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 [Taraifdar and Garg 98a]
  – False Causality: Predicate Detection [Taraifdar and Garg 98b]
  – False Causality: General [Taraifdar and Garg 98c]

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

\[ B = \neg (critical_1 \land critical_2) \]
\[ K \text{ satisfies } B \]
\[ G \text{ and } H \text{ do not satisfy } B \]
→^c is a controlling computation of B in →, if:

(1) →^c is stricter than →, and
(2) all consistent global states in →^c satisfy B
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 [Taraﬁdar and Garg 98a]
- Software fault-tolerance of concurrent programs [Taraﬁdar and Garg 99a]
Disjunctive Predicates

\[ B = l_1 \lor l_2 \lor \cdots \lor l_n \]

true intervals and false intervals

\( H \) satisfies \( B \), and \( G \) does not satisfy \( B \)
Disjunctive Predicates: Examples

- At least one philosopher is thinking:
  \[ \text{think}_1 \lor \text{think}_2 \lor \cdots \lor \text{think}_n \]

- At least one server is available:
  \[ \text{avail}_1 \lor \text{avail}_2 \lor \cdots \lor \text{avail}_n \]

- Two-process mutual exclusion:
  \[ \neg \text{critical}_1 \lor \neg \text{critical}_2 \]
Adding Synchronizations

Diagram showing the process of adding synchronizations in a system.
No Controlling Computations

sometimes no controlling computation exists!

\( n \) overlapping false intervals
Conditions for Existence of Controlling Computation

Necessary Condition:
\[\text{a controlling computation exists} \Rightarrow \text{no } n \text{ overlapping false intervals}\]

Sufficient Condition:
\[\text{no } n \text{ overlapping false intervals} \Rightarrow \text{a controlling computation exists}\]
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

next true interval

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]

Simple Mutual Exclusion

Readers Writers

Independent Mutual Exclusion

Independent Read-Write Mutual Exclusion

$O(mn)$ algorithm, where $m$ is the number of critical sections
Software Fault Tolerance: Background

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

- Recently, it was discovered that many software failures are transient
  ⇒ 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

CS1 | G
CS2
CS3 | CS4 | H

Controlling Computation

CS1 | G
CS2
CS3 | CS4 | H

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 [Taraﬁdar and Garg 98a]
  – False Causality: Predicate Detection [Taraﬁdar and Garg 98b]
  – False Causality: General [Taraﬁdar and Garg 98c]

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

• Future Work:
  – Predicate Control: Other Predicates
  – Predicate Control: Software Fault Tolerance
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
no crossable pair exists $\Rightarrow$ no controlling computation exists
the chain of true intervals does not create any cycles