Concurrency Theory

Module 1: Introduction to Concurrency

Khushraj Madnani

Department of Computer Science and Engineering
IIT Guwahati

1. What is Concurrency?

- **Concurrency**: multiple computational activities overlap in time
- Components may execute independently and interact
- Interaction via:
  - shared memory
  - message passing
  - synchronization primitives
- Core difficulty: reasoning about **all possible behaviors**

Key question: How do we model and reason about such systems formally?

2. Concurrency vs. Parallelism

Concurrency

Logical notion: multiple tasks progress independently
May execute on a single processor (via interleaving)

Parallelism

Physical notion: tasks execute simultaneously
Requires multiple processing units

Important: Concurrency does not imply parallelism, but parallelism implies concurrency.

3. Sources of Nondeterminism

- Scheduling decisions (OS, runtime, hardware)
- Communication delays and message ordering
- Shared resource contention
- Interaction with environment

Example

Two threads incrementing a shared variable may yield different outcomes depending on interleaving.

Nondeterminism is inherent and unavoidable in concurrent systems.

4. Models of Concurrency

Shared Memory

- Direct access to common variables
- Requires mutual exclusion
- Prone to races and deadlocks

Message Passing

- No shared state
- Explicit communication
- Common in distributed systems

Both paradigms need precise semantic models.

5. Example: Cache Coherence Problem

- Modern processors use per-core caches
- Writes by one core may not be immediately visible to others
- Leads to inconsistent views of memory

Coherence Property

All processors observe writes to the same location in a consistent order.

From a theory perspective:

- States = cache configurations
- Transitions = reads, writes, invalidations
- Coherence = **safety property**

6. Example: Transactions and Serializability

- Concurrent transactions may interleave
- Incorrect interleavings cause anomalies (lost updates)

Serializability

Every concurrent execution should be equivalent to some serial execution.

Key insight:

- Executions correspond to traces
- Correctness = equivalence of traces

This motivates trace-based models of concurrency.

7. Why Formal Models?

- State space grows exponentially
- Testing explores only a tiny fraction
- Bugs depend on rare interleavings

We need:

- Mathematical models of behavior
- Proof-based reasoning techniques
- Decidability and complexity results

8. Labeled Transition Systems (LTS)

Definition

A labeled transition system is a triple:

$$(S, Act, \rightarrow)$$

- S: states
- Act: actions
- $\rightarrow \subseteq S \times Act \times S$

Write $s \xrightarrow{a} s'$.

LTS form the semantic backbone of process algebras and verification.

9. Safety and Liveness

Safety

"Nothing bad ever happens"
Example: mutual exclusion is never violated

Liveness

"Something good eventually happens"
Example: every request is eventually served

Many concurrency problems are naturally expressed as safety or liveness properties.

10. Summary and Outlook

- Concurrency introduces nondeterminism and interaction
- Interleavings motivate formal semantics
- Cache coherence and transactions illustrate real-world issues
- LTS provide a foundational behavioral model

Why Study Concurrency Theory?

- Concurrency is not just an implementation issue
- Bugs arise from **unexpected interleavings**
- Testing and simulation are fundamentally insufficient

Key Insight

Concurrency theory studies *sets of possible executions*, not a single run.

Goal of the course:
To develop mathematical tools to reason about *all behaviors* of concurrent systems.

Why Informal Reasoning Fails

- Human intuition assumes sequential reasoning
- Rare interleavings cause catastrophic failures
- Many bugs are:
  - non-reproducible
  - architecture-dependent
  - timing-dependent

Examples

Cache coherence violations, lost updates in transactions, deadlocks, livelocks.

Conclusion: We need formal abstractions.

Module 1: Introduction to Concurrency

Central Question

Why is concurrency fundamentally harder than sequential computation?

Focus

- Nondeterminism and interaction
- Interleavings vs. causality
- Real-world motivation

Key Abstractions

- Cache coherence as a safety problem
- Transaction control and serializability
- Labeled Transition Systems (LTS)

Outcome

Concurrency requires formal semantic models to reason about behavior.

Module 2: Process Algebra

Central Question

How can we *compose* concurrent systems and reason about equivalence?

Focus

- Algebraic description of interacting processes
- Operational semantics via transitions

Key Abstractions

- CCS / CSP style process operators
- Strong and weak bisimulation
- Compositional reasoning

Limitation

Interleaving semantics obscures true concurrency and causality.

Module 3: Trace Theory

Central Question

Which executions are equivalent modulo reordering of independent actions?

Focus

- Independence and causality
- Partial-order semantics

Key Abstractions

- Mazurkiewicz traces
- Trace monoids and Foata normal forms
- Event structures

Limitation

Trace models capture behavior but abstract away state and resources.

Module 4: Petri Nets, VAS, and WSTS

Central Question

Which properties of infinite-state concurrent systems are decidable?

Focus

- Concurrency with explicit resources
- Infinite-state behavior

Key Abstractions

- Petri Nets and Vector Addition Systems
- Coverability problem
- WSTS, Karp–Miller trees, Rackoff bounds

Outcome

Powerful decidability results at the cost of limited expressiveness.