Software Fault Tolerance of Concurrent Programs Using Controlled Re-execution

Ashis Tarafdar
ashis@cs.utexas.edu

Vijay K. Garg
garg@ece.utexas.edu

Parallel and Distributed Systems Laboratory
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, USA 78712

http://maple.ece.utexas.edu

Introduction

Software Fault Tolerance:

to ensure that the system continues normal operation despite the presence of software faults (bugs)

software faults cause software failures

Goals

- A new approach to software fault tolerance
- The predicate control problem: introduction and results

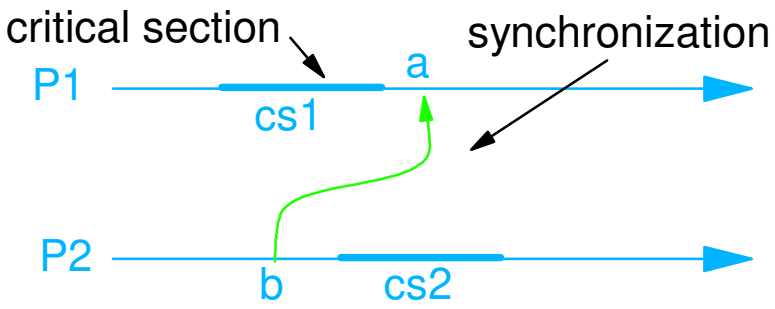
Background: Software Fault Tolerance

- The Progressive Retry Approach: [Wang et al, 1997]
  - software failures are often transient
  - rollback and re-execute
  - no guarantees

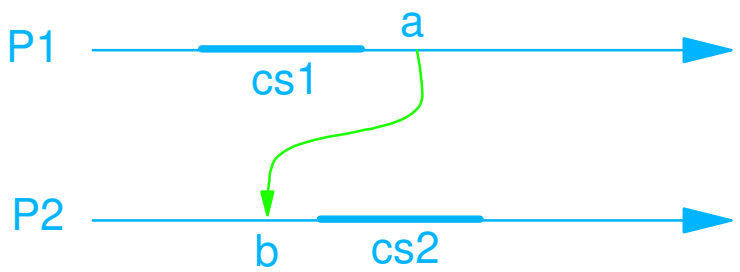
Background: Races in Concurrent Programs

What is a race?

A race occurs when two processes can concurrently access the same shared resource.



A race in a concurrent computation

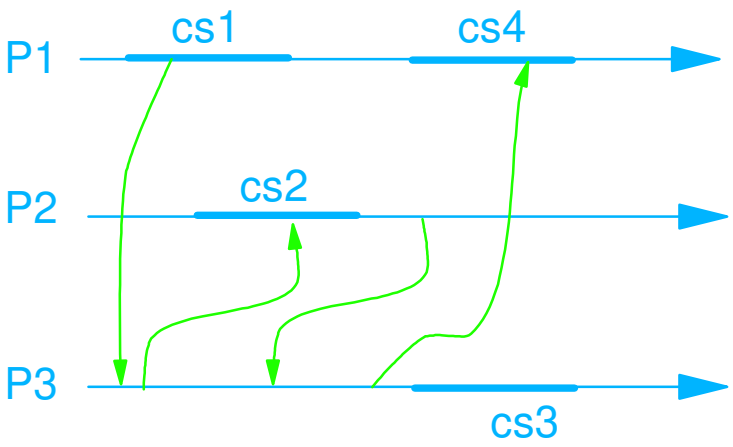


A race-free computation

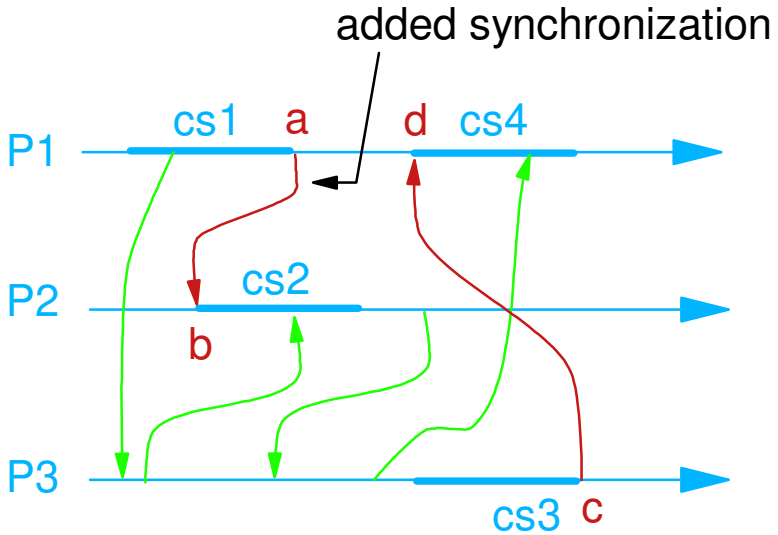
Races are an important class of software faults. [Iyer & Lee, 95]

The Controlled Re-execution Approach

1. Tracing an execution
2. Detecting a race failure
3. Determining a control strategy
4. Re-executing under control

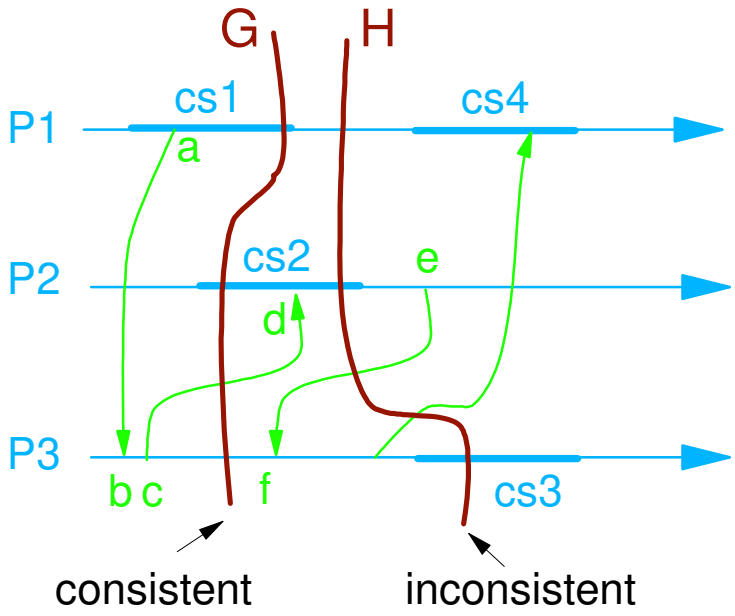


Traced Computation



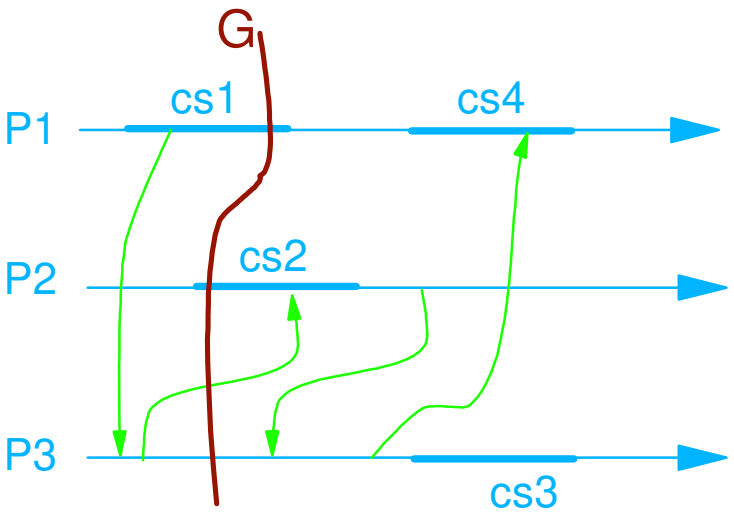
Controlling Computation

Model

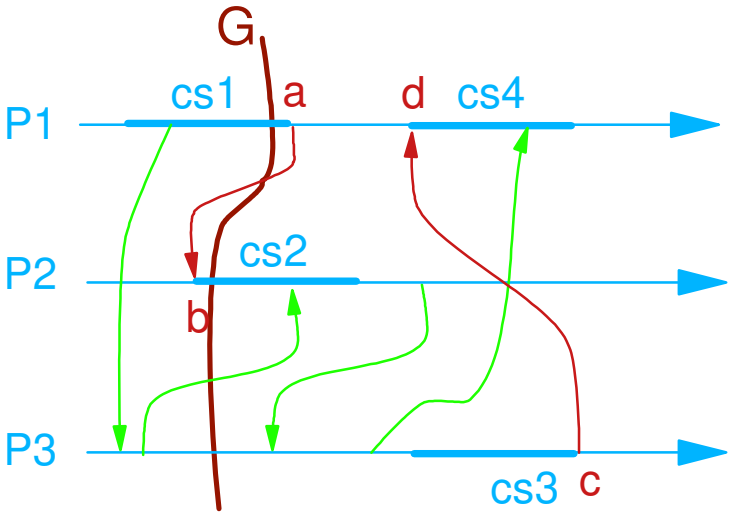


states
computation (happened before)
global state
consistent global state
global predicate (e.g. mutual exclusion)

The Off-line Predicate Control Problem



Computation C



Controlling Computation C'
of B in C

B = mutual exclusion

Note: A controlling computation must have no cycles !

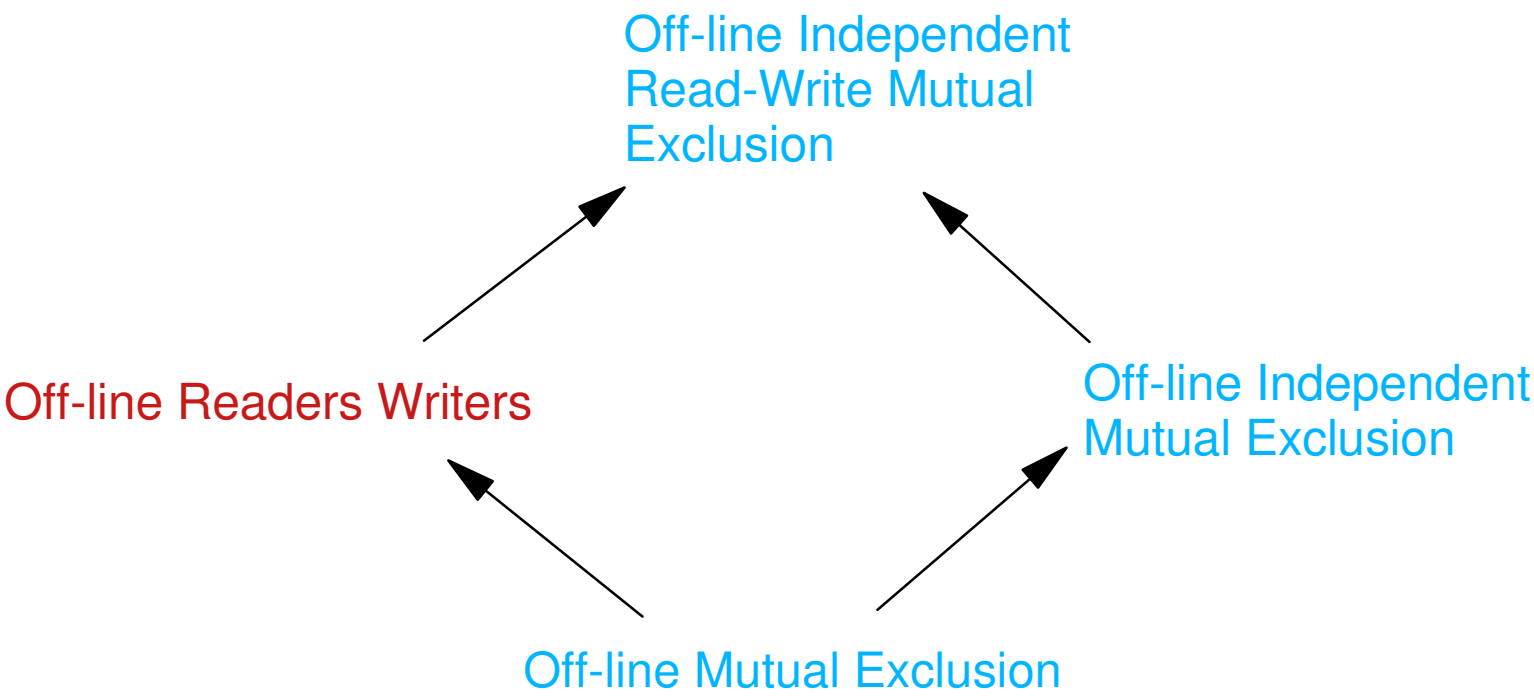
Problem Statement:

Given a computation C and a global predicate B, find a controlling computation of B in C

Off-line Mutual Exclusion

Theorem: The off-line predicate control problem is NP-Hard

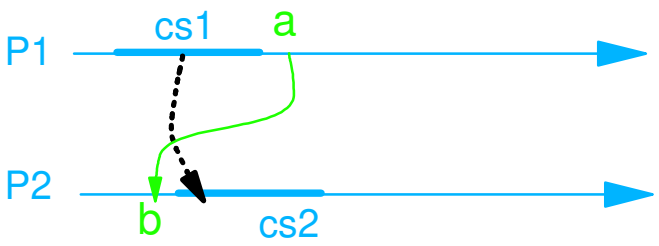
[Tarafdar & Garg, 98]

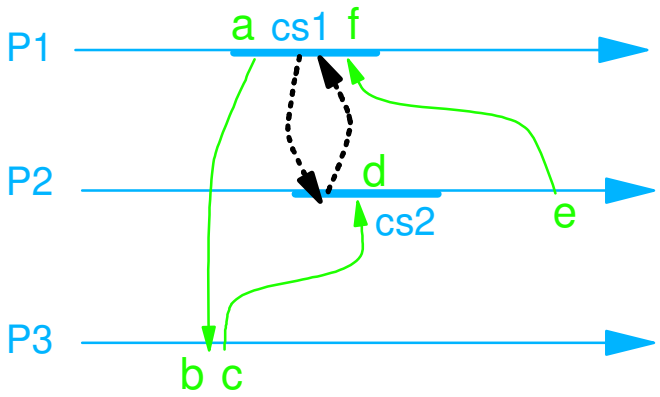


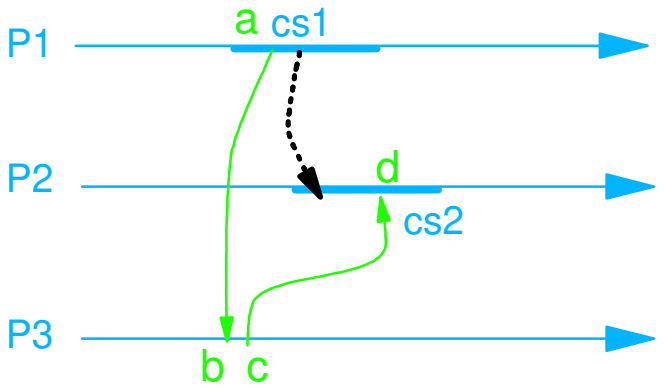
Variants of Off-line Mutual Exclusion

A Relation on Critical Sections

cs1 ⇢ cs2 iff cs1 starts before cs2 finishes







Off-line Readers Writers: Result

Theorem: For a computation C and a global predicate B_{rw} ,

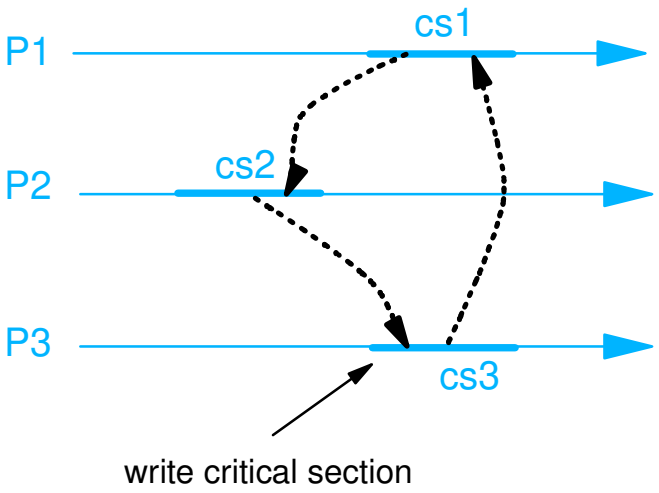
a controlling computation of B_{rw} in C exists

iff

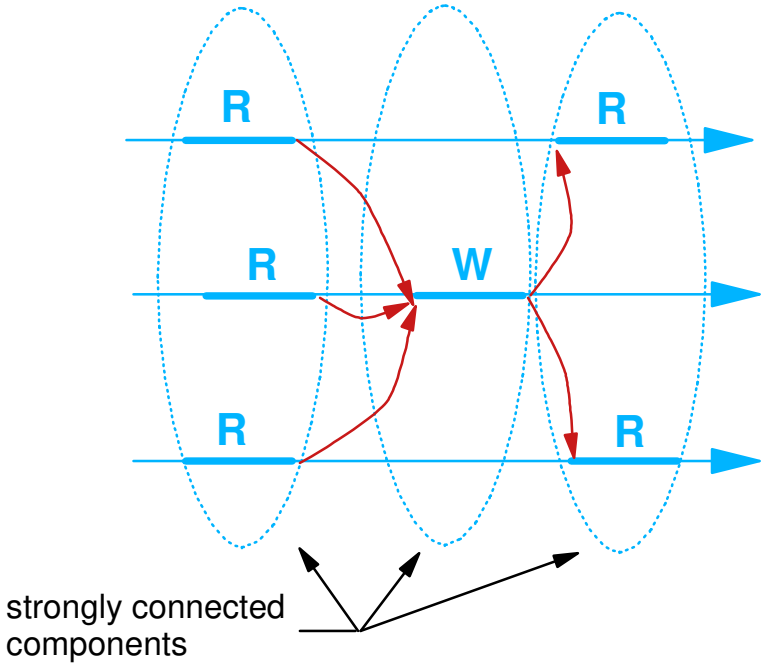
all cycles in ⇢ contain only read critical sections

Proof: Key Ideas:

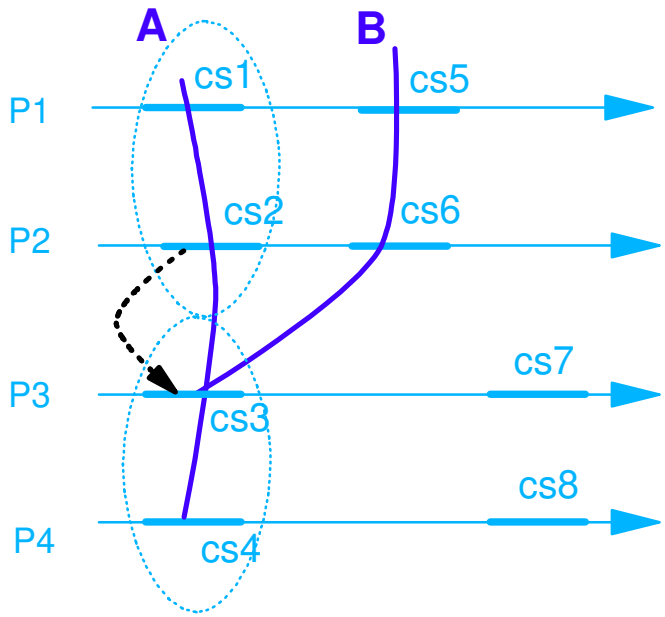
Necessary:



Sufficient:



Off-line Readers Writers: Algorithm



n : number of processes

p : number of critical sections in computation

Algorithm 1: $O(p^2)$

Algorithm 2: $O(n^2p)$

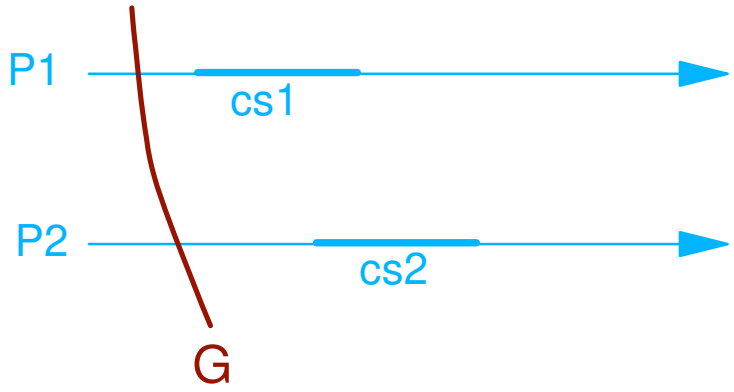
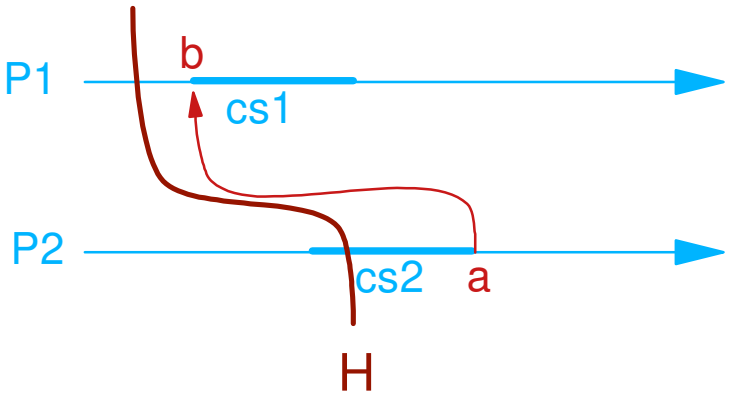
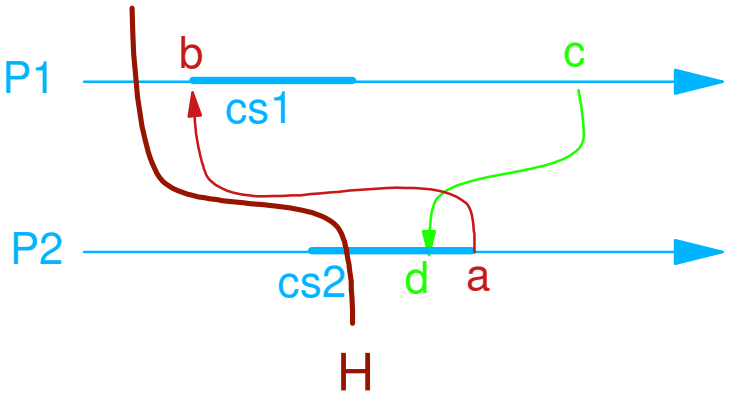
Algorithm 3: $O(np)$

Key Idea:
An SCC contains at most one CS per process

Key Idea:
Only "new" CS's need be considered

Summary

- A new approach to software fault tolerance
  - introduced the controlled re-execution approach for race faults
  - focussed on the problem of determining a control strategy
- The off-line predicate control problem: introduction and results
  - defined the off-line predicate control problem
  - necessary and sufficient conditions for the off-line readers writers problem
  - *O(np)* algorithm for the off-line readers writers problem

also: other variants of off-line mutual exclusion

On-line Mutual Exclusion is Impossible