

Concurrency Workbench (CWB)

(A Formal Tool for Analyzing Concurrent Systems)

Dr. Varsha Singh

7th Feb 2025

Introduction

- A tool for analyzing concurrent systems using process algebras (CCS, CSP, etc.)
- Supports verification techniques like bisimulation checking, equivalence checking, and model checking

Need for Concurrency Workbench

- Why analyze concurrent systems?
 - because they involve multiple processes running simultaneously, which can lead to unexpected issues like:
- Issues in concurrency: Deadlocks, Race Conditions, Livelocks, Fairness & Starvation, Correctness & Verification
- Importance of formal verification in reactive and distributed systems as it ensures that **reactive** and **distributed systems** function correctly under all conditions.
- Here's why it's crucial:
 - Eliminating Errors in Critical Systems, Handling Concurrency Issues, Ensuring Security & Robustness, Proving System Properties, Improving Reliability in Distributed Systems.
- Beyond Testing: Exhaustive Checking
- Real-World Example: Model Checking in Hardware Verification

Features of Concurrency Workbench

- Supports multiple process calculi (CCS, CSP, LOTOS)
- Allows simulation and verification of models
- Implements fixed-point algorithms for model checking
- Can check bisimulation and logical properties

Process Algebra & CCS in CWB

- **CCS (Calculus of Communicating Systems)** as a foundation
- CCS Syntax and Operators:
 - Action Prefix: $a.P$
 - Choice: $P + Q$
 - Parallel Composition: $P \mid Q$
 - Restriction: $P \backslash \{a\}$
- Example CCS process in CWB

Bisimulation

Bisimulation is an equivalence relation between two processes in a concurrent system, ensuring that they behave indistinguishably in all possible executions.

Two processes P and Q are bisimilar (denoted as $P \sim Q$) if:

1. Whenever P makes a transition $P \xrightarrow{a} P'$, there exists a corresponding transition Q can make $Q \xrightarrow{a} Q'$ such that $P' \sim Q'$.
2. Conversely, whenever Q makes a transition $Q \xrightarrow{a} Q'$, there exists a corresponding transition $P \xrightarrow{a} P'$ such that $P' \sim Q'$.

This ensures that P and Q always match each other's observable behaviors step by step.

Strong bisimulation

Two processes are strongly bisimilar if they match each other's observable actions exactly, step by step. There must be a one-to-one correspondence between every action performed by one process and the other.

✓ **Key Property:** No invisible or internal steps are ignored; every transition must match precisely.

Example of Strong Bisimulation

Let's define two CCS processes:

$$P = a.P_1 + b.P_2$$

$$Q = a.Q_1 + b.Q_2$$

where

$$P_1 = c.0, \quad P_2 = d.0, \quad Q_1 = c.0, \quad Q_2 = d.0$$

- If P does **a**, Q must also do **a**, and they must reach **bisimilar states**.
- If P does **b**, Q must also do **b**, and again they must be bisimilar.

Since every step matches **exactly**, $P \sim Q$ under **strong bisimulation**.

Weak bisimulation

Two processes are weakly bisimilar if they can match observable actions, but they are allowed to skip over internal (τ) transitions.

✓ Key Property:

- Internal (hidden) transitions τ are ignored when comparing processes.
- If one process can take multiple τ -steps before an observable action, the other process must be able to match the final observable action.

Example of Weak bisimulation

Consider the processes:

$$P = \tau.a.0$$

$$Q = a.0$$



✓ Step-by-Step Weak Bisimulation Check:

- P can take an internal τ -transition and become $a.0$.
- Q directly performs $a.0$ without τ .
- Since τ is **unobservable**, both processes appear the same from the outside.

Therefore, Even though P has an extra τ -transition, it does not affect the observable behavior. P and Q are weakly bisimilar:

$$P \approx Q$$

Strong vs. Weak Bisimulation

Feature	Strong Bisimulation \sim	Weak Bisimulation \approx
Internal (τ) steps	Must match exactly	Can be ignored
Transitions	One-to-one matching	May skip τ -steps
More flexible?	 No (strict)	 Yes (allows abstraction)
Example	$a.b.0 \sim a.b.0$	$\tau.a.b.0 \approx a.b.0$

Then, Why Use Weak Bisimulation?

Reason is:

- In real-world systems, internal actions (τ -transitions) often represent hidden computations that we do not need to verify.
- Weak bisimulation is useful for abstracting system behavior without worrying about implementation details.

CWB's role in checking bisimulation equivalence

Concurrency Workbench (CWB) and Bisimulation Equivalence

Concurrency Workbench (CWB) is a formal verification tool that helps check bisimulation equivalence between processes modeled using process algebra (e.g., CCS, CSP, and LOTOS). It automates the comparison of two processes to determine if they behave identically in all possible executions.

- How CWB Checks Bisimulation Equivalence?

CWB Checks Bisimulation Equivalence

1. CWB allows users to define concurrent processes using **CCS syntax**.
2. Transition System Generation:
 - a. CWB constructs the Labelled Transition System (LTS) for each process.
 - b. This LTS represents all possible states and transitions of the process.
3. Bisimulation Comparison:
 - a. CWB applies strong or weak bisimulation algorithms to compare two processes.
 - b. The tool checks if every transition in one process has a matching transition in the other.
 - c. If all steps match under the chosen bisimulation type (strong or weak), the processes are equivalent.
4. Verification Output:
 - a. If the processes are bisimilar, CWB confirms equivalence
 - b. If not, CWB provides a counterexample showing where the processes differ.

Strong Bisimulation Check in CWB

1. CCS Process:

$P = a.(b.0 + c.0);$

$Q = a.b.0 + a.c.0;$

2. Using CWB Command:

`bisim P Q;`

3. CWB Output:

P and Q are NOT strongly bisimilar ❌

Reason

- In P , after a , a choice is made between b and c .
- In Q , a is duplicated before the choice happens.
- Since transitions do not match exactly, P and Q fail **strong bisimulation**.

Weak Bisimulation Check in CWB

1. CCS Processes with τ -Transitions:

$P = \tau.a.0;$

$Q = a.0;$

2. Using CWB Command:

`wbisim P Q;`

3. CWB Output:

P and Q are weakly bisimilar 

Reason

- P can silently perform τ , then execute a .
- Since weak bisimulation ignores τ -transitions, P and Q are considered equivalent.

Advantages of Using CWB for Bisimulation Checking

- ✓ Automates tedious manual verification of process equivalence.
- ✓ Handles both strong and weak bisimulation efficiently.
- ✓ Generates counterexamples when processes are not bisimilar.
- ✓ Supports real-world verification of concurrent system correctness.

Model Checking in CWB

- **Temporal logic verification** in CWB
- Safety and Liveness properties
- Hennessy-Milner logic (HML) integration
- Example: Checking if a system satisfies a temporal property

Using CWB – Workflow

- Define a process in CCS
- Specify the verification goal (e.g., equivalence, model checking)
- Execute the verification using CWB commands
- Analyze results

Example: Simple System in CWB

- Define a CCS model of a simple concurrent system
- Check for deadlock-free execution
- Demonstrate basic CWB commands

Advantages & Limitations

✓ Advantages:

- Supports multiple verification techniques
- Useful for both theoretical and practical system analysis
- Strong foundation in process algebra

⚠ Limitations:

- Steep learning curve
- Scalability issues for large systems
- Requires expertise in process algebra

Summary

- Concurrency Workbench helps in formal verification of concurrent systems
- Uses CCS and bisimulation techniques
- Supports model checking and equivalence checking
- Practical tool for validating system correctness

References & Further Reading

- [1] Robin Milner, Communication and Concurrency
- [2] CWB Documentation
- [3] Research papers on process algebra & verification