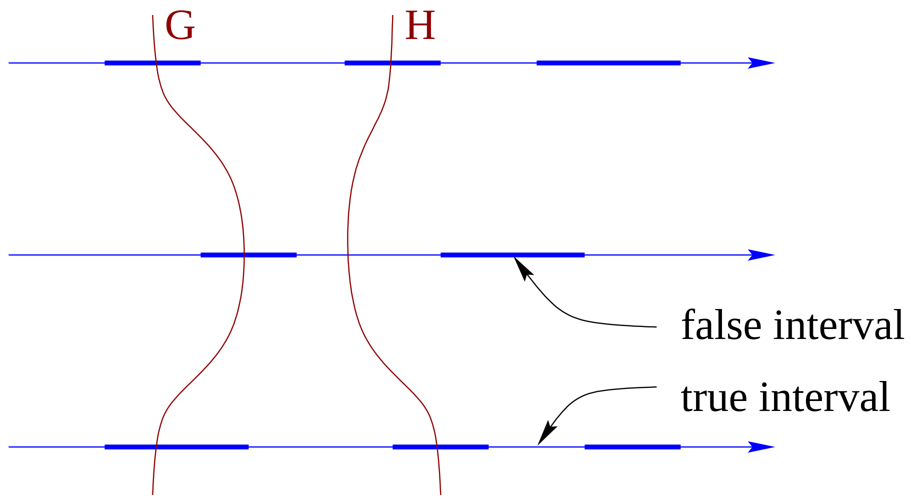
Applications

Trace-and-replay applications:

- Active debugging of concurrent programs [Tarafdar and Garg 98a]
- Software fault-tolerance of concurrent programs [Tarafdar and Garg 99a]

Disjunctive Predicates

$$B = l_1 \vee l_2 \vee \cdots \vee l_n$$



true intervals and false intervals

H satisfies B, and G does not satisfy B

Disjunctive Predicates: Examples

- At least one philosopher is thinking:

$$think_1 \lor think_2 \lor \cdots \lor think_n$$

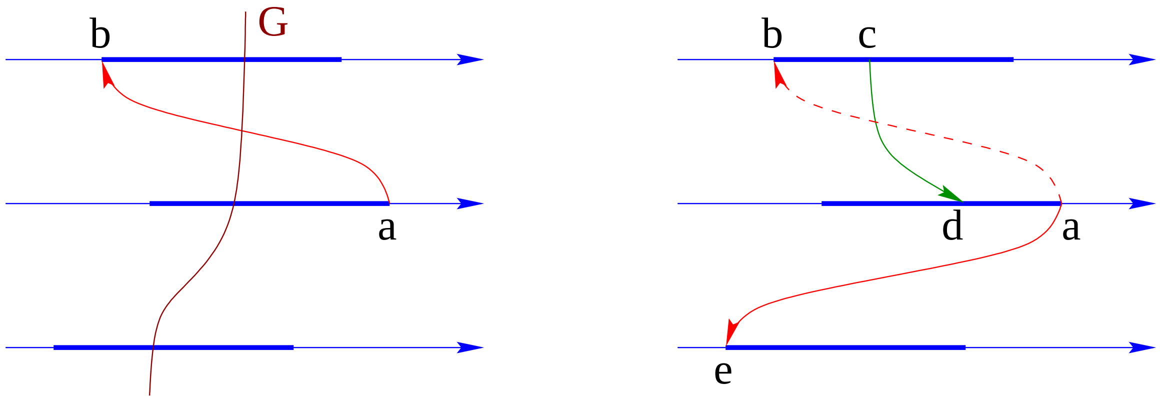
- At least one server is available:

$$avail_1 \lor avail_2 \lor \cdots \lor avail_n$$

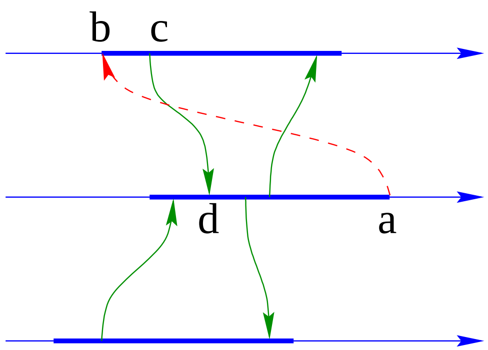
- Two-process mutual exclusion:

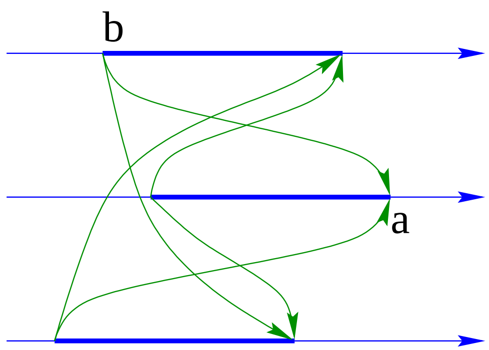
$$\neg critical_1 \lor \neg critical_2$$

Adding Synchronizations



No Controlling Computations





sometimes no controlling computation exists!

n overlapping false intervals

Conditions for Existence of Controlling Computation

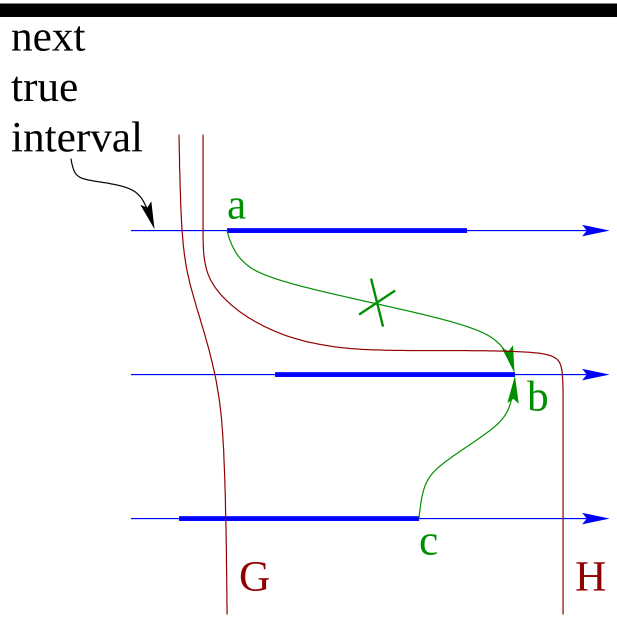
Necessary Condition:

a controlling computation exists $\Rightarrow$ no n overlapping false intervals

Sufficient Condition:

no n overlapping false intervals $\Rightarrow$ a controlling computation exists

Algorithm: Key Idea



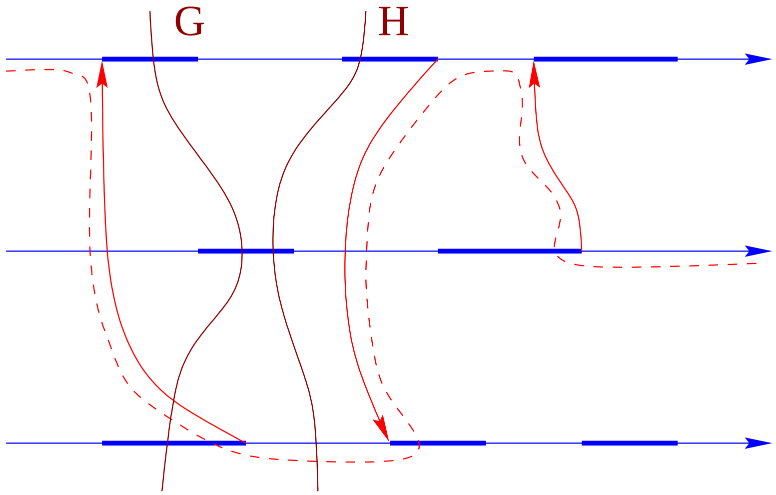
In each iteration, advance the global state so that:

it advances across at least one false interval,

while staying fixed on one true interval

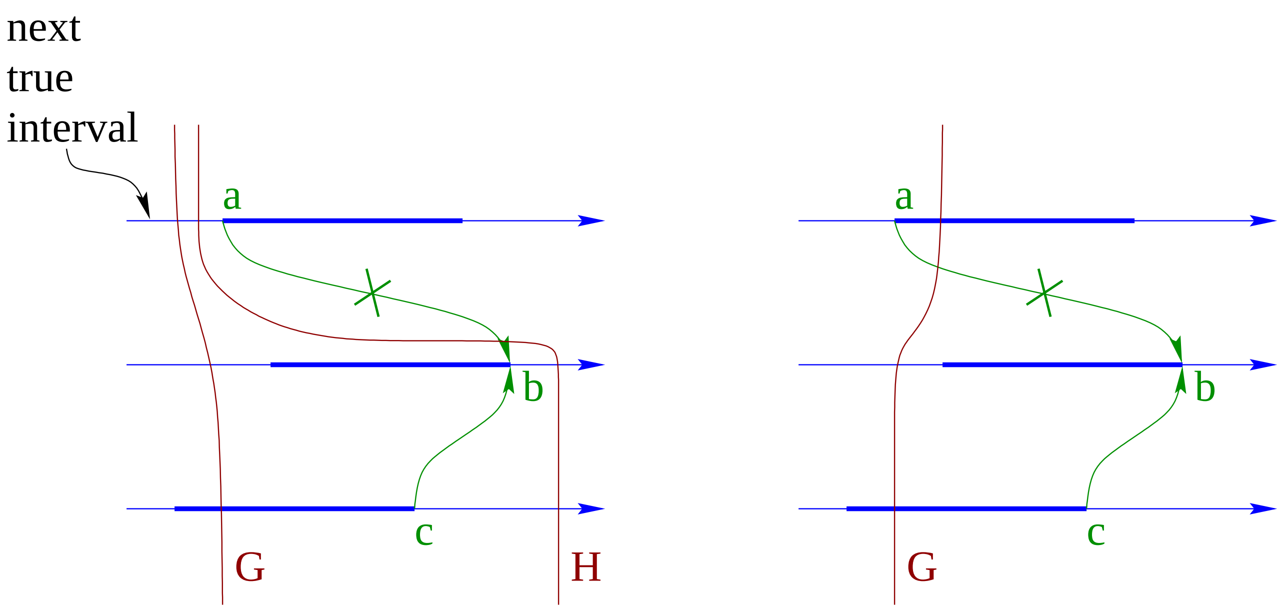
Store the fixed true intervals in a sequence

Algorithm: Key Idea



Add synchronizations to link the true intervals in a chain
In the controlling computation, G is inconsistent, H satisfies B

Algorithm: Complication



What if the global state has advanced beyond the required true interval?

Algorithm: Analysis

Time complexity:

$$O(mn)$$

Added Synchronizations:

$$O(m)$$

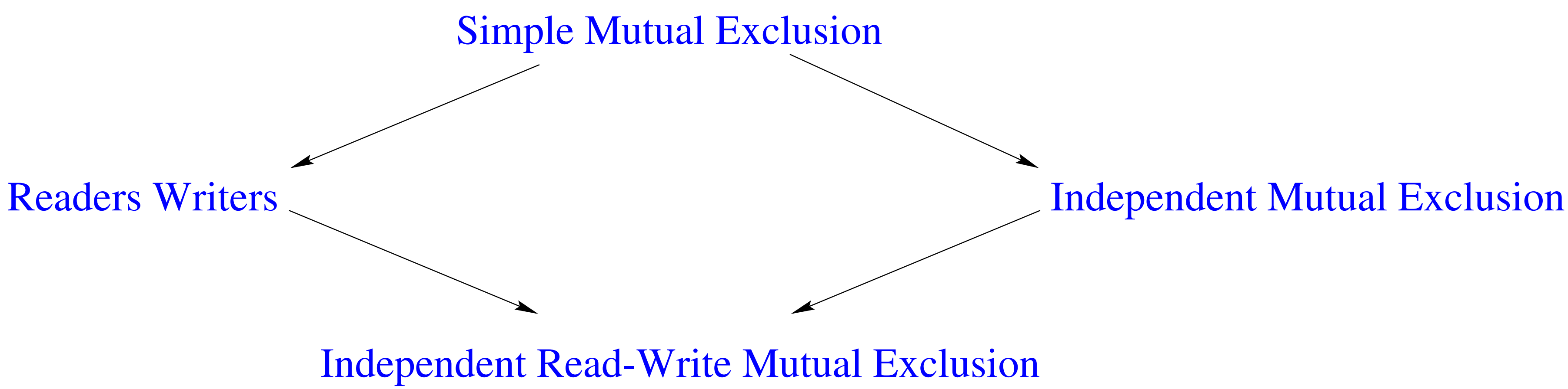
where:

m is the total number of false intervals, and

n is the number of processes

Other Results in Off-line Predicate Control

Solving off-line predicate control for mutual exclusion predicates:
[Tarafdar and Garg 99a]



$O(mn)$ algorithm, where m is the number of critical sections

Software Fault Tolerance: Background

- Earlier, it was thought that software failures are permanent
  $\Rightarrow$ design diversity approaches [Ran 75, AC 77]

- Recently, it was discovered that many software failures are transient
  $\Rightarrow$ rollback approaches [HK 93, WHF 97]

- In concurrent programs, synchronization failures (e.g races) form a large class of transient software failures [IL 95]

- Existing rollback approaches depend on chance to recover from transient failures

Tolerating Races Using Controlled Re-execution

A race is a violation of mutual exclusion

Our Approach:

1. Trace the execution

2. Detect a race

3. Find a controlling computation (predicate control)

4. Re-execute under control

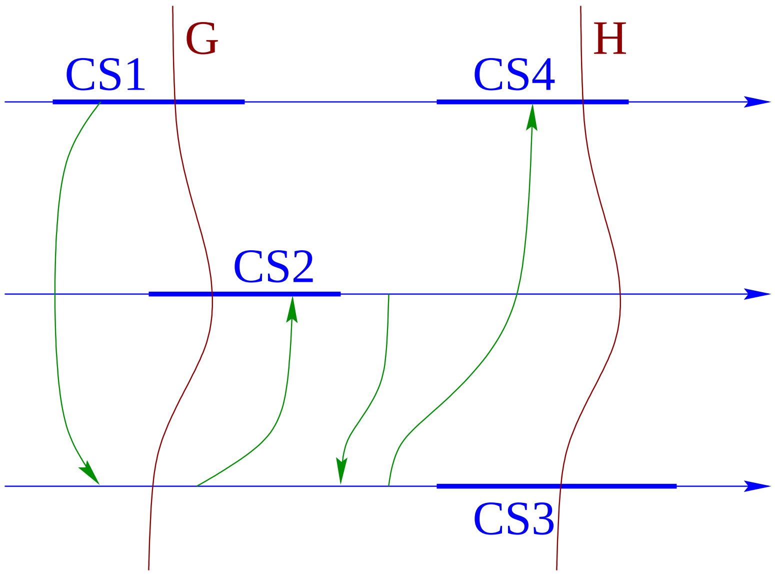
Example Scenario

- distributed processes communicate only using messages in MPI
- processes write (append) to a file in NFS
- only one process must write at a time, otherwise the file is corrupted

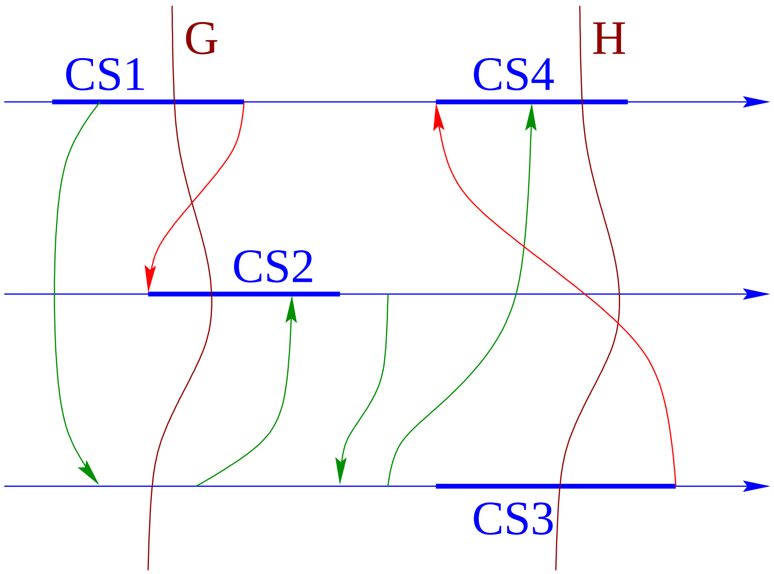
Why would a race occur?

- the programmer made a mistake
- changing requirements
- optimistic approaches

How does it work?



Traced Computation



Controlling Computation

Assumption: identifiable critical sections

Trace-and-replay, race detection in message-passing systems

Replay under control: adding synchronizations

Future Work: Implementation

Goals:

- to determine time and space overheads
- to study implementation issues
- to demonstrate viability

Status

Future Work: Theory

Extending Predicate Control:

- deadlocks
- channel predicates
- K-mutual exclusion

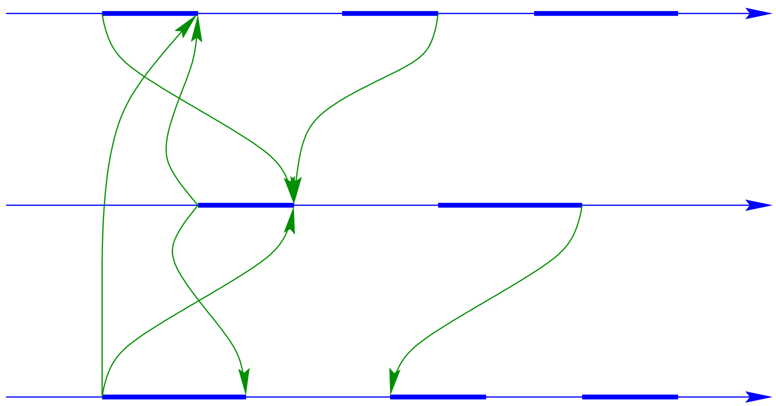
Research Summary

- Pre-Proposal Work:
  - Predicate Control: Disjunctive Predicates [Tarafdar and Garg 98a]
  - False Causality: Predicate Detection [Tarafdar and Garg 98b]
  - False Causality: General [Tarafdar and Garg 98c]

- Post-Proposal Work:
  - Predicate Control: Mutual Exclusion [Tarafdar and Garg 99a]
  - False Causality: Optimistic Recovery [Damani, Tarafdar and Garg 99b]

- Future Work:
  - Predicate Control: Other Predicates
  - Predicate Control: Software Fault Tolerance

Extra Slide: Algorithm



repeat until done

find a crossable pair of true and false intervals

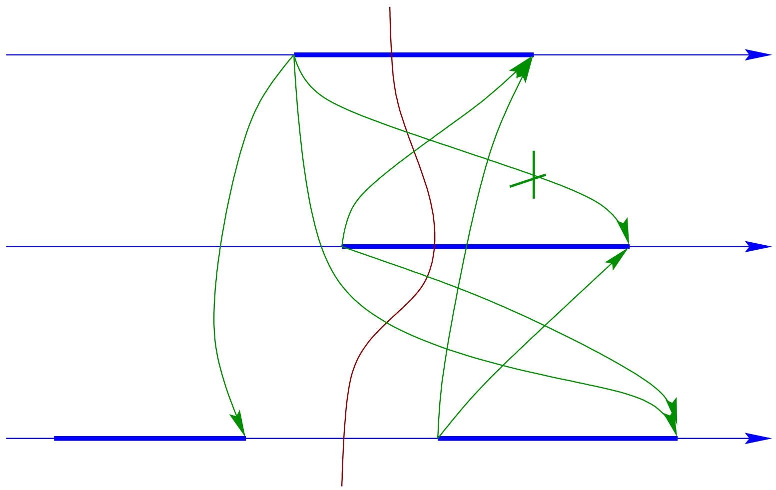
if none exists then exit("no controlling computation exists")

advance to the least consistent global state that crosses the false interval

save the true interval in the chain

Extra Slide: Proof Hint - 1

no crossable pair exists $\Rightarrow$ no controlling computation exists



Extra Slide: Proof Hint - 2

the chain of true intervals does not create any cycles