Revising Transition Systems and Understanding Behavioral Equivalences

Khushraj

Reference

As easier reference for CCS.

Easier Reference for CCS:
Roberto Gorrieri and Cristian Versari
Introduction to Concurrency Theory

These slides follow the structure and terminology of this chapter.

Why Transition Systems?

- Process algebras describe systems syntactically
- We need a mathematical model of behaviour
- Behaviour is captured by states and transitions

Key abstraction:

Systems are transition systems; programs are states.

Labelled Transition Systems - Revising

Definition

A *Labelled Transition System (LTS)* is a triple:

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

where:

- S is a set of states
- Act is a set of action labels
- $\rightarrow \subseteq S \times Act \times S$

We write:

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

Observable and Internal Actions

- Observable actions: $a, b, \ldots$
- Internal (silent) action: τ

Intuition:

- τ represents unobservable internal computation/communication.
- Used to abstract implementation details.

This distinction is central to behavioral equivalences.

From CCS to LTS

- CCS processes are LTS states
- Operational semantics generates transitions

Example

$$a.P \xrightarrow{a} P$$

$$(P \mid Q) \xrightarrow{\tau} (P' \mid Q')$$

when complementary actions synchronize.

(Full SOS rules done in Lecture 1 and 2.)

Paths and Computations

Definition

A *path* is a sequence:

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots$$

A *computation* is a maximal path.

Paths describe executions; equivalences compare sets of paths.

Traces

Definition

A trace is the sequence of observable actions along a path, ignoring τ.

$$a_1 a_2 \cdots a_n$$

Trace semantics:

- Abstracts from branching
- Abstracts from internal structure

Trace Equivalence

Two states are *trace equivalent* if they have the same set of traces.
LTS of Two Vending-Machine Behaviours: Example

Processes:

$$P ::= coin.(\overline{tea} + \overline{coffee}) \qquad Q ::= coin.\overline{tea} + coin.\overline{coffee}$$

LTS of P:

$$P \xrightarrow{coin} P' \quad \text{where} \quad P' \xrightarrow{\overline{tea}} \mathbf{0} \text{ and } P' \xrightarrow{\overline{coffee}} \mathbf{0}$$

LTS of Q:

$$Q \xrightarrow{coin} Q_{\overline{tea}} \xrightarrow{\overline{tea}} \mathbf{0} \qquad Q \xrightarrow{coin} Q_{\overline{coffee}} \xrightarrow{\overline{coffee}} \mathbf{0}$$

Key structural difference:

- In P, the drink is chosen *after* inserting the coin
- In Q, the choice is made *before* inserting the coin

Trace Equivalent but Not Behaviourally Equivalent

Consider the two vending-machine processes:

$$P ::= coin.(tea + coffee) \qquad Q ::= coin.tea + coin.coffee$$

Traces:

$$Tr(P) = \{\, coin\, tea,\ coin\, coffee \,\}$$
$$Tr(Q) = \{\, coin\, tea,\ coin\, coffee \,\}$$

Observation:

- Both machines allow the same sequences of visible actions
- Hence, they are *trace equivalent*

However:

- In P, the choice of drink happens *after* payment
- In Q, the choice is fixed *before* payment

Conclusion:

$$P \ \not\sim_{\text{bisim}}$$

Trace equivalence does not preserve when choices are made.

Branching Structure Matters

- Concurrency introduces choices
- Choices affect future possibilities

Two systems may have the same traces but different interaction capabilities.

This motivates stronger equivalences.

Strong Bisimulation

Definition

A relation $\mathcal{R} \subseteq S \times S$ is a *strong bisimulation* if:

Whenever $(s, t) \in \mathcal{R}$:

- if $s \xrightarrow{a} s'$, then $t \xrightarrow{a} t'$ with $(s', t') \in \mathcal{R}$
- and symmetrically

Strong Bisimulation: Intuition

- Step-by-step matching
- Observes all actions, including τ
- Preserves full branching structure

Very precise, but often too discriminating.

Weak Transitions

Definition

$$s \stackrel{a}{\Longrightarrow} s'$$

means:

- zero or more τ
- followed by action a
- followed by zero or more τ

Weak transitions abstract from internal computation.

Weak Bisimulation

Definition

A relation $\mathcal{R}$ is a *weak bisimulation* if:

Whenever $(s, t) \in \mathcal{R}$:

- if $s \xrightarrow{a} s'$, then $t \xRightarrow{a} t'$ with $(s', t') \in \mathcal{R}$
- and symmetrically

Why Weak Bisimulation?

- Abstracts from implementation details
- Preserves observable interaction patterns
- Matches refinement intuition

Central equivalence in concurrency theory.

Equivalence Spectrum

$$\begin{array}{c}\text{Strong bisimulation}\\ \Downarrow \\ \text{Weak bisimulation}\\ \Downarrow \\ \text{Trace equivalence}\end{array}$$

More abstraction means less discrimination.

Application to CCS Examples

- Different ATM implementations
- Different DB internal structures

They may differ in τ-structure but:

- are weakly bisimilar
- are not strongly bisimilar

Why Behavioral Equivalence Matters

- Correctness of implementations
- Program refinement
- Protocol verification
- Model checking

Equivalence justifies replacing one system by another.

Summary

- LTS gives semantics to processes
- Traces observe executions
- Bisimulation observes interaction structure
- Weak bisimulation abstracts from internals

This forms the semantic foundation of concurrency theory.