An Introduction to Bisimulation and Coinduction

Davide Sangiorgi

Focus Team,
INRIA (France)/University of Bologna (Italy)

Email: Davide.Sangiorgi@cs.unibo.it
http://www.cs.unibo.it/~sangio/

18th Estonian Winter School in Computer Science,
Palmse, Estonia, March 2013

Induction

- **pervasive in Computer Science and Mathematics**
  definition of objects, proofs of properties, ...
- **stratification**
  finite lists, finite trees, natural numbers, ...
- **natural** (least fixed points)

Coinduction

- **less known**
  discovered and studied in recent years
- **a growing interest**
- **quite different from induction**
  the dual of induction
- **no need of stratification**
  objects can be infinite, or circular
- **natural** (greatest fixed points)

Why coinduction: examples

- **streams**
- **real numbers**
- **a process that continuously accepts interactions with the environment**
- **memory cells with pointers** (and possibly cylces)
- **graphs**
- **objects on which we are unable to place size bounds**
  (eg, a database, a stack)
- **objects that operate in non-fixed environments**
  (eg, distributed systems)

Other examples

In a sequential language

- **definition of the terminating terms**

  inductively, from the rules of the operational semantics

- **definition of the non-terminating terms**
  - the complement set of the terminating ones
    (an indirect definition)
  - coinductively, from the rules of the operational semantics
    (a direct definition: more elegant, better for reasoning)

In constructive mathematics

- **open sets**

  a direct inductive definition

- **closet sets**
  - the complement set of the open ones
  - a direct coinductive definition

Bisimulation

- **the best known instance of coinduction**
- **discovered in Concurrency Theory**
  formalising the idea of behavioural equality on processes
- **one of the most important contributions of Concurrency Theory to CS (and beyond)**
- **it has spurred the study of coinduction**
- **in concurrency: the most studied behavioural equivalence**
  many others have been proposed

Coinduction in programming languages

- **widely used in concurrency theory**
  - to define equality on processes (fundamental !!)
  - to prove equalities
  - to justify algebraic laws
  - to minimise the state space
  - to abstract from certain details
- **functional languages and OO languages**

  A major factor in the movement towards operationally-based techniques in PL semantics in the 90s
- **program analysis** **(see any book on program analysis)**
- **verification tools**: algorithms for computing gfp (for modal and temporal logics), tactics and heuristics

– **Types**
  * type soundness
  * coinductive types and definition by corecursion
    Infinite proofs in Coq
  * recursive types (equality, subtyping, ...)
    A coinductive rule:

$$\frac{\Gamma, \langle p_1, q_1\rangle \sim \langle p_2, q_2\rangle \vdash p_i \sim q_i}{\Gamma \vdash \langle p_1, q_1\rangle \sim \langle p_2, q_2\rangle}$$

– **Databases**

– **Compiler correctness**

– ...

Other fields

Today bisimulation and coinduction also used in

- **Artificial Intelligence**
- **Cognitive Science**
- **Mathematics**
- **Modal Logics**
- **Philosophy**
- **Physics**

mainly to explain phenomena involving some kind of circularity

Objectives of the course

At the end of the course, a student should:

- **have an idea of the meaning of bisimulation and coinduction**
- **have a grasp of the duality between induction and coinduction**
- **be able to read (simple) bisimulation proofs and coinductive definitions**

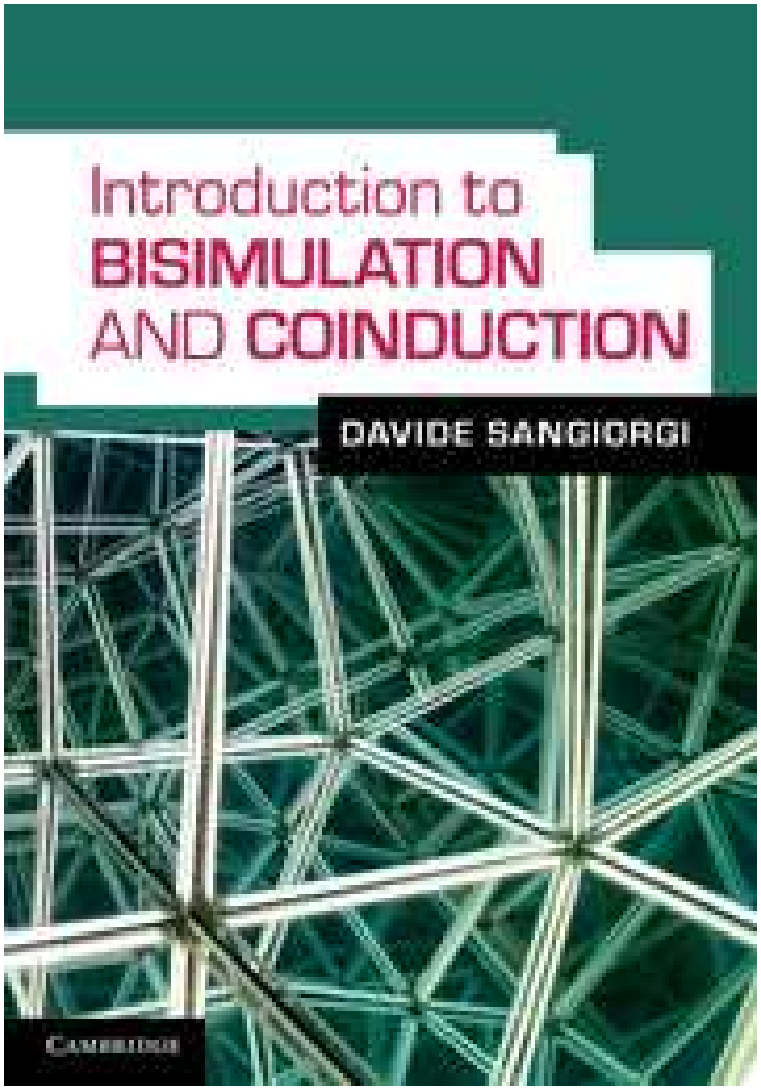
Why induction and coinduction in this school

- **fundamental notions for programming languages**
  - defining structures, objects
  - reasoning on them (proofs, tools)
- **abstract, unifying notions**
- **notions that will still be around when most present-day programming languages will be obsolete**
- **an introduction to these notions that uses some very simple set theory**

References

This course is based on the first 3 chapters of the book:

Davide Sangiorgi, *An introduction to bisimulation and coinduction*, Cambridge University Press, October 2011



Outline

- The success of bisimulation and coinduction
- Towards bisimulation, or: from functions to processes
- Bisimulation
- Induction and coinduction
- (Enhancements of the bisimulation proof method)

Towards bisimulation, or :

from functions to processes

A very simple example: a vending machine

In your office you have a tea/coffee machine, a red box whose behaviour is described thus:

- you put a coin
- you are then allowed to press the tea button or the coffee button
- after pressing the tea button you collect tea, after pressing the coffee button you collect coffee
- after collecting the beverage, the services of machine are again available

Equivalence of machines

- The machine breaks
- You need a new machine, with the same behaviour
- You show the description in the previous slide
- You get a red box machine that, when a tea or coffee button is pressed non-deterministically delivers tea or coffee
- After paying some money, you get another machine, a green box that behaves as you wanted

Questions

1. How can we specify formally the behaviour of the machines?
2. What does it mean that two machines “are the same”?
3. How do we prove that the first replacement machine is “wrong”, and that the second replacement machine is “correct”?

Answers from this course

1. Labelled Transitions Systems (automata-like)
2. Bisimulation
3. Coinduction

Processes?

We can think of sequential computations as mathematical objects, namely **functions**.

Concurrent program are not functions, but **processes**. But what is a process?

No universally-accepted mathematical answer.

Hence we do not find in mathematics tools/concepts for the denotational semantics of concurrent languages, at least not as successful as those for the sequential ones.

Processes are not functions

A sequential imperative language can be viewed as a function from states to states.

These two programs denote the same function from states to states:

$$\mathtt{X} := \mathbf{2} \qquad \text{and} \qquad \mathtt{X} := \mathbf{1};\ \mathtt{X} := \mathtt{X} + \mathbf{1}$$

But now take a context with parallelism, such as $[\cdot] \mid \mathtt{X} := \mathbf{2}$. The program

$$\mathtt{X} := \mathbf{2} \mid \mathtt{X} := \mathbf{2}$$

always terminates with $\mathtt{X} = \mathbf{2}$. This is not true (why?) for

$$(\ \mathtt{X} := \mathbf{1};\ \mathtt{X} := \mathtt{X} + \mathbf{1}\ ) \mid \mathtt{X} := \mathbf{2}$$

Therefore: Viewing processes as functions gives us a notion of equivalence that is not a **congruence**. In other words, such a semantics of processes as functions would not be **compositional**.

Furthermore:

- A concurrent program may not terminate, and yet perform meaningful computations (examples: an operating system, the controllers of a nuclear station or of a railway system).
  In sequential languages programs that do not terminate are undesirable; they are ‘wrong’.

- The behaviour of a concurrent program can be non-deterministic.
  Example:
  $$(\ \mathtt{X} := 1;\ \ \mathtt{X} :=\ \ \mathtt{X} + 1\ ) \mid\ \ \mathtt{X} := 2$$
  In a functional approach, non-determinism can be dealt with using powersets and powerdomains.

This works for pure non-determinism, as in $\boldsymbol{\lambda x.\,(3 \oplus 5)}$

But not for parallelism.

What is a process?
When are two processes behaviourally equivalent?

These are basic, fundamental, questions; they have been at the core of the research in concurrency theory for the past 30 years. (They are still so today, although remarkable progress has been made)

Fundamental for a model or a language on top of which we want to make proofs

Interaction

In the example at page 17

$$\mathtt{x} := \mathbf{2} \qquad \text{and} \qquad \mathtt{x} := \mathbf{1};\ \mathtt{x} := \ \mathtt{x} + \mathbf{1}$$

should be distinguished because they interact in a different way with the memory.

Computation is **interaction**. Examples: access to a memory cell, interrogating a data base, selecting a programme in a washing machine, ...

The participants of an interaction are **processes** (a cell, a data base, a washing machine, ...)

The **behaviour** of a process should tell us **when** and **how** a process can interact with its environment

How to represent interaction: labelled transition systems

Definition A **labeled transition system** (LTS) is a triple $(\mathcal{P}, \mathtt{Act}, \mathcal{T})$ where

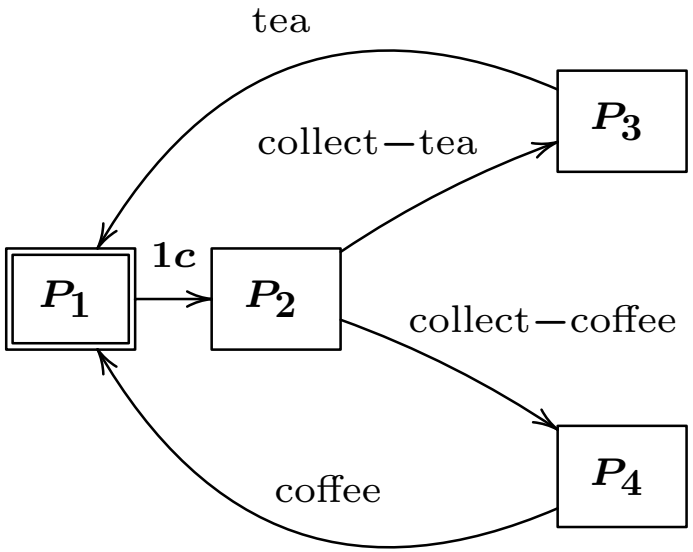
- $\mathcal{P}$ is the set of **states**, or **processes**;
- $\mathtt{Act}$ is the set of **actions**; (NB: can be infinite)
- $\mathcal{T} \subseteq (\mathcal{P}, \mathtt{Act}, \mathcal{P})$ is the **transition relation**.

We write $P \xrightarrow{\mu} P'$ if $(P, \mu, P') \in \mathcal{T}$. Meaning: process P accepts an interaction with the environment where P performs action μ and then becomes process P'.

P' is a **derivative** of P if there are $P_1, \ldots, P_n, \mu_1, \ldots, \mu_n$ s.t. $P \xrightarrow{\mu_1} P_1 \ldots \xrightarrow{\mu_n} P_n$ and $P_n = P'$.

Example: the behaviour of our beloved vending machine

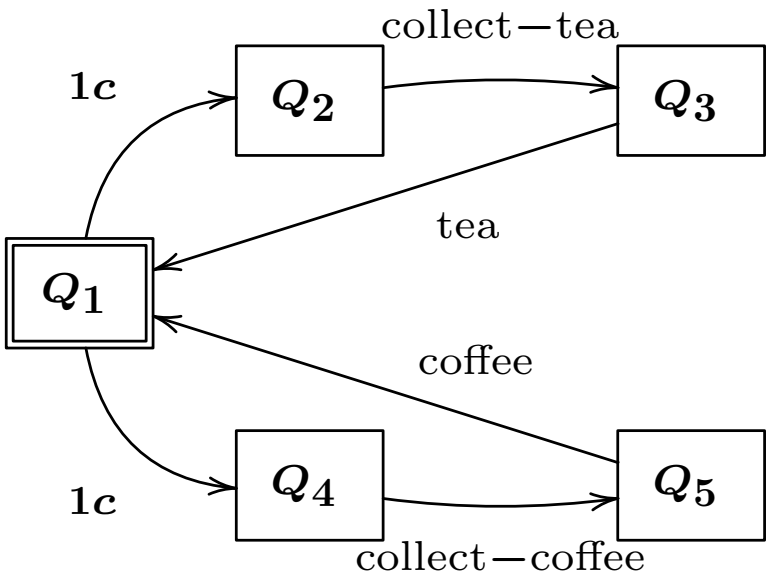
The behaviour is what we can observe, by interacting with the machine.
We can represent such a behaviour as an LTS:



where ▭ indicates the processes we are interested in (the “initial state”)

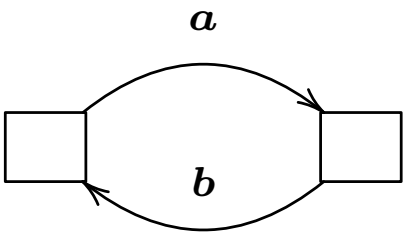
NB: the color of the machine is irrelevant

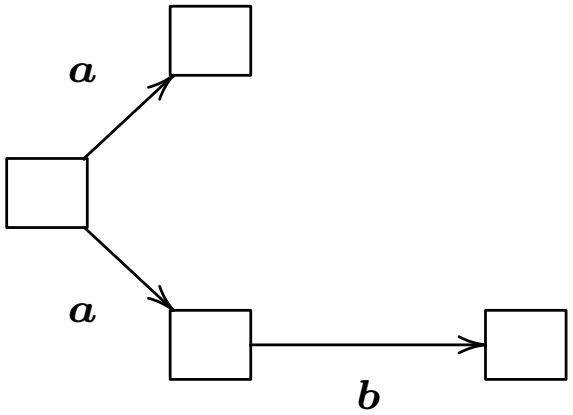
The behaviour of the first replacement machine



Other examples of LTS

(we omit the name of the states)





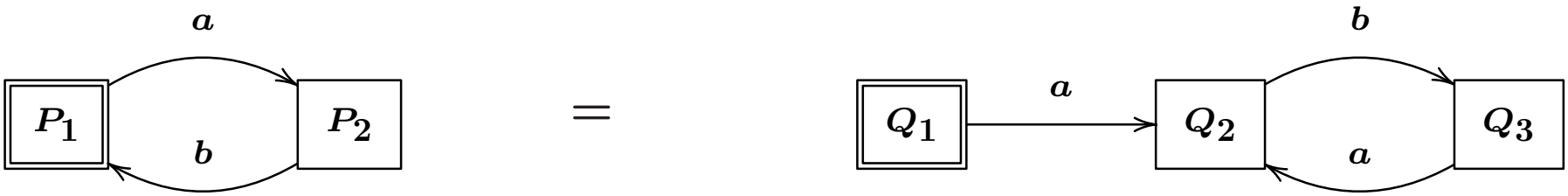
Now we now: how to write beahviours

Next: When should two behaviours be considered equal?

Equivalence of processes

Two processes should be equivalent if we cannot distinguish them by interacting with them.

Example

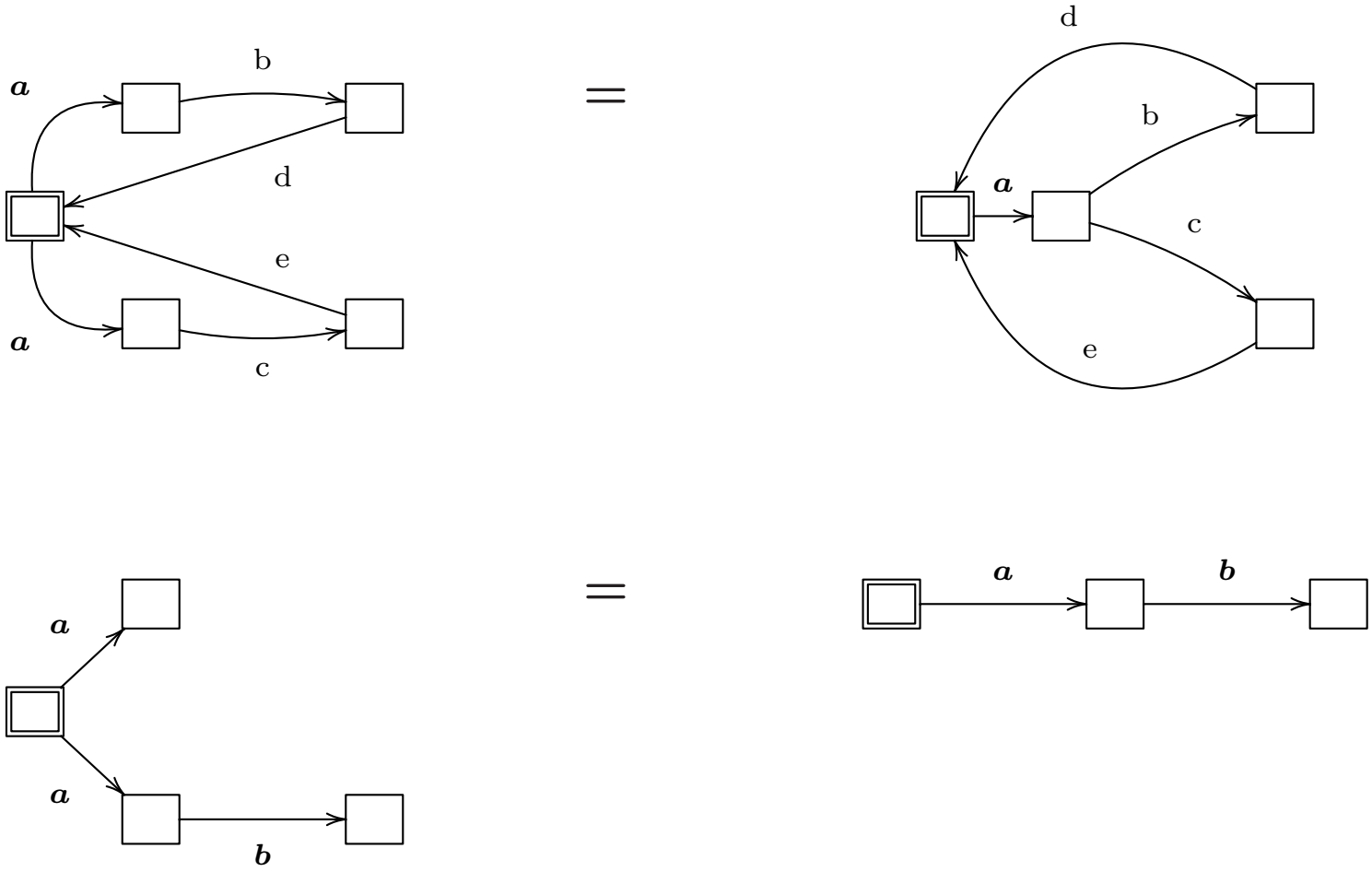


Can **graph theory** help? (equality is **graph isomorphism**)

... too strong (example above)

What about **automata theory**? (equality is **trace equivalence**)

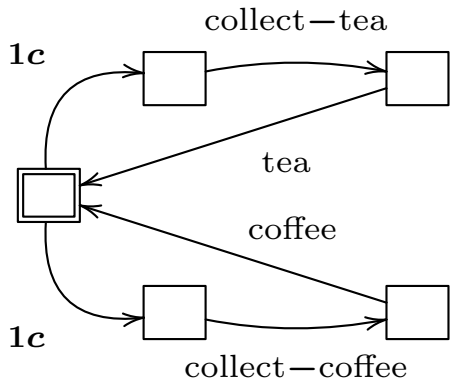
Examples of trace-equivalent processes:

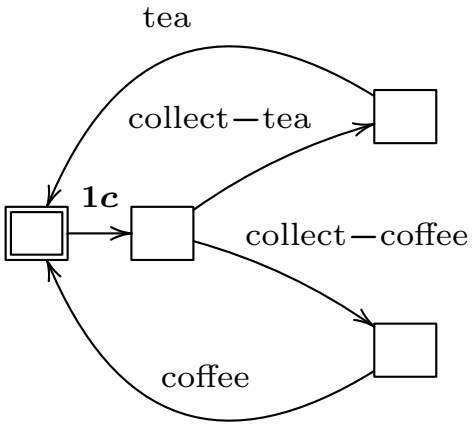


These equalities are OK on automata.

... but they are not on processes (deadlock risk!)

For instance, you would not consider these two vending machines ‘the same’:





Trace equivalence (also called language equivalence) is still important in concurrency.

Examples: confluent processes; liveness properties such as termination

These examples suggest that the notion of equivalence we seek:

- should imply a tighter correspondence between transitions than language equivalence,
- should be based on the informations that the transitions convey, and not on the shape of the diagrams.

Intuitively, what does it mean for an observer that two machines are equivalent?

If you do something with one machine, you must be able to the same with the other, and on the two states which the machines evolve to the same is again true.

This is the idea of equivalence that we are going to formalise; it is called **bisimilarity**.

Bisimulation and bisimilarity

We define bisimulation on a single LTS, because: the union of two LTSs is an LTS; we will often want to compare derivatives of the same process.

Definition A relation $\mathcal{R}$ on processes is a **bisimulation** if whenever $P \mathcal{R} Q$:

1. $\forall \mu, P'$ s.t. $P \xrightarrow{\mu} P'$, then $\exists Q'$ such that $Q \xrightarrow{\mu} Q'$ and $P' \mathcal{R} Q'$;
2. $\forall \mu, Q'$ s.t. $Q \xrightarrow{\mu} Q'$, then $\exists P'$ such that $P \xrightarrow{\mu} P'$ and $P' \mathcal{R} Q'$.

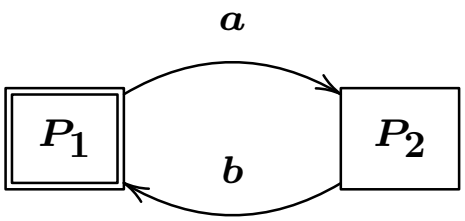
P and Q are **bisimilar**, written $P \sim Q$, if $P \mathcal{R} Q$, for some bisimulation $\mathcal{R}$.

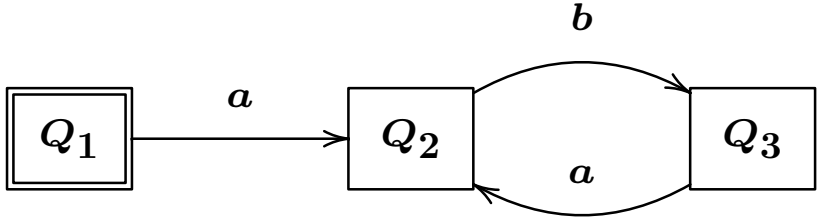
The bisimulation diagram:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \mu \downarrow & & \mu \downarrow \\ P' & \mathcal{R} & Q' \end{array}$$

Examples

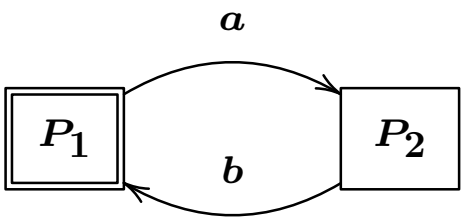
Show $P_1 \sim Q_1$ (easy, processes are deterministic):

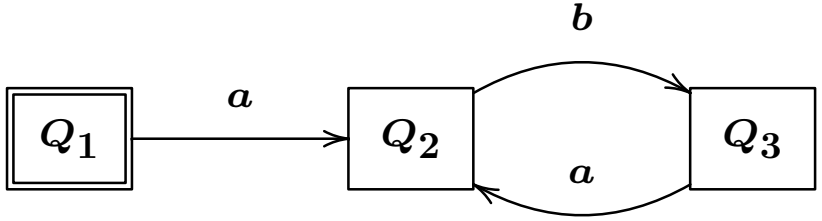




Examples

Show $P_1 \sim Q_1$ (easy, processes are deterministic):





First attempt for a bisimulation:

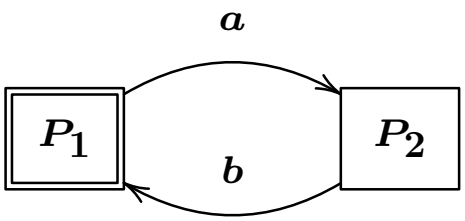
$\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2)\}$

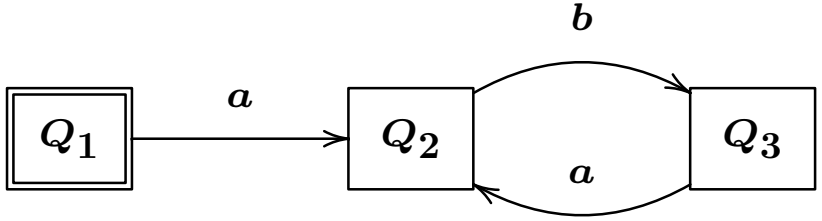
Bisimulation diagrams for (P_1, Q_1):

$$\begin{array}{ccc} P_1 & \mathcal{R} & Q_1 \\ a \downarrow & & \\ P_2 & & \end{array} \qquad \begin{array}{ccc} P_1 & \mathcal{R} & Q_1 \\ & & a \downarrow \\ & & Q_2 \end{array}$$

Examples

Show $P_1 \sim Q_1$ (easy, processes are deterministic):





First attempt for a bisimulation:

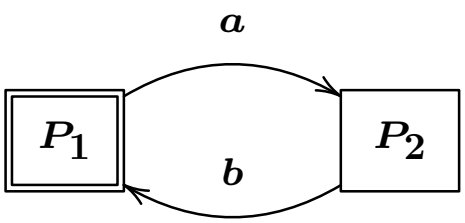
$\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2)\}$

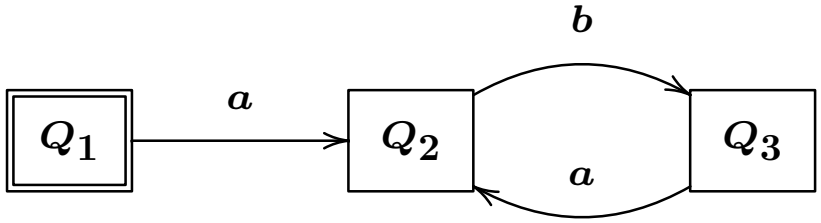
Bisimulation diagrams for (P_1, Q_1):

$$\begin{array}{ccc} P_1 & \mathcal{R} & Q_1 \\ a \downarrow & & a \downarrow \\ P_2 & \mathcal{R} & Q_2 \end{array} \qquad \begin{array}{ccc} P_1 & \mathcal{R} & Q_1 \\ a \downarrow & & a \downarrow \\ P_2 & \mathcal{R} & Q_2 \end{array}$$

Examples

Show $P_1 \sim Q_1$ (easy, processes are deterministic):





First attempt for a bisimulation:

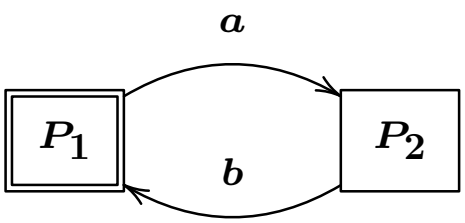
$\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2)\}$

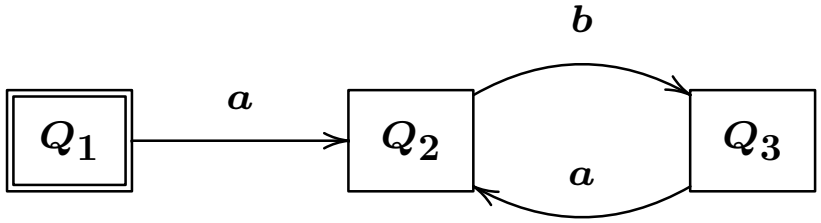
Bisimulation diagrams for (P_2, Q_2):

$$\begin{array}{ccc} P_2 & \mathcal{R} & Q_2 \\ b \downarrow & & \\ P_1 & & \end{array} \qquad \begin{array}{ccc} P_2 & \mathcal{R} & Q_2 \\ & & b \downarrow \\ & & Q_3 \end{array}$$

Examples

Show $P_1 \sim Q_1$ (easy, processes are deterministic):





First attempt for a bisimulation:

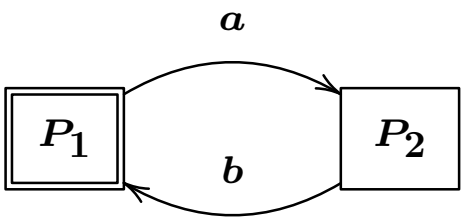
$\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2)\}$

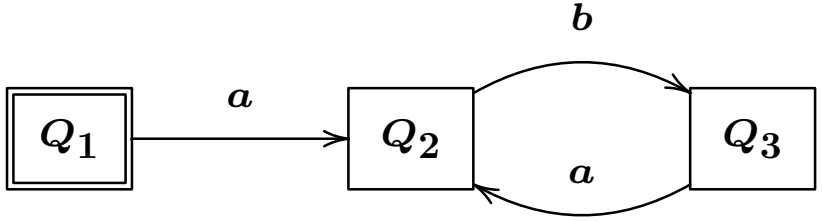
Bisimulation diagrams for (P_2, Q_2):

$$\begin{array}{ccc} P_2 & \mathcal{R} & Q_2 \\ b \downarrow & & b \downarrow \\ P_1 & \not\mathcal{R} & Q_3 \end{array} \qquad \begin{array}{ccc} P_2 & \mathcal{R} & Q_2 \\ b \downarrow & & b \downarrow \\ P_1 & \not\mathcal{R} & Q_3 \end{array}$$

Examples

Show $P_1 \sim Q_1$ (easy, processes are deterministic):



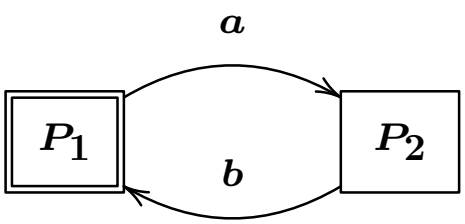


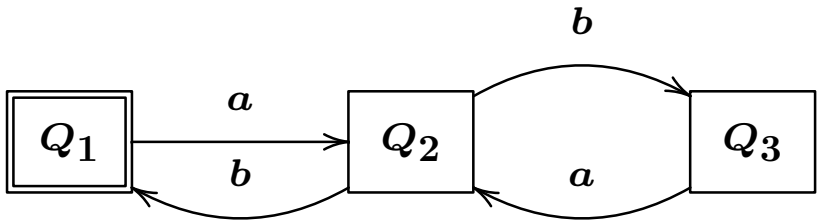
A bisimulation:

$$\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2), (P_1, Q_3)\}$$

All diagrams are ok

Suppose we add a b-transition to $Q_2 \xrightarrow{b} Q_1$:



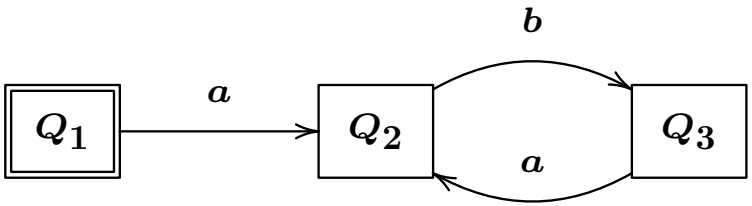


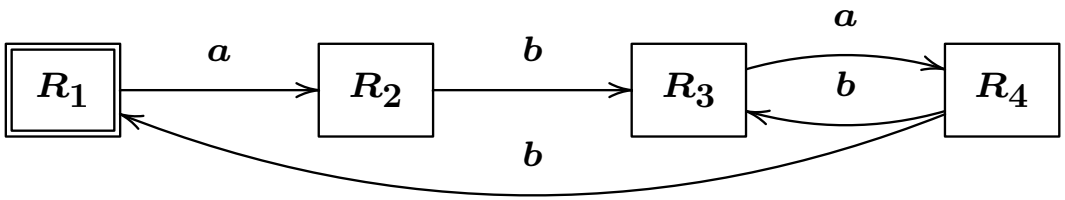
In the original $\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2)\}$ now the diagrams for (P_2, Q_2) look ok:

$$\begin{array}{ccc} P_2 & \mathcal{R} & Q_2 \\ b\downarrow & & b\downarrow \\ P_1 & \mathcal{R} & Q_1 \end{array} \qquad \begin{array}{ccc} P_2 & \mathcal{R} & Q_2 \\ b\downarrow & & b\downarrow \\ P_1 & \mathcal{R} & Q_1 \end{array}$$

$\mathcal{R}$ is still not a bisimulation: why?

Now we want to prove $Q_1 \sim R_1$ (all processes but R_4 are deterministic):





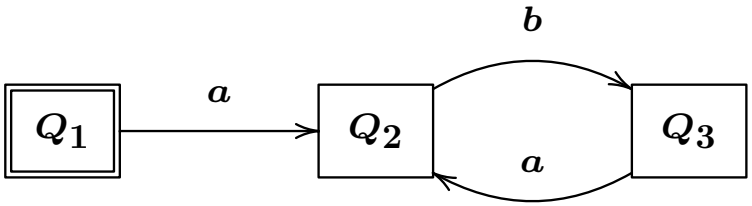
Our initial guess: $\{(Q_1, R_1), (Q_2, R_2), (Q_3, R_3), (Q_2, R_4)\}$

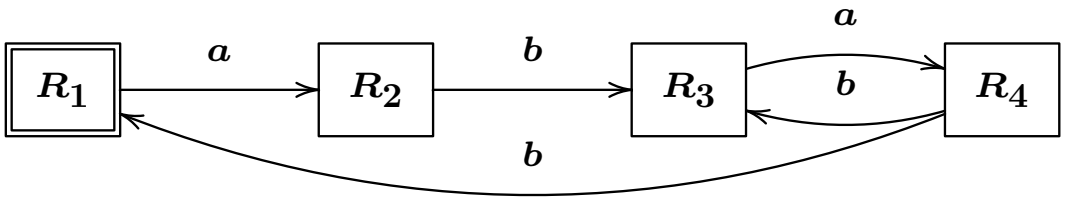
The diagram checks for the first 3 pairs are easy. On (Q_2, R_4):

$$\begin{array}{ccc} Q_2 & \mathcal{R} & R_4 \\ b \downarrow & & b \downarrow \\ Q_3 & \mathcal{R} & R_3 \end{array} \qquad \begin{array}{ccc} Q_2 & \mathcal{R} & R_4 \\ b \downarrow & & b \downarrow \\ Q_3 & \mathcal{R} & R_3 \end{array}$$

One diagram check is missing. Which one?

Now we want to prove $Q_1 \sim R_1$ (all processes but R_4 are deterministic):





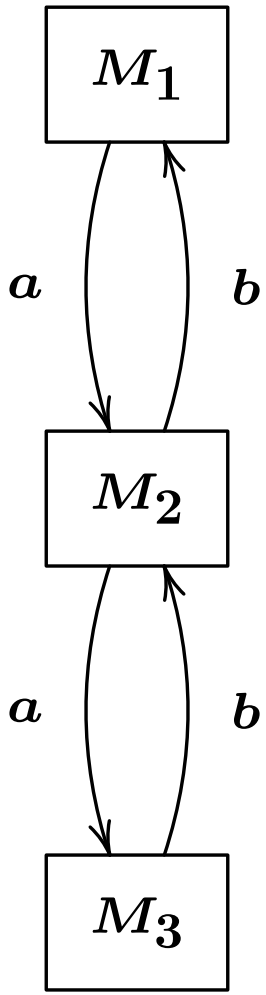
Our initial guess: $\{(Q_1, R_1), (Q_2, R_2), (Q_3, R_3), (Q_2, R_4)\}$

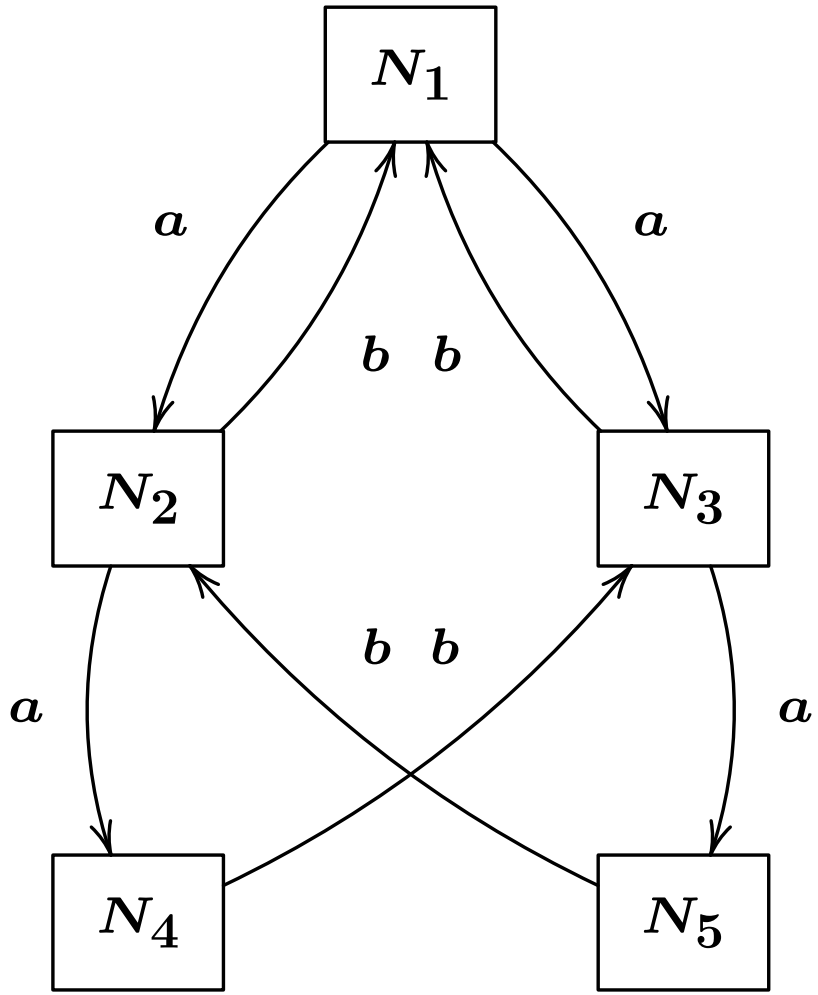
The diagram checks for the first 3 pairs are easy. On (Q_2, R_4):

$$\begin{array}{ccc} Q_2 & \mathcal{R} & R_4 \\ b\downarrow & & b\downarrow \\ Q_3 & \mathcal{R} & R_3 \end{array} \qquad\qquad \begin{array}{ccc} Q_2 & \mathcal{R} & R_4 \\ b\downarrow & & b\downarrow \\ Q_3 & \mathcal{R} & R_3 \end{array}$$

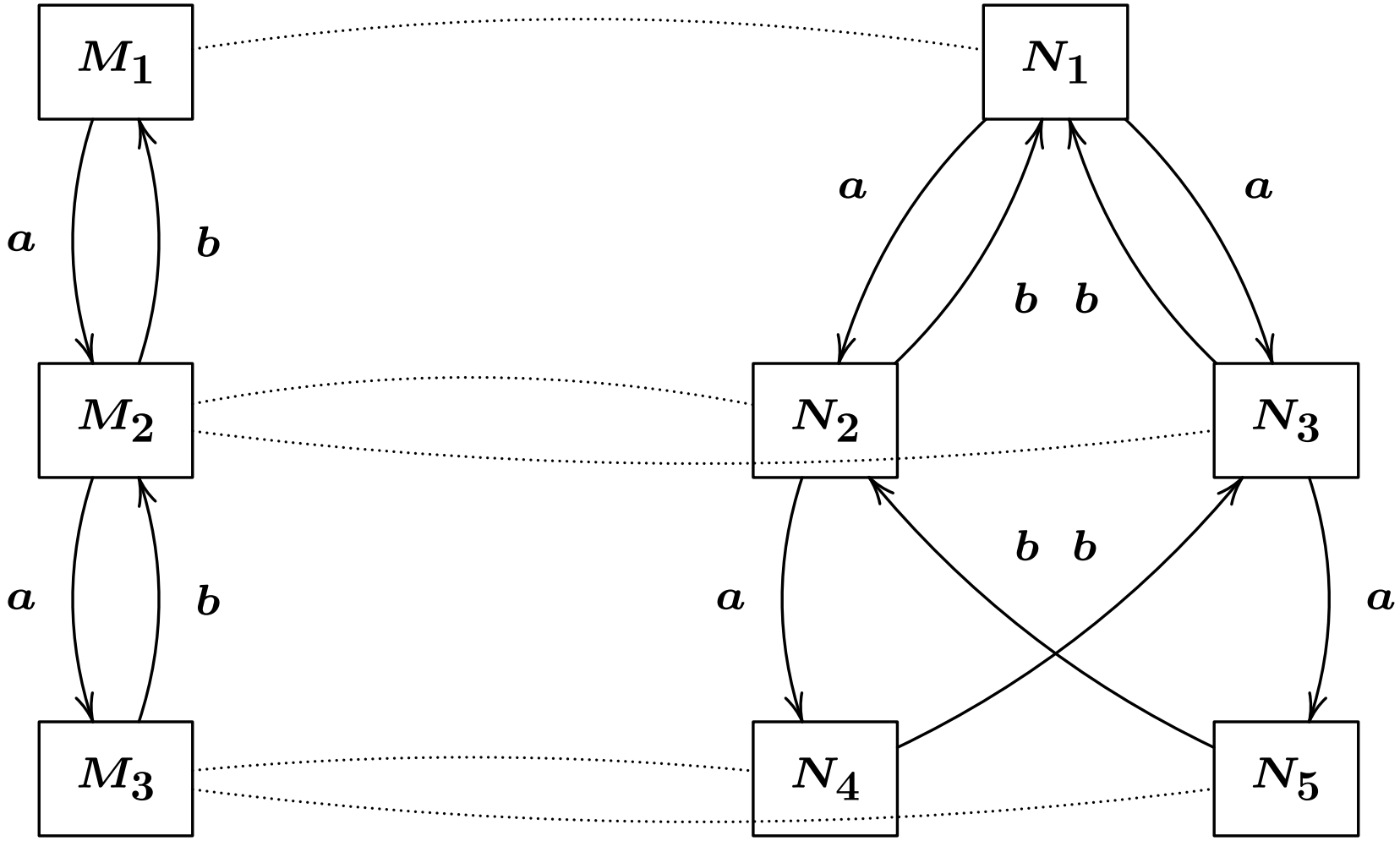
The diagram for $R_4 \xrightarrow{b} R_1$ is missing. Add (Q_3, R_1)

We want to prove $M_1 \sim N_1$:



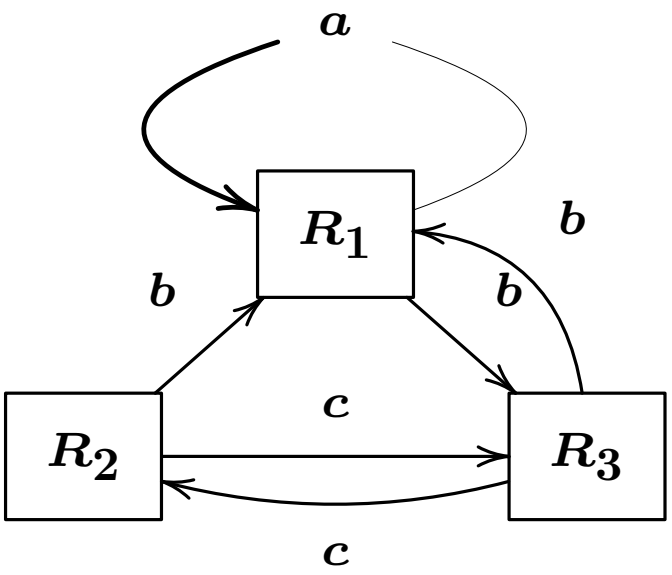


A graphical representation of a bisimulation:

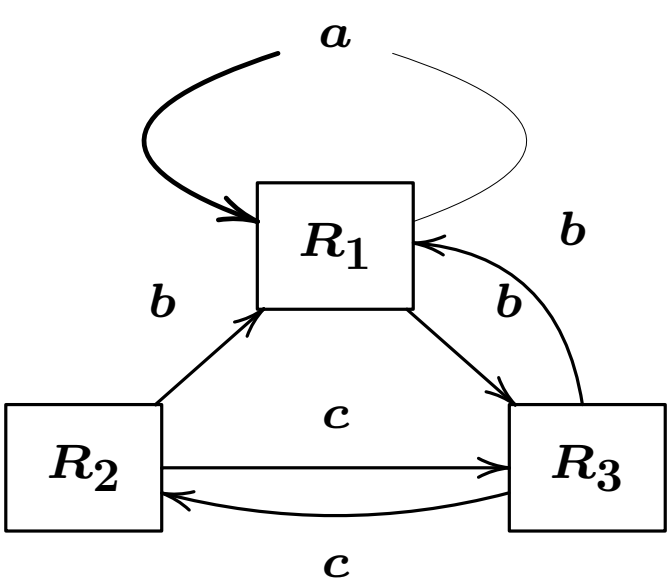


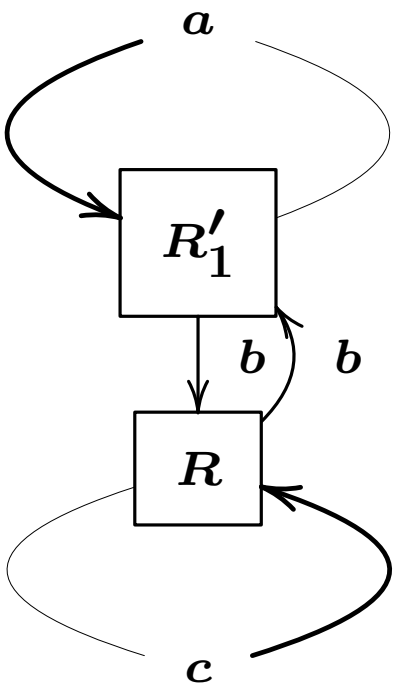
$$\{(M_1, N_1), (M_2, N_2), (M_2, N_3), (M_3, N_4), (M_3, N_5)\}$$

Find an LTS with only two states, and in a bisimulation relation with the states of following LTS:



Take

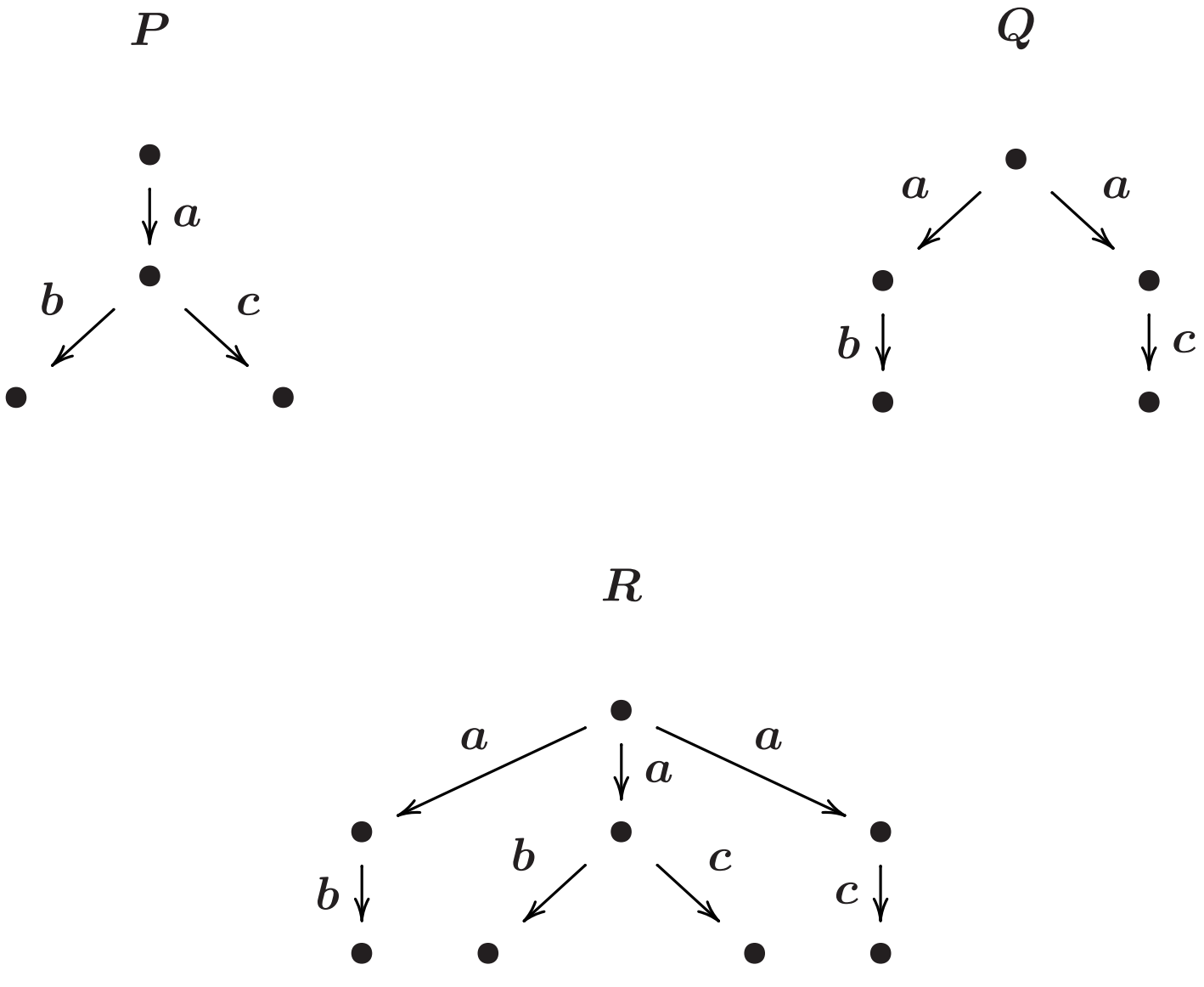


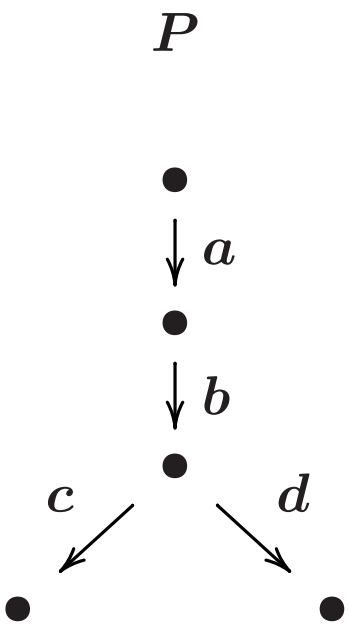


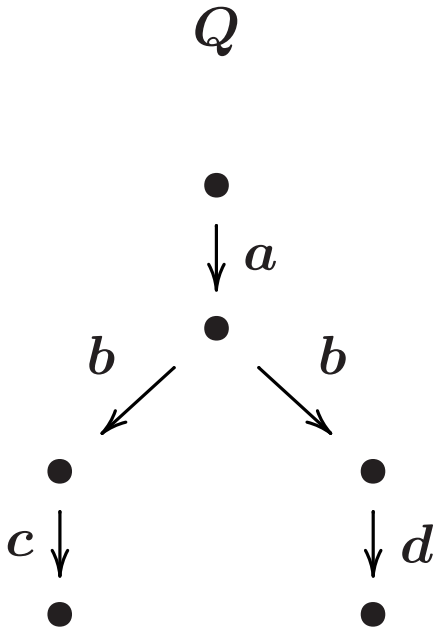
A bisimulation is $\{(R_1, R_1'), (R_2, R), (R_3, R)\}$.

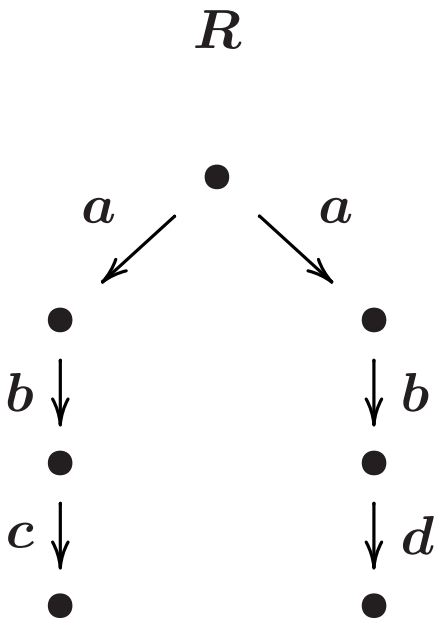
Examples: nondeterminism

Are the following processes bisimilar?




P
a
b
c
d


Q
a
b
b
c
d


R
a
a
b
b
c
d

Basic properties of bisimilarity

Theorem $\sim$ is an equivalence relation, i.e. the following hold:

1. $P \sim P$ (reflexivity)
2. $P \sim Q$ implies $Q \sim P$ (symmetry)
3. $P \sim Q$ and $Q \sim R$ imply $P \sim R$ (transitivity);

Corollary $\sim$ itself is a bisimulation.

Exercise Prove the corollary. You have to show that

$$\cup\{\mathcal{R} \mid \mathcal{R} \text{ is a bisimulation } \}$$

is a bisimulation.

The previous corollary suggests an alternative definition of $\sim$:

Corollary $\sim$ is the largest relation on processes such that $P \sim Q$ implies:

1. $\forall \mu, P'$ s.t. $P \xrightarrow{\mu} P'$, then $\exists Q'$ such that $Q \xrightarrow{\mu} Q'$ and $P' \sim Q'$;
2. $\forall \mu, Q'$ s.t. $Q \xrightarrow{\mu} Q'$, then $\exists P'$ such that $P \xrightarrow{\mu} P'$ and $P' \sim Q'$.

Proof of transitivity

Hp: $P \sim Q$ and $Q \sim R$. Th: $P \sim R$.

For $P \sim R$, we need a bisimulation $\mathcal{R}$ with $P \; \mathcal{R} \; R$.

Since $P \sim Q$ and $Q \sim R$, there are bisimulations $\mathcal{R}_1$ and $\mathcal{R}_2$ with $P \; \mathcal{R}_1 \; Q$ and $Q \; \mathcal{R}_2 \; R$.

Set

$$\mathcal{R} = \{(P, R) \mid \text{there is } Q \text{ with } P \; \mathcal{R}_1 \; Q \text{ and } Q \; \mathcal{R}_2 \; R\}$$

Claim: $\mathcal{R}$ is a bisimulation.

Take $(P, R) \in \mathcal{R}$, because $\exists \, Q$ with $P \; \mathcal{R}_1 \; Q$ and $Q \; \mathcal{R}_2 \; R$, with $P \xrightarrow{a} P'$:

$$\begin{array}{ccccc} P & \mathcal{R}_1 & Q & \mathcal{R}_2 & R \\ \mu \downarrow & & \mu \downarrow & & \mu \downarrow \\ P' & \mathcal{R}_1 & Q' & \mathcal{R}_2 & R' \end{array}$$

Proof of symmetry

First, show that inverse of a bisimulation $\mathcal{R}$ is again a bisimulation:

$$\mathcal{R}^{-1} = \{(P, Q) \mid Q \mathcal{R} P\}$$

Now conclude: If $P \sim Q$, there is a bisimulation $\mathcal{R}$ with $P \mathcal{R} Q$.

We also have $Q \mathcal{R}^{-1} P$ and $\mathcal{R}^{-1}$ is a bisimulation.

Hence $Q \sim P$.

An enhancement of the bisimulation proof method

We write $P \sim \mathcal{R} \sim Q$ if there are P', Q' s.t. $P \sim P'$, $P' \; \mathcal{R} \; Q'$, and $Q' \sim Q$ (and alike for similar notations).

Definition A relation $\mathcal{R}$ on processes is a **bisimulation up-to** $\sim$ if $P \; \mathcal{R} \; Q$ implies:

1. if $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \sim \mathcal{R} \sim Q'$.
2. if $Q \xrightarrow{\mu} Q'$, then there is P' such that $P \xrightarrow{\mu} P'$ and $P' \sim \mathcal{R} \sim Q'$.

Exercise If $\mathcal{R}$ is a bisimulation up-to $\sim$ then $\mathcal{R} \subseteq \sim$. (Hint: prove that $\sim \mathcal{R} \sim$ is a bisimulation.)

Simulation

Definition A relation $\mathcal{R}$ on processes is a **simulation** if $P \mathcal{R} Q$ implies:

1. if $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \mathcal{R} Q'$.

P is *simulated by* Q, written $P < Q$, if $P \mathcal{R} Q$, for some simulation $\mathcal{R}$.

Exercise Does $P \sim Q$ imply $P < Q$ and $Q < P$? What about the converse? (Hint for the second point: think about the 2nd equality at page 26.)

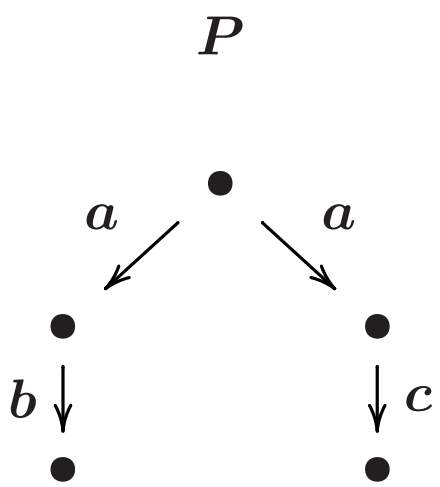
Exercize: quantifiers

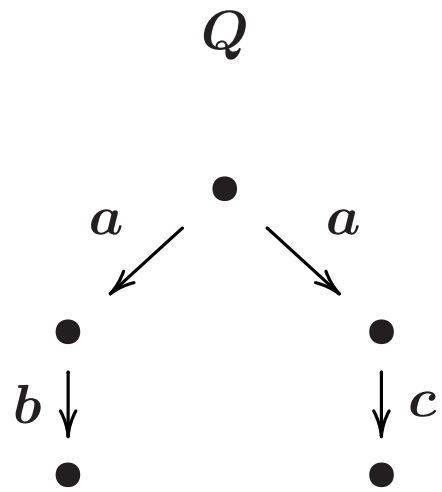
Suppose the existential quantifiers in the definition of bisimulation were replaced by universal quantifiers. For instance, clause (1) would become:

- for all P' with $P \xrightarrow{\mu} P'$, and for all Q' such that $Q \xrightarrow{\mu} Q'$, we have $P' \mathcal{R} Q'$;

and similarly for clause (2).

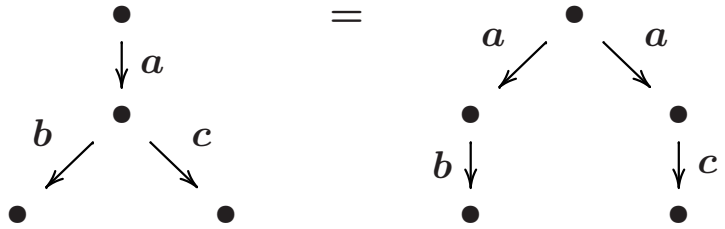
Would these two (identical!) processes be bisimilar? What do you think bisimilarity would become?





Other equivalences: examples

We have seen: trace equivalence has deadlock problems, e.g.,



Besides bisimilarity, many other solutions have been suggested, usually inductive.

Ex: using **decorated traces**, i.e., pairs $a_1 \ldots a_n; S$ of a sequence of actions and a set of action

P has $a_1 \ldots a_n; S$ if:

$\exists R'$ st $P \xrightarrow{a_1} \ldots \xrightarrow{a_n} R'$ and then $R' \xrightarrow{b} \Leftrightarrow b \in S$

The mathematical robustness of bisimilarity and the bisimulation proof method are however major advantages

(i.e., on finite-state processes: bisimilarity is P-space complete, inductive equivalences are PSPACE-complete)

We have seen:

- **the problem of equality between processes**
- **representing behaviours: LTSs**
- **graph theory, automata theory**
- **bisimilarity**
- **the bisimulation proof method**
- **impredicativity (circularity)**

Bisimilarity and the bisimulation proof method:
very different from the the usual, familiar **inductive definitions** and **inductive proofs**.

They are examples of a **coinductive definition** and of a **coinductive proof technique**.

Induction and coinduction

- examples
- duality
- fixed-point theory

Examples of induction and coinduction

Mathematical induction

To prove a property for all natural numbers:

1. Show that **the property holds at 0** (basis)
2. Show that, **whenever the property holds at n, it also holds at $n+1$** (inductive part)

In a variant, step (2) becomes:

Show that, whenever the property holds at all natural less than or equal to n, then it also holds at $n+1$

NB: other variants are possible, modifying for instance the basis

Example of mathematical induction

$$1 + 2 + \ldots + n = \frac{n \times (n+1)}{2}$$

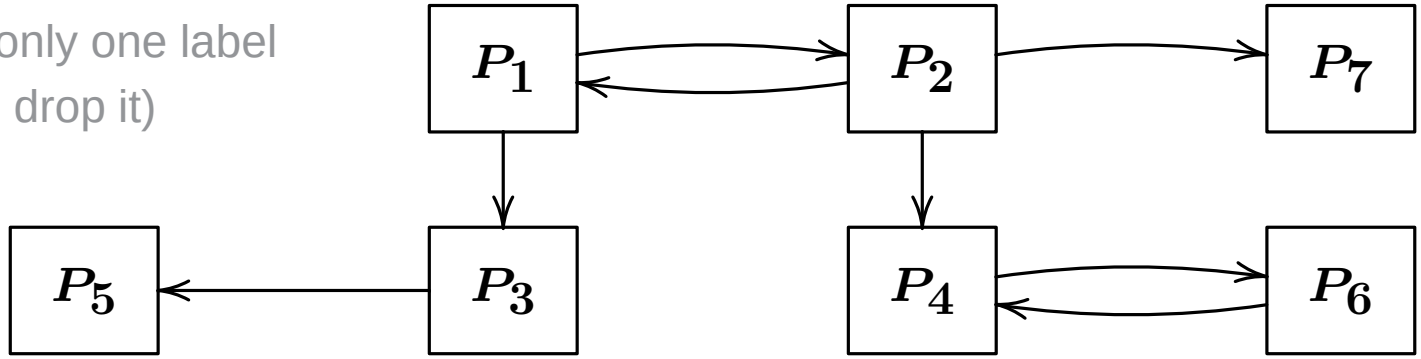
Basis: $1 = \frac{1 \times 2}{2}$

Inductive step: (assume true at n, prove statement for $n+1$)

$$\begin{aligned}
1 + 2 + \ldots + n + (n+1) &= \text{(inductive hypothesis)} \\
\frac{n \times (n+1)}{2} + (n+1) &= \\
\frac{n \times (n+1)}{2} + \frac{2 \times (n+1)}{2} &= \\
\frac{n \times (n+1) + 2 \times (n+1)}{2} &= \\
\frac{(n+1) \times (n+2)}{2} &= \frac{(n+1) \times ((n+1)+1)}{2}
\end{aligned}$$

Rule induction: finite traces (may termination)

(assume only one label
hence we drop it)



A process **stopped**: it cannot do any transitions

P has a **finite trace**, written $P \downarrow$, if P has a finite sequence of transitions that lead to a stopped process

Examples: P_1, P_2, P_3, P_5, P_7 (how many finite traces for P_2?)

(inductive definition of $\downarrow$) $P \downarrow$ if
(1) P is stopped
(2) $\exists\, P'$ with $P \longrightarrow P'$ and $P' \downarrow$

as rules:

$$\frac{P \text{ stopped}}{P \downarrow} \qquad \frac{P \longrightarrow P' \qquad P' \downarrow}{P \downarrow}$$

What is a set inductively defined by a set of rules?

...**later**, using some (very simple) fixed-point and lattice theory

Now: 3 equivalent readings of inductive sets

derived from the definition of inductive sets

(we will show this for one reading)

Equivalent readings for $\downarrow$

$$\frac{P \text{ stopped}}{P\downarrow}\ (\mathrm{AX}) \qquad\qquad \frac{P \longrightarrow P' \qquad P'\downarrow}{P\downarrow}\ (\mathrm{INF})$$

– The processes obtained with a finite proof from the rules

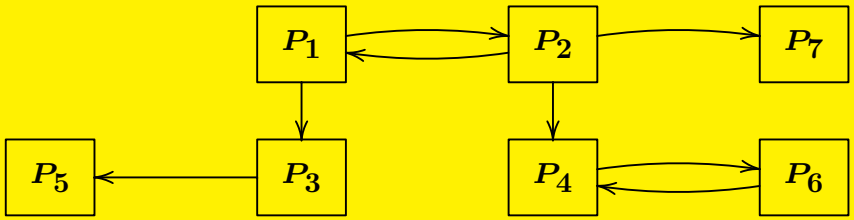
Equivalent readings for $\downarrow$

$$\frac{P \text{ stopped}}{P\downarrow}\ (\text{AX}) \qquad \frac{P \longrightarrow P' \qquad P'\downarrow}{P\downarrow}\ (\text{INF})$$

– The processes obtained with a finite proof from the rules

Example

$$\dfrac{P_1 \longrightarrow P_2 \qquad \dfrac{P_2 \longrightarrow P_7 \qquad \dfrac{P_7 \text{ stopped}}{P_7\downarrow}\ (\text{AX})}{P_2\downarrow}\ (\text{INF})}{P_1\downarrow}\ (\text{INF})$$



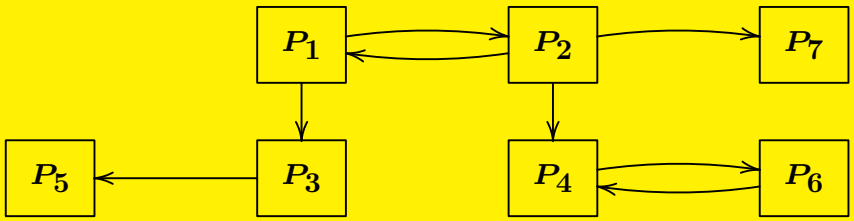
Equivalent readings for $\downarrow$

$$\frac{P \text{ stopped}}{P\downarrow}\ (\text{AX}) \qquad \frac{P \longrightarrow P' \qquad P'\downarrow}{P\downarrow}\ (\text{INF})$$

– The processes obtained with a finite proof from the rules

Example (another proof for P_1; how many other proofs?) :

$$\dfrac{P_1 \longrightarrow P_2 \qquad \dfrac{P_2 \longrightarrow P_1 \qquad \dfrac{P_1 \longrightarrow P_2 \qquad \dfrac{P_2 \longrightarrow P_7 \qquad \dfrac{P_7 \text{ stopped}}{P_7\downarrow}}{P_2\downarrow}}{P_1\downarrow}}{P_2\downarrow}}{P_1\downarrow}$$



Equivalent readings for $\downarrow$

$$\frac{P \text{ stopped}}{P\downarrow}\ (\text{AX}) \qquad\qquad \frac{P \longrightarrow P' \qquad P'\downarrow}{P\downarrow}\ (\text{INF})$$

- The processes obtained with a finite proof from the rules
- the **smallest** set of processes that is **closed forward under the rules**; i.e., the smallest subset S of *Pr* (all processes) such that
  - all stopped processes are in S;
  - if there is P' with $P \longrightarrow P'$ and $P' \in S$, then also $P \in S$.

Equivalent readings for $\downarrow$

$$\frac{P \text{ stopped}}{P\downarrow}\ (\text{AX}) \qquad\qquad \frac{P \longrightarrow P' \qquad P'\downarrow}{P\downarrow}\ (\text{INF})$$

- The processes obtained with a finite proof from the rules
- the **smallest** set of processes that is **closed forward under the rules**;
  i.e., the smallest subset S of *Pr* (all processes) such that
  - all stopped processes are in S;
  - if there is P' with $P \longrightarrow P'$ and $P' \in S$, then also $P \in S$.

Hence a proof technique for $\downarrow$ (rule induction):

given a property T on the processes (a subset of processes),
to prove $\downarrow \subseteq T$ (all processes in $\downarrow$ have the property)
show that T is closed forward under the rules.

Example of rule induction for finite traces

A partial function f, from processes to integers, that satisfies the following conditions:

$$
\begin{array}{lcll}
f(P) & = & 0 & \text{if } P \text{ is stopped} \\
f(P) & = & \mathtt{min}\{f(P') + 1 \mid P \longrightarrow P' \text{ for some } P' & \\
 & & \text{and } f(P') \text{ is defined } \} & \text{otherwise}
\end{array}
$$

(f can have any value, or even be undefined, if the set on which the $\mathtt{min}$ is taken is empty)

We wish to prove f defined on processes with a finite trace (i.e., $\mathtt{dom}(\downarrow) \subseteq \mathtt{dom}(f)$)

We can show that $\mathtt{dom}(f)$ is closed forward under the rules defining $\downarrow$.

Proof:

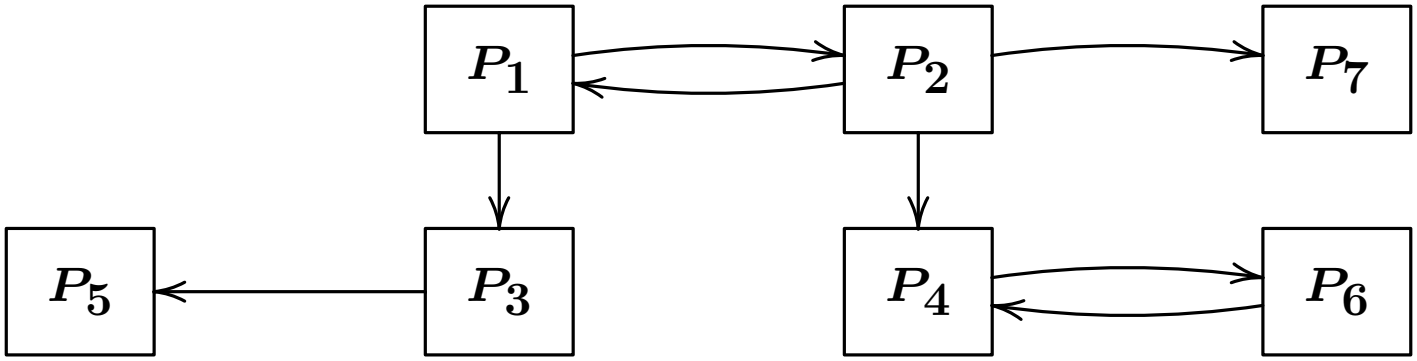
1. $f(P)$ is defined whenever P is stopped;
2. if there is P' with $P \longrightarrow P'$ and $f(P')$ is defined, then also $f(P)$ is defined.

Equivalent readings for $\downarrow$

$$\frac{P \text{ stopped}}{P\downarrow}\ (\text{AX}) \qquad \frac{P \longrightarrow P' \qquad P'\downarrow}{P\downarrow}\ (\text{INF})$$

- The processes obtained with a finite proof from the rules
- the **smallest** set of processes that is **closed forward under the rules**;
  i.e., the smallest subset S of *Pr* (all processes) such that
  - all stopped processes are in S;
  - if there is P' with $P \longrightarrow P'$ and $P' \in S$, then also $P \in S$.
- (iterative construction)
  Start from $\emptyset$;
  add all objects as in the axiom;
  repeat adding objects following the inference rule **forwards**

Rule coinduction definition: ω-traces (non-termination)



P has an **ω-trace**, written $P \upharpoonright$, if it there is an infinite sequence of transitions starting from P.

Examples: P_1, P_2, P_4, P_6

Coinductive definition of $\upharpoonright$:

$$\frac{P \longrightarrow P' \qquad P' \upharpoonright}{P \upharpoonright}$$

Equivalent readings for $\downarrow$

$$\frac{P \longrightarrow P' \qquad P' \downarrow}{P \downarrow}$$

- The processes obtained with an **infinite** proof from the rules

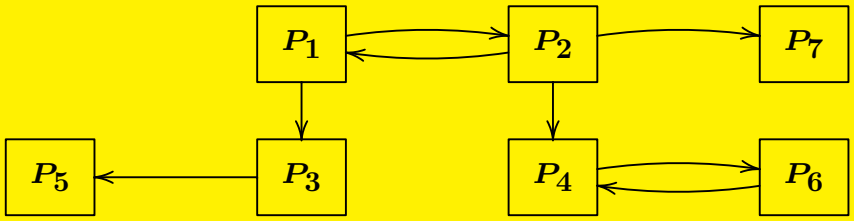
Equivalent readings for $\lfloor$

$$\frac{P \longrightarrow P' \qquad P' \restriction}{P \restriction}$$

– The processes obtained with an **infinite** proof from the rules

Example

$$\dfrac{P_1 \longrightarrow P_2 \qquad \dfrac{P_2 \longrightarrow P_1 \qquad \dfrac{P_1 \longrightarrow P_2 \qquad \dfrac{\vdots}{P_2 \restriction}}{P_1 \restriction}}{P_2 \restriction}}{P_1 \restriction}$$



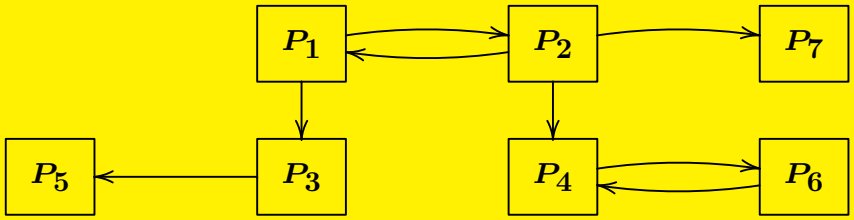
Equivalent readings for $\lfloor$

$$\frac{P \longrightarrow P' \qquad P' \restriction}{P \restriction}$$

– The processes obtained with an **infinite** proof from the rules

An invalid proof:

$$\frac{P_1 \longrightarrow P_3 \qquad \dfrac{P_3 \longrightarrow P_5 \qquad \dfrac{??}{P_5 \restriction}}{P_3 \restriction}}{P_1 \restriction}$$



Equivalent readings for $\downarrow$

$$\frac{P \longrightarrow P' \qquad P' \downarrow}{P \downarrow}$$

- The processes obtained with an **infinite** proof from the rules
- the **largest** set of processes that is **closed backward under the rule**; i.e., the largest subset S of processes such that if $P \in S$ then
  - there is P' such that $P \longrightarrow P'$ and $P' \in S$.

Equivalent readings for $\downarrow$

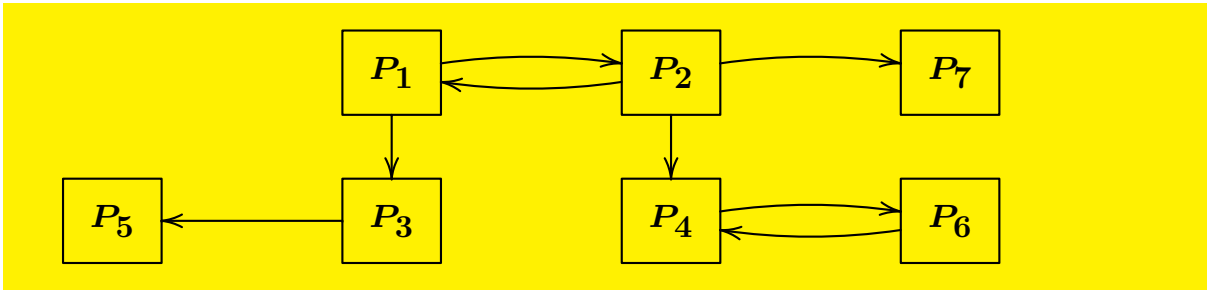
$$\frac{P \longrightarrow P' \qquad P' \uparrow}{P \uparrow}$$

- The processes obtained with an **infinite** proof from the rules
- the **largest** set of processes that is **closed backward under the rule**; i.e., the largest subset S of processes such that if $P \in S$ then
  - there is P' such that $P \longrightarrow P'$ and $P' \in S$.

Hence a proof technique for $\uparrow$ (rule coinduction):

to prove that each process in a set T has an ω-trace
show that T is closed backward under the rule.

Example of rule coinduction for ω-traces



$$\frac{P \longrightarrow P' \qquad P' \restriction}{P \restriction}$$

Suppose we want to prove $P_1 \restriction$

Proof

$T = \{P_1, P_2\}$ is closed backward :

$$\frac{P_1 \longrightarrow P_2 \qquad P_2 \in T}{P_1 \in T} \qquad\qquad \frac{P_2 \longrightarrow P_1 \qquad P_1 \in T}{P_2 \in T}$$

Another choice: $T = \{P_1, P_2, P_4, P_6\}$ (correct, but more work in the proof)

Would $T = \{P_1, P_2, P_4\}$ or $T = \{P_1, P_2, P_3\}$ be correct?

ω-traces in the bisimulation style

A predicate S on processes is **ω-closed** if whenever $P \in S$:

– there is $P' \in S$ such that $P \longrightarrow P'$.

P has an **ω-trace**, written $P \restriction$, if $P \in S$, for some ω-closed predicate S.

The proof technique is explicit

Compare with the definition of bisimilarity:

A relation $\mathcal{R}$ on processes is a **bisimulation** if whenever $P \; \mathcal{R} \; Q$:

1. $\forall \mu, P'$ s.t. $P \xrightarrow{\mu} P'$, then $\exists Q'$ such that $Q \xrightarrow{\mu} Q'$ and $P' \; \mathcal{R} \; Q'$;
2. $\forall \mu, Q'$ s.t. $Q \xrightarrow{\mu} Q'$, then $\exists P'$ such that $P \xrightarrow{\mu} P'$ and $P' \; \mathcal{R} \; Q'$.

P and Q are **bisimilar**, written $P \sim Q$, if $P \; \mathcal{R} \; Q$, for some bisimulation $\mathcal{R}$.

Equivalent readings for $\downarrow$

$$\frac{P \longrightarrow P' \qquad P' \downarrow}{P \downarrow}$$

- The processes obtained with an **infinite** proof from the rules
- the **largest** set of processes that is **closed backward under the rule**; i.e., the largest subset S of processes such that if $P \in S$ then
  - there is $P' \in S$ such that $P \longrightarrow P'$.
- (iterative construction) start with the set *Pr* of all processes; repeatedly remove a process P from the set if one of these applies (the **backward closure** fails):
  - P has no transitions
  - all transitions from P lead to derivatives that are not anymore in the set.

An inductive definition: finite lists over a set A

$$\frac{}{\texttt{nil} \in \mathcal{L}} \qquad \frac{\ell \in \mathcal{L} \qquad a \in A}{\langle a \rangle \bullet \ell \in \mathcal{L}}$$

3 equivalent readings (in the **"forward"** direction):

- The objects obtained with a finite proof from the rules
- The smallest set closed forward under these rules

> A set T is closed forward if: – $\texttt{nil} \in T$
> – $\ell \in T$ implies $\langle a \rangle \bullet \ell \in T$, for all $a \in A$

Inductive proof technique for lists: Let T be a predicate (a property) on lists. To prove that T holds on all lists, prove that T is closed forward

- (iterative construction) Start from $\emptyset$; add all objects as in the axiom; repeat adding objects following the inference rule forwards

A coinductive definition: finite and infinite lists over A

$$\frac{}{\texttt{nil} \in \mathcal{L}} \qquad \frac{\ell \in \mathcal{L} \qquad a \in A}{\langle a \rangle \bullet \ell \in \mathcal{L}}$$

3 equivalent readings (in the **"backward"** direction) :

- The objects that are conclusion of a finite or infinite proof from the rules
- The largest set closed backward under these rules

> A set T is closed backward if $\forall t \in T$:
> – either $t = \texttt{nil}$
> – or $t = \langle a \rangle \bullet \ell$, for some $\ell \in T$ and $a \in A$

Coinduction proof method: to prove that ℓ is a finite or infinite list, find a set D with $\ell \in D$ and D closed backward

- $X =$ all (finite and infinite) strings of $A \cup \{\texttt{nil}, \langle, \rangle, \bullet\}$

Start from X (all strings) and keep removing strings, following the backward-closure

An inductive definition: convergence, in λ-calculus

Set of λ-terms (an inductive def!) $\quad e ::= x \mid \lambda x.\, e \mid e_1(e_2)$

Convergence to a value ($\Downarrow$), on closed λ-terms, call-by-name:

$$\frac{}{\lambda x.\, e \Downarrow \lambda x.\, e} \qquad \frac{e_1 \Downarrow \lambda x.\, e_0 \qquad e_0\{e_2/x\} \Downarrow e'}{e_1(e_2) \Downarrow e'}$$

As before, $\Downarrow$ can be read in terms of finite proofs, limit of an iterative construction, or smallest set closed forward under these rules

$\Downarrow$ is the smallest relation $\mathcal{S}$ on (closed) λ-terms s.t.
- $\lambda x.\, e \;\mathcal{S}\; \lambda x.\, e$ for all abstractions,
- if $e_1 \;\mathcal{S}\; \lambda x.\, e_0$ and $e_0\{e_2/x\} \;\mathcal{S}\; e'$ then also $e_1(e_2) \;\mathcal{S}\; e'$.

A coinductive definition: divergence in the λ-calculus

Divergence ($\Uparrow$), on closed λ-terms, call-by-name:

$$\frac{e_1 \Uparrow}{e_1(e_2) \Uparrow} \qquad \frac{e_1 \Downarrow \lambda x.\, e_0 \qquad e_0\{e_2/x\} \Uparrow}{e_1(e_2) \Uparrow}$$

The 'closed backward' reading:

> $\Uparrow$ is the *largest* predicate on λ-terms that is closed backward under these rules; i.e., the largest subset D of λ-terms s.t. if $e \in D$ then
> – either $e = e_1(e_2)$ and $e_1 \in D$,
> – or $e = e_1(e_2)$, $e_1 \Downarrow \lambda x.\, e_0$ and $e_0\{e_2/x\} \in D$.

Coinduction proof technique :

to prove $e \Uparrow$, find $E \subseteq \Lambda$ closed backward and with $e \in E$

What is the smallest predicate closed backward?

The duality induction/coinduction

Constructors/destructors

- An inductive definition tells us what are the **constructors** for generating all the elements (cf: the forward closure).
- A coinductive definition tells us what are the **destructors** for decomposing the elements (cf: the backward closure).

The destructors show what we can **observe** of the elements
(think of the elements as black boxes;
the destructors tell us what we can do with them;
this is clear in the case of infinite lists).

Definitions given by means of rules

- if the definition is **inductive**, we look for the **smallest** universe in which such rules live.
- if it is **coinductive**, we look for the **largest** universe.
- the **inductive proof principle** allows us to infer that the **inductive set is included in a set** (ie, has a given property) by proving that the set satisfies the **forward closure**;
- the **coinductive proof principle** allows us to infer that **a set is included in the coinductive set** by proving that the given set satisfies the **backward closure**.

Forward and backward closures

A set T being closed forward intuitively means that

> *for each* **rule whose premise is satisfied in T**
> *there is* **an element of T**
> *such that the element is the conclusion of the rule.*

In the backward closure for T, the order between the two quantified entities is swapped:

> *for each* **element of T**
> *there is* **a rule whose premise is satisfied in T**
> *such that the element is the conclusion of the rule.*

In fixed-point theory, the duality between forward and backward closure will the duality between pre-fixed points and post-fixed points.

Congruences vs bisimulation equivalences

Congruence: an equivalence relation
that respects the constructors of a language

Example (λ-calculus)

Consider the following rules, acting on pairs of (open) λ-terms:

$$\frac{}{(x, x)} \qquad \frac{(e_1, e_2)}{(e\ e_1, e\ e_2)} \qquad \frac{(e_1, e_2)}{(e_1\ e, e_2\ e)} \qquad \frac{(e_1, e_2)}{(\lambda x.\, e_1, \lambda x.\, e_2)}$$

A congruence: an equivalence relation closed forward under the rules

The smallest such relation is **syntactic equality**: the **identity relation**

In other words, congruence rules express syntactic constraints

Bisimulation equivalence: an equivalence relation that respects the destructors

Example (λ-calculus, call-by-name)

Consider the following rules

$$\frac{e_1 \Uparrow \qquad e_2 \Uparrow}{(e_1, e_2)} \quad e_1, e_2 \text{ closed}$$

$$\frac{e_1 \Downarrow \lambda x.\, e_1' \qquad e_2 \Downarrow \lambda x.\, e_2' \qquad \cup_{e''} \{(e_1'\{e''/x\}, e_2'\{e''/x\})\}}{(e_1, e_2)} \quad e_1, e_2, e'' \text{ closed}$$

$$\frac{\cup_{\sigma} \{(e_1\sigma, e_2\sigma)\}}{(e_1, e_2)} \quad e_1, e_2 \text{ non closed, } \sigma \text{ closing substitution for } e_1, e_2$$

A bisimulation equivalence: an equivalence relation closed backward under the rules

The largest such relation is **semantic equality**: **bisimilarity**

In other words, the bisimulation rules express semantic constraints

Substitutive relations vs bisimulations

In the duality between congruences and bisimulation equivalences, the equivalence requirement is not necessary.

Leave it aside, we obtaining the duality between **bisimulations** and **substitutive relations**

> a relation is substitutive if whenever s and t are related,
> then any term t' must be related to a term s'
> obtained from t' by replacing occurrences of t with s

Bisimilarity is a congruence

To be useful, a bisimilarity on a term language should be a congruence

This leads to proofs where inductive and coinductive techniques are intertwined

In certain languages, for instance higher-order languages, such proofs may be hard, and how to best combine induction and coinduction remains a research topic.

What makes the combination delicate is that the rules on which congruence and bisimulation are defined — the rules for syntactic and semantic equality — are different.

Summary of the dualities

inductive definition	coinductive definition
induction proof principle	coinduction proof principle
constructors	observations
smallest universe	largest universe
'forward closure' in rules	'backward closure' in rules
congruence	bisimulation equivalence
substitutive relation	bisimulation
identity	bisimilarity
least fixed point	greatest fixed point
pre-fixed point	post-fixed point
algebra	coalgebra
syntax	semantics
semi-decidable set	cosemi-decidable set
strengthening of the candidate in proofs	weakening of the candidate in proofs

We have seen:

- examples of induction and coinduction
- 3 readings for the sets inductively and coinductively obtained from a set of rules
- justifications for the induction and coinduction proof principles
- the duality between induction and coinduction, informally

Remaining questions

- **What is the definition of an inductive set?**
- **From this definition, how do we derive the previous 3 readings for sets inductively and coinductively obtained from a set of rules?**
- **How is the duality induction/coinduction formalised?**

What follows answers these questions. It is a simple application of fixed-point theory on complete lattices.

To make things simpler, we work on **powersets** and **fixed-point theory**.
(It is possible to be more general, working with universal algebras or category theory.)

Complete lattices and fixed-points

Complete lattices

The important example of complete lattice for us: **powersets**.

For a given set X, the powerset of X, written $\wp(X)$, is

$$\wp(X) \stackrel{\text{def}}{=} \{T \mid T \subseteq X\}$$

$\wp(X)$ is a complete lattice because:

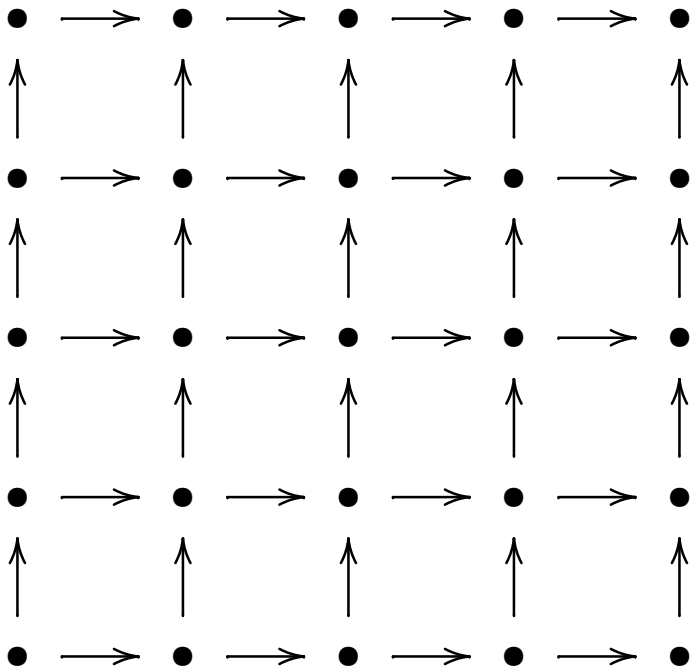
- it comes with a relation $\subseteq$ (set inclusion) that is reflexive, transitive, and antisymmetric.
- it is closed under union and intersection

($\cup$ and $\cap$ give least upper bounds and greatest lower bounds for $\subseteq$)

A **partially ordered set** (or **poset**): a non-empty set with a relation on its elements that is reflexive, transitive, and antisymmetric.

A **complete lattice**: a poset with all joins (least upper bounds) and (hence) also all meets (greatest lower bounds).

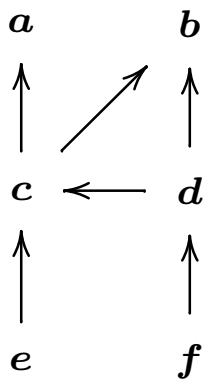
Example of a complete lattice



Two points x, y are in the relation $\leq$ if there is a path from x to y following the directional edges

(a path may also be empty, hence $x \leq x$ holds for all x)

A partially ordered set that is not a complete lattice



Again, $x \leq y$ if there is a path from x to y

The Fixed-point Theorem

NB: Complete lattices are “dualisable” structures: reverse the arrows and you get another complete lattice. Similarly, statements on complete lattices can be dualised.

For simplicity, we will focus on complete lattices produced by the powerset construction. But all statements can be generalised to arbitrary complete lattices

Given a function F on a complete lattice:

- F is **monotone** if $x \leq y$ implies $F(x) \leq F(y)$, for all x, y.
- x is a **pre-fixed point** of F if $F(x) \leq x$.
  Dually, x is a **post-fixed point** if $x \leq F(x)$.
- x is a **fixed point** of F if $F(x) = x$ (it is both pre- and post-fixed point)
- The set of fixed points of F may have a least element, the **least fixed point**, and a greatest element, the **greatest fixed point**

Theorem [Fixed-point Theorem] If $F : \wp(X) \to \wp(X)$ is monotone, then

$$\texttt{lfp}(F) = \bigcap \{T \mid F(T) \subseteq T\}$$

$$\texttt{gfp}(F) = \bigcup \{T \mid T \subseteq F(T)\}$$

(the meet of the pre-fixed points, the join of the post-fixed points)

NB: the theorem actually says more: the set of fixed points is itself a complete lattice, and the same for the sets of pre-fixed points and post-fixed points.

Proof of the Fixed-point Theorem

We consider one part of the statement (the other part is dual), namely

$$\texttt{gfp}(F) = \bigcup\{S \mid S \subseteq F(S)\}$$

Set $T = \bigcup\{S \mid S \subseteq F(S)\}$. We have to show T fixed point (it is then the greatest: any other fixed point is a post-fixed point, hence contained in T)

Proof of $T \subseteq F(T)$

For each S s.t. $S \subseteq F(S)$ we have:

	$S \subseteq T$	(def of T as a union)
hence	$F(S) \subseteq F(T)$	(monotonicity of F)
hence	$S \subseteq F(T)$	(since S is a post-fixed point)

We conclude $F(T) \supseteq \bigcup\{S \mid S \subseteq F(S)\} = T$

Proof of the Fixed-point Theorem

We consider one part of the statement (the other part is dual), namely

$$\texttt{gfp}(F) = \bigcup\{S \mid S \subseteq F(S)\}$$

Set $T = \bigcup\{S \mid S \subseteq F(S)\}$. We have to show T fixed point (it is then the greatest: any other fixed point is a post-fixed point, hence contained in T)

Proof of $F(T) \subseteq T$

We have	$T \subseteq F(T)$	(just proved)
hence	$F(T) \subseteq F(F(T))$	(monotonicity of F)
that is,	$F(T)$ is a post-fixed point	

Done, by definition of T as a union of the post-fixed points.

Sets coinductively and inductively defined by F

Definition Given a complete lattice produced by the powerset construction, and an endofunction F on it, the sets:

$$F_{\mathtt{ind}} \stackrel{\text{def}}{=} \bigcap \{x \mid F(x) \subseteq x\}$$

$$F_{\mathtt{coind}} \stackrel{\text{def}}{=} \bigcup \{x \mid x \subseteq F(x)\}$$

are the sets **inductively defined by F**, and **coinductively defined by F**.

By the Fixed-point Theorem, when F monotone:

$$\begin{aligned} F_{\mathtt{ind}} &= \mathtt{lfp}(F) \\ &= \text{least pre-fixed point of } F \end{aligned}$$

$$\begin{aligned} F_{\mathtt{coind}} &= \mathtt{gfp}(F) \\ &= \text{greatest post-fixed point of } F \end{aligned}$$

It remains to show:

a set of rules	$\Leftrightarrow$	a monotone function on a complete lattice
a forward closure for the rules	$\Leftrightarrow$	a pre-fixed point for the function
a backward closure for the rules	$\Leftrightarrow$	a post-fixed point for the function

NB: all inductive and coinductive definitions can be given in terms of rules

Definitions by means of rules

Given a set X, a **ground rule on X** is a pair (S, x) with $S \subseteq X$ and $x \in X$

We can write a rule (S, x) as

$$\frac{x_1 \dots x_n \dots}{x} \qquad \text{where } \{x_1, \dots, x_n, \dots\} = S.$$

A rule $(\emptyset, x)$ is an **axiom**

Definitions by means of rules

Given a set X, a **ground rule on X** is a pair (S, x) with $S \subseteq X$ and $x \in X$

We can write a rule (S, x) as

$$\frac{x_1 \ldots x_n \ldots}{x} \qquad \text{where } \{x_1, \ldots, x_n, \ldots\} = S.$$

A rule $(\emptyset, x)$ is an **axiom**

NB: previous rules, eg $\dfrac{P \longrightarrow P' \qquad P' \restriction}{P \restriction}$ were not ground (P, P' are metavariables)

The translation to ground rules is trivial (take all valid instantiations)

Definitions by means of rules

Given a set X, a **ground rule on X** is a pair (S, x) with $S \subseteq X$ and $x \in X$

We can write a rule (S, x) as

$$\frac{x_1 \dots x_n \dots}{x} \qquad \text{where } \{x_1, \dots, x_n, \dots\} = S.$$

A rule $(\emptyset, x)$ is an **axiom**

A set $\mathcal{R}$ of rules on X yields a monotone endofunction $\Phi_{\mathcal{R}}$, called the **functional of $\mathcal{R}$** (or **rule functional**), on the complete lattice $\wp(X)$, where

$$\Phi_{\mathcal{R}}(T) = \{x \mid (T', x) \in \mathcal{R} \text{ for some } T' \subseteq T\}$$

Exercise Show $\Phi_{\mathcal{R}}$ monotone, and that every monotone operator on $\wp(X)$ can be expressed as the functional of some set of rules.

By the Fixed-point Theorem there are least fixed point and greatest fixed point, $\mathtt{lfp}(\Phi_{\mathcal{R}})$ and $\mathtt{gfp}(\Phi_{\mathcal{R}})$, obtained via the join and meet in the theorem.

They are indeed called the sets **inductively** and **coinductively defined by the rules**.

Thus indeed:

a set of rules $\Leftrightarrow$ a monotone function on a complete lattice

Next: pre-fixed points and forward closure (and dually)

What does it mean $\Phi_{\mathcal{R}}(T) \subseteq T$ (ie, set T is a pre-fixed point of $\Phi_{\mathcal{R}}$)?

As $\Phi_{\mathcal{R}}(T) = \{x \mid (S, x) \in \mathcal{R} \text{ for some } S \subseteq T\}$ it means:

for all rules $(S, x) \in \mathcal{R}$,
if $S \subseteq T$ (so that $x \in \Phi_{\mathcal{R}}(T)$), **then also $x \in T$.**

That is:

(i) the conclusions of each axiom is in T;

(ii) each rule whose premises are in T has also the conclusion in T.

This is precisely the 'forward' closure in previous examples.

The Fixed-point Theorem tells us that the least fixed point is the least pre-fixed point: the set inductively defined by the rules is therefore the smallest set closed forward.

For rules, the induction proof principle, in turn, says:

> for a given T,
> if for all rules $(S, x) \in \mathcal{R}$, $S \subseteq T$ implies $x \in T$
> then (the set inductively defined by the rules) $\subseteq T$.

As already seen discussing the forward closure, this is the familiar way of reasoning inductively on rules.

(the assumption "$S \subseteq T$" is the **inductive hypothesis**; the base of the induction is given by the axioms of $\mathcal{R}$)

We have recovered the **principle of rule induction**

Now the case of coinduction. Set T is a post-fixed if

$T \subseteq \Phi_{\mathcal{R}}(T)$, where $\Phi_{\mathcal{R}}(T) = \{x \mid (T', x) \in \mathcal{R} \text{ for some } T' \subseteq T\}$

This means:

for all $t \in T$ there is a rule $(S, t) \in \mathcal{R}$ with $S \subseteq T$

This is precisely the 'backward' closure

By Fixed-point Theory, the set coinductively defined by the rules is the largest set closed backward.

The coinduction proof principle reads thus (**principle of rule coinduction**):

for a given T,
if for all $x \in T$ there is a rule $(S, x) \in \mathcal{R}$ with $S \subseteq T$,
then $T \subseteq$ (the set coinductive defined by the rules)

Exercise Let $\mathcal{R}$ be a set of ground rules, and suppose each rule has a non-empty premise. Show that $\mathtt{lfp}(\Phi_{\mathcal{R}}) = \emptyset$.

The examples, revisited

- the previous examples of rule induction and coinduction reduced to the fixed-point format
- other induction principles reduced to rule induction

Finite traces

$$\frac{P \text{ stopped}}{P\downarrow} \qquad \frac{P \xrightarrow{\mu} P' \qquad P'\downarrow}{P\downarrow}$$

As ground rules, these become:

$$\begin{aligned}\mathcal{R}_{\downarrow} \stackrel{\text{def}}{=} \; & \{(\emptyset, P) \mid P \text{ is stopped}\} \\ & \bigcup\{(\{P'\}, P) \mid P \xrightarrow{\mu} P' \text{ for some } \mu\}\end{aligned}$$

This yields the following functional:

$$\Phi_{\mathcal{R}_{\downarrow}}(T) \stackrel{\text{def}}{=} \{P \mid P \text{ is stopped, or there are } P', \mu \text{ with } P' \in T \text{ and } P \xrightarrow{\mu} P'\}$$

The sets ‘closed forward’ are the pre-fixed points of $\Phi_{\mathcal{R}_{\downarrow}}$.

Thus the smallest set closed forward and the associated proof technique become examples of inductively defined set and of induction proof principle.

ω-traces

$$\frac{P \xrightarrow{\mu} P' \qquad P' \restriction}{P \restriction}$$

As ground rules, this yields:

$$\mathcal{R}_{\restriction} \stackrel{\text{def}}{=} \{(\{P'\}, P) \mid P \xrightarrow{\mu} P'\} .$$

This yields the following functional:

$$\Phi_{\mathcal{R}_{\restriction}}(T) \stackrel{\text{def}}{=} \{P \mid \text{ there is } P' \in T \text{ and } P \xrightarrow{\mu} P'\}$$

Thus the sets ‘closed backward’ are the post-fixed points of $\Phi_{\mathcal{R}_{\restriction}}$, and the largest set closed backward is the greatest fixed point of $\Phi_{\mathcal{R}_{\restriction}}$;

Similarly, the proof technique for ω-traces is derived from the coinduction proof principle.

Finite lists (finLists)

The rule functional (from sets to sets) is:

$$\boldsymbol{F}(\boldsymbol{T}) \stackrel{\text{def}}{=} \{\mathtt{nil}\} \cup \{\langle \boldsymbol{a} \rangle \bullet \boldsymbol{\ell} \mid \boldsymbol{a} \in \boldsymbol{A}, \boldsymbol{\ell} \in \boldsymbol{T}\}$$

$\boldsymbol{F}$ is monotone, and $\mathrm{finLists} = \mathtt{lfp}(\boldsymbol{F})$. (i.e., finLists is the smallest set solution to the equation $\boldsymbol{\mathcal{L}} = \mathtt{nil} + \langle \boldsymbol{A} \rangle \bullet \boldsymbol{\mathcal{L}}$).

From the induction and coinduction principles, we infer: Suppose $\boldsymbol{T} \subseteq \mathrm{finLists}$. If $\boldsymbol{F}(\boldsymbol{T}) \subseteq \boldsymbol{T}$ then $\boldsymbol{T} \subseteq \mathrm{finLists}$ (hence $\boldsymbol{T} = \mathrm{finLists}$).

Proving $\boldsymbol{F}(\boldsymbol{T}) \subseteq \boldsymbol{T}$ requires proving

- $\mathtt{nil} \in \boldsymbol{T}$;
- $\boldsymbol{\ell} \in \mathrm{finLists} \cap \boldsymbol{T}$ implies $\langle \boldsymbol{a} \rangle \bullet \boldsymbol{\ell} \in \boldsymbol{T}$, for all $\boldsymbol{a} \in \boldsymbol{A}$.

This is the same as the familiar induction technique for lists

λ-calculus

In the case of $\Downarrow$, the rules manipulate pairs of closed λ-terms, thus they act on the set $\Lambda^0 \times \Lambda^0$. The rule functional for $\Downarrow$, written $\Phi_{\Downarrow}$, is

$$\begin{aligned}\Phi_{\Downarrow}(T) \stackrel{\text{def}}{=} & \{(e,e') \mid e = e' = \lambda x.\, e'' \text{ , for some } e'' \} \\ & \bigcup \{(e,e') \mid e = e_1\, e_2 \text{ and} \\ & \qquad \exists\, e_0 \text{ such that } (e_1, \lambda x.\, e_0) \in T \text{ and } (e_0\{^{e_2}/_x\}, e') \in T\} \,.\end{aligned}$$

In the case of $\Uparrow$, the rules are on Λ^0. The rule functional for $\Uparrow$ is

$$\begin{aligned}\Phi_{\Uparrow}(T) \stackrel{\text{def}}{=} & \{e_1\, e_2 \mid e_1 \in T,\} \\ & \bigcup \{e_1\, e_2 \mid e_1 \Downarrow \lambda x.\, e_0 \text{ and } e_0\{^{e_2}/_x\} \in T\}.\end{aligned}$$

Mathematical induction

The rules (on the set $\{0, 1, \ldots\}$ of natural numbers or any set containing the natural numbers) are:

$$\frac{}{0} \qquad \frac{n}{n+1} \quad \text{(for all } n \geq 0\text{)}$$

The natural numbers: the least fixed point of a rule functional.

Principle of rule induction: if a property on the naturals holds at 0 and, whenever it holds at n, it also holds at $n+1$, then the property is true for all naturals.

This is the ordinary **mathematical induction**

A variant induction on the natural numbers: the inductive step assumes the property at all numbers less than or equal to n

$$\frac{}{0} \qquad \frac{0, 1, \ldots, n}{n+1} \text{ (for all } n \geq 0)$$

These are the ground-rule translation of this (open) rule, where S is a property on the natural numbers:

$$\frac{i \in S\,,\ \forall\, i < j}{j \in S}$$

Well-founded induction

Given a well-founded relation $\mathcal{R}$ on a set X, and a property T on X, to show that $X \subseteq T$ (the property T holds at all elements of X), it suffices to prove that, for all $x \in X$: if $y \in T$ for all y with $y \mathcal{R} x$, then also $x \in T$.

mathematical induction, structural induction can be seen as special cases

Well-founded induction is indeed the natural generalisation of mathematical induction to sets and, as such, it is frequent to find it in Mathematics and Computer Science.

Example: proof of a property reasoning on the lexicographical order on pairs of natural numbers

We can derive well-founded induction from fixed-point theory in the same way as we did for rule induction.

In fact, we can reduce well-founded induction to rule induction taking as rules, for each $x \in X$, the pair (S, x) where S is the set $\{y \mid y \mathrel{\mathcal{R}} x\}$ and $\mathcal{R}$ the well-founded relation.

Note that the set inductively defined by the rules is precisely X; that is, any set equipped with a well-founded relation is an inductive set.

Transfinite induction

The extension of mathematical induction to ordinals

Transfinite induction says that to prove that a property T on the ordinals holds at all ordinals, it suffices to prove, for all ordinals α: if $\beta \in T$ for all ordinals $\beta < \alpha$ then also $\alpha \in T$.

In proofs, this is usually split into three cases:

(i) $0 \in T$;

(ii) for each ordinal α, if $\alpha \in T$ then also $\alpha + 1 \in T$;

(iii) for each limit ordinal β, if $\alpha \in T$ for all $\alpha < \beta$ then also $\beta \in T$.

Transfinite induction acts on the ordinals, which form a proper class rather than a set.

As such, we cannot derive it from the fixed-point theory presented.

However, in practice, transfinite induction is used to reason on sets, in cases where mathematical induction is not sufficient because the set has ’too many’ elements.

In these cases, in the transfinite induction each ordinal is associated to an element of the set. Then the $<$ relation on the ordinals is a well-founded relation on a set, so that transfinite induction becomes a special case of well-founded induction on sets.

Another possibility: lifting the theory of induction to classes.

Other examples

Structural induction

Induction on derivation proofs

Transition induction

...

Back to bisimulation

- bisimilarity as a fixed point
- bisimulation outside concurrency: equality on coinductive data

Bisimulation as a fixed-point

Definition Consider the following function $F_\sim : \wp(Pr \times Pr) \to \wp(Pr \times Pr)$. $F_\sim(\mathcal{R})$ is the set of all pairs (P, Q) s.t.:

1. $\forall \mu, P'$ s.t. $P \xrightarrow{\mu} P'$, then $\exists Q'$ such that $Q \xrightarrow{\mu} Q'$ and $P' \; \mathcal{R} \; Q'$;
2. $\forall \mu, Q'$ s.t. $Q \xrightarrow{\mu} Q'$, then $\exists P'$ such that $P \xrightarrow{\mu} P'$ and $P' \; \mathcal{R} \; Q'$.

Proposition We have:

- $F_\sim$ is monotone;
- $\mathcal{R}$ is a bisimulation iff $\mathcal{R} \subseteq F_\sim(\mathcal{R})$;
- $\sim \; = \mathtt{gfp}(F_\sim)$.

Equality on coinductive data types

On infinite lists (more generally coinductively defined sets) proving equality may be delicate (they can be “infinite objects”, hence one cannot proceed inductively, eg on their depth)

We can prove equalities adapting the idea of bisimulation.

The coinductive definition tells us what can be observed

We make this explicit in lists defining an LTS thus:

$$\langle a \rangle \bullet s \xrightarrow{a} s$$

Lemma On finite-infinite lists, $s = t$ if and only if $s \sim t$.

Of course it is not necessary to define an LTS from lists.

We can directly define a kind of bisimulation on lists, as follows:

A relation $\mathcal{R}$ on lists is a **list bisimulation** if whenever $(s, t) \in \mathcal{R}$ then

1. $s = \texttt{nil}$ implies $t = \texttt{nil}$;
2. $s = \langle a \rangle \bullet s'$ implies there is t' such that $t = \langle a \rangle \bullet t'$ and $(s', t') \in \mathcal{R}$

Then **list bisimilarity** as the union of all list bisimulations.

To see how natural is the bisimulation method on lists, consider the following characterisation of equality between lists:

$$\frac{}{\mathtt{nil} = \mathtt{nil}} \qquad \frac{s_1 = s_2 \qquad a \in A}{\langle a \rangle \bullet s_1 = \langle a \rangle \bullet s_2}$$

The inductive interpretation of the rules gives us equality on finite lists, as the least fixed point of the corresponding rule functional.

The coinductive interpretation gives us equality on finite-infinite lists, and list bisimulation as associated proof technique.

To see this, it suffices to note that the post-fixed points of the rule functional are precisely the list bisimulations; hence the greatest fixed point is list bisimilarity and, by the previous Lemma, it is also the equality relation.

Example

$$\begin{aligned} \texttt{map}\ f\ \texttt{nil} &= \texttt{nil} \\ \texttt{map}\ f\ (\langle a \rangle \bullet s) &= \langle f(a) \rangle \bullet \texttt{map}\ f\ s \end{aligned}$$

$$\texttt{iterate}\ f\ a = \langle a \rangle \bullet \texttt{iterate}\ f\ f(a)$$

Thus `iterate` $f\ a$ builds the infinite list

$$\langle a \rangle \bullet \langle f(a) \rangle \bullet \langle f(f(a)) \rangle \bullet \ldots$$

Show that, for all $a \in A$:

$$\texttt{map}\ f\ (\texttt{iterate}\ f\ a) = \texttt{iterate}\ f\ f(a)$$

Proof

$$\mathcal{R} \stackrel{\text{def}}{=} \{(\texttt{map}\ f\ (\texttt{iterate}\ f\ a), \texttt{iterate}\ f\ f(a))\ \mid\ a \in A\}$$

is a bisimulation. Let $(P, Q) \in \mathcal{R}$, for $P \stackrel{\text{def}}{=} \texttt{map}\ f\ (\texttt{iterate}\ f\ a)$

$Q \stackrel{\text{def}}{=} \texttt{iterate}\ f\ f(a)$

Applying the definitions of `iterate`, and of LTS

$$\begin{array}{rcl} Q & = & \langle f(a) \rangle \bullet \texttt{iterate}\ f\ f(f(a)) \\ & \xrightarrow{f(a)} & \texttt{iterate}\ f\ f(f(a)) \stackrel{\text{def}}{=} Q'. \end{array}$$

Similarly,
$$\begin{array}{rcl} P & = & \texttt{map}\ f\ \langle a \rangle \bullet (\texttt{iterate}\ f\ f(a)) \\ & = & \langle f(a) \rangle \bullet \texttt{map}\ f\ (\texttt{iterate}\ f\ f(a)) \\ & \xrightarrow{f(a)} & \texttt{map}\ f\ (\texttt{iterate}\ f\ f(a)) \\ & \stackrel{\text{def}}{=} & P' \end{array}$$

We have $P'\ \mathcal{R}\ Q'$, as $f(a) \in A$.

Done (we have showed that P and Q have a single transition, with same labels, and with derivatives in $\mathcal{R}$)

CCS: a process calculus

Some simple process operators (from CCS)

$$P ::= P_1 \mid P_2 \;\Big|\; P_1 + P_2 \;\Big|\; \mu.\, P \;\Big|\; (\nu a)\, P \;\Big|\; 0 \;\Big|\; K$$

where K is a **constant**

Nil, written 0 : a terminated process, no transitions

Prefixing (action sequentialisation)

$$\frac{}{\mu.\, P \xrightarrow{\mu} P}\ \textsc{Pre}$$

Parallel composition

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \mid P_2 \xrightarrow{\mu} P_1' \mid P_2}\ \textsc{ParL}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 \mid P_2 \xrightarrow{\mu} P_1 \mid P_2'}\ \textsc{ParR}$$

$$\frac{P_1 \xrightarrow{\mu} P_1' \qquad P_2 \xrightarrow{\overline{\mu}} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}\ \textsc{Com}$$

As an example, the process $P \stackrel{\text{def}}{=} (a.0 \mid b.0) \mid \overline{a}.0$ has the transitions

$$P \xrightarrow{a} (0 \mid b.0) \mid \overline{a}.0 \qquad P \xrightarrow{\tau} (0 \mid b.0) \mid 0$$
$$P \xrightarrow{b} (a.0 \mid 0) \mid \overline{a}.0 \qquad P \xrightarrow{\overline{a}} (a.0 \mid b.0) \mid 0$$

Choice

$$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 + P_2 \xrightarrow{\mu} P_1'}\ \textsc{SumL}$$

$$\frac{P_2 \xrightarrow{\mu} P_2'}{P_1 + P_2 \xrightarrow{\mu} P_2'}\ \textsc{SumR}$$

As an example, the process $P \stackrel{\text{def}}{=} (\overline{a}.\, Q_1 \mid a.\, Q_2) + b.\, R$ has the transitions

$$P \xrightarrow{\tau} Q_1 \mid Q_2 \qquad P \xrightarrow{a} \overline{a}.\, Q_1 \mid Q_2$$
$$P \xrightarrow{\overline{a}} Q_1 \mid a.\, Q_2 \qquad P \xrightarrow{b} R$$

Constants (Recursive process definitions)

Each constant K has a behaviour specified by a set of transitions of the form $K \xrightarrow{\mu} P$.

Example: $K \xrightarrow{a} K$

A specification and an implementation of a counter

Take constants $\mathtt{Counter}_n$, for $n \geq 0$, with transitions

$$\mathtt{Counter}_0 \xrightarrow{\mathtt{up}} \mathtt{Counter}_1$$

and, for $n > 0$,

$$\mathtt{Counter}_n \xrightarrow{\mathtt{up}} \mathtt{Counter}_{n+1} \qquad \mathtt{Counter}_n \xrightarrow{\mathtt{down}} \mathtt{Counter}_{n-1} \ .$$

The initial state is $\mathtt{Counter}_0$

An implementation of the counter in term of a constant $\mathtt{C}$ with transition

$$\mathtt{C} \xrightarrow{\mathtt{up}} \mathtt{C} \mid \mathtt{down}.\mathbf{0} \ .$$

We want to show: $\mathtt{Counter}_0 \sim \mathtt{C}$

Proof

$$\mathcal{R} \stackrel{\text{def}}{=} \{(\mathtt{C} \mid \mathbf{\Pi}_1^n\, \mathtt{down}.\,\mathbf{0}, \mathtt{Counter}_n) \;\mid\; n \geq 0\}\,,$$

is a bisimulation up-to $\sim$

Take $(\mathtt{C} \mid \mathbf{\Pi}_1^n\, \mathtt{down}.\,\mathbf{0}, \mathtt{Counter}_n)$ in $\mathcal{R}$.

Suppose $\mathtt{C} \mid \mathbf{\Pi}_1^n\, \mathtt{down}.\,\mathbf{0} \xrightarrow{\mu} P$.

By inspecting the inference rules for parallel composition: μ can only be either $\mathtt{up}$ or $\mathtt{down}$.

$\mu = \mathtt{up}$. the transition from $\mathtt{C} \mid \mathbf{\Pi}_1^n\, \mathtt{down}.\,\mathbf{0}$ originates from $\mathtt{C}$, which performs the transition $\mathtt{C} \xrightarrow{\mathtt{up}} \mathtt{C} \mid \mathtt{down}.\,\mathbf{0}$, and $P = \mathtt{C} \mid \mathbf{\Pi}_1^{n+1}\, \mathtt{down}.\,\mathbf{0}$. Process $\mathtt{Counter}_n$ can answer $\mathtt{Counter}_n \xrightarrow{\mathtt{up}} \mathtt{Counter}_{n+1}$. For $P = P'$ and $Q = \mathtt{Counter}_{n+1}$, this closes the diagram.

The pair being inspected: $(\mathtt{C} \mid \mathbf{\Pi}_1^n\ \mathtt{down}.\mathbf{0}, \mathtt{Counter}_n)$

Action: $\mathtt{C} \mid \mathbf{\Pi}_1^n\ \mathtt{down}.\mathbf{0} \xrightarrow{\mu} \boldsymbol{P}$

$\boldsymbol{\mu = \mathtt{down}}$. It must be $\boldsymbol{n > 0}$. The action must originate from one of the $\mathtt{down}.\mathbf{0}$ components of $\mathbf{\Pi}_1^n\ \mathtt{down}.\mathbf{0}$, which has made the transition $\mathtt{down}.\mathbf{0} \xrightarrow{\mathtt{down}} \mathbf{0}$.

Therefore $\boldsymbol{P} = \mathtt{C} \mid \mathbf{\Pi}_1^n\ \boldsymbol{P_i}$, where exactly one $\boldsymbol{P_i}$ is $\mathbf{0}$ and all the others are $\mathtt{down}.\mathbf{0}$.

we have: $\boldsymbol{P} \sim \mathtt{C} \mid \mathbf{\Pi}_1^{n-1}\ \mathtt{down}.\mathbf{0}$.

Process $\mathtt{Counter}_n$ can answer with the transition $\mathtt{Counter}_n \xrightarrow{\mathtt{down}} \mathtt{Counter}_{n-1}$.

This closes the diagram, for $\boldsymbol{P'} \overset{\text{def}}{=} \mathtt{C} \mid \mathbf{\Pi}_1^{n-1}\ \mathtt{down}.\mathbf{0}$ and $\boldsymbol{Q} \overset{\text{def}}{=} \mathtt{Counter}_{n-1}$, as $\boldsymbol{P'\ \mathcal{R}\ Q}$.

The case when $\mathtt{Counter}_n$ moves first and $\mathtt{C} \mid \mathbf{\Pi}_1^n\ \mathtt{down}.\mathbf{0}$ has to answer is similar.

Weak bisimulation

Consider the processes

$$\tau.\overline{a}.\mathbf{0} \qquad \text{and} \qquad \overline{a}.\mathbf{0}$$

They are not strongly bisimilar.

But we do want to regard them as behaviourally equivalent! τ-transitions represent internal activities of processes, which are not visible.

(Analogy in functional languages: $(\lambda x.\, x)\mathbf{3}$ and $\mathbf{3}$ are semantically the same.)

Internal work (τ-transitions) should be ignored in the bisimulation game. Define:

(i) $\Longrightarrow$ as the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$.

(ii) $\stackrel{\mu}{\Longrightarrow}$ as $\Longrightarrow\stackrel{\mu}{\longrightarrow}\Longrightarrow$ (relational composition).

(iii) $\stackrel{\widehat{\mu}}{\Longrightarrow}$ is $\Longrightarrow$ if $\mu = \tau$; it is $\stackrel{\mu}{\Longrightarrow}$ otherwise.

Definition A process relation $\mathcal{R}$ is a **weak bisimulation** if $P\ \mathcal{R}\ Q$ implies:

1. if $P \stackrel{\mu}{\Longrightarrow} P'$, then there is Q' s.t. $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P'\ \mathcal{R}\ Q'$;
2. the converse of (1) on the actions from Q.

Definition P and Q are **weakly bisimilar**, written $P \approx Q$, if $P\ \mathcal{R}\ Q$ for some weak bisimulation $\mathcal{R}$.

Why did we study strong bisimulation?

- $\sim$ is simpler to work with, and $\sim \subseteq \approx$; (cf: exp. law)
- the theory of $\approx$ is in many aspects similar to that of $\sim$;
- the differences between $\sim$ and $\approx$ correspond to subtle points in the theory of $\approx$

Are the processes $\tau.\,0 + \tau.\,\overline{a}.\,0$ and $\overline{a}.\,0$ weakly bisimilar ?

Examples of non-equivalence:

$$a + b \not\approx a + \tau.b \not\approx \tau.a + \tau.b \not\approx a + b$$

Examples of equivalence:

$$\tau.a \approx a \approx a + \tau.a$$

$$a.(b + \tau.c) \approx a.(b + \tau.c) + a.c$$

These are instances of useful algebraic laws, called the **τ laws**:

Lemma

1. $P \approx \tau.P$
2. $\tau.N + N \approx N$
3. $M + \alpha.(N + \tau.P) \approx M + \alpha.(N + \tau.P) + \alpha.P$

In the clauses of weak bisimulation, the use of $\stackrel{\mu}{\Longrightarrow}$ on the challenger side can be heavy.

For instance, take $K \triangleq \tau.(a \mid K)$; for all n, we have $K \Longrightarrow (a \mid)^n \mid K$, and all these transitions have to be taken into account in the bisimulation game.

The following definition is much simpler to use:

Definition A process relation $\mathcal{R}$ is a **weak bisimulation** if $P \mathcal{R} Q$ implies:

1. if $P \stackrel{\mu}{\longrightarrow} P'$, then there is Q' s.t. $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$;
2. the converse of (1) on the actions from Q

Proposition The two definitions of weak bisimulation coincide.

Proof: a useful exercise.

Weak bisimulations “up-to”

Definition [weak bisimulation up-to $\sim$] A process relation $\mathcal{R}$ is a **weak bisimulation up-to $\sim$** if $P \; \mathcal{R} \; Q$ implies:

1. if $P \stackrel{\mu}{\longrightarrow} P'$, then there is Q' s.t. $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P' \sim \mathcal{R} \sim Q'$;
2. the converse of (1) on the actions from Q.

Exercize If $\mathcal{R}$ is a weak bisimulation up-to $\sim$ then $\mathcal{R} \subseteq \approx$.

Definition [weak bisimulation up-to $\approx$] A process relation $\mathcal{R}$ is a **weak bisimulation up-to $\approx$** if $P \; \mathcal{R} \; Q$ implies:

1. if $P \stackrel{\mu}{\Longrightarrow} P'$, then there is Q' s.t. $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P' \approx \mathcal{R} \approx Q'$;
2. the converse of (1) on the actions from Q.

Exercize If $\mathcal{R}$ is a weak bisimulation up-to $\approx$ then $\mathcal{R} \subseteq \approx$.

Enhancements of the bisimulation proof method

- The forms of “up-to” techniques we have seen are examples of **enhancements** of the bisimulation proof method
- Such enhancements are **extremely useful**
  - They are **essential** in π-calculus-like languages, higher-order languages
- **Various forms** of enhancement (“up-to techniques”) exist (up-to context, up-to substitution, etc.)
- They are **subtle**, and not well-understood yet

Example: up-to bisimilarity that fails

In Definition of “weak bisimulation up-to $\sim$” we cannot replace $\sim$ with $\approx$:

$$\begin{array}{ccc} \tau.a.0 & \mathcal{R} & 0 \\ \downarrow & & \Downarrow \\ a.0 & & 0 \\ \\ \approx & & \approx \\ \\ \tau.a.0 & \mathcal{R} & 0 \end{array}$$

Other equivalences

Concurrency theory: models of processes

- LTS
- Petri Nets
- Mazurkiewikz traces
- Event structures
- I/O automata

Process calculi

- CCS [$\rightarrow$ π-calculus $\rightarrow$ Join]
- CSP
- ACP
- Additional features: real-time, probability,...

Behavioural equivalences (and preorders)

- traces
- bisimilarity (in various forms)
- failures and testing
- non-interleaving equivalences (in which parallelism cannot be reduced to non-determinism, cf. the expansion law)
  [causality, location-based]

Depending on the desired level of abstraction or on the tools available, an equivalence may be better than an other.

van Glabbeek, in ’93, listed more than 50 forms of behavioural equivalence, today the listing would be even longer

Rob J. van Glabbeek: The Linear Time - Branching Time Spectrum II, LNCS 715, 1993

Failure equivalence

In CSP equivalence, it is intended that the observations are those obtained from all possible finite experiments with the process

A failure is a pair (μ^+, A), where μ^+ is a trace and A a set of actions. The failure (μ^+, A) belongs to process P if

- $P \xrightarrow{\mu^+} P'$, for some P'
- not $P' \xrightarrow{\tau}$
- not $P' \xrightarrow{a}$, for all $a \in A$

Example: $P \stackrel{\text{def}}{=} a.\,(b.\,c.\,0 + b.\,d.\,0)$ has the following failures:

- (ϵ, A) for all A with $a \notin A$.
- (a, A) for all A with $b \notin A$.
- (ab, A) for all A with $\{c, d\} \not\subseteq A$.
- (abc, A) and (abd, A), for all A

Two processes are **failure-equivalent** if they possess the same failures

Advantages of failure equivalence:

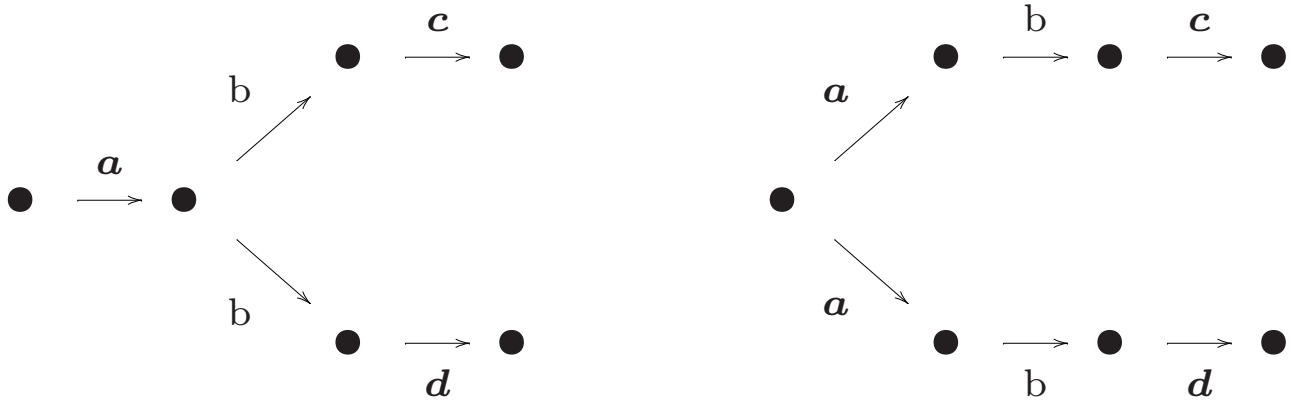
- the coarsest equivalence sensitive to deadlock
- characterisation as testing equivalence

Advantages of bisimilarity:

- the coinductive technique
- the finest reasonable behavioural equivalence for processes
- robust mathematical characterisations

Failure is not preserved, for instance, by certain forms of priority

These processes are failure equivalent but not bisimilar



A law valid for failure but not for bisimilarity:

$$a.(b.P + b.Q) = a.b.P + a.b.Q$$

Ready equivalence

A similar, but slightly finer, equivalence: ready equivalence.

A pair (μ^+, A) is a ready pair of P if $P \xrightarrow{\mu^+} P'$ and A is the set of action that P' can immediately perform.

These processes are failure, but not ready, equivalent:

$$a.b + a.c \qquad a.b + a.c + a.(b + c)$$

Testing

The testing theme

Processes should be equivalent unless

there is some test that can tell them apart

- We first show how to capture bisimilarity this way
- Then we will notice that there are other reasonable ways of defining the language of tests, and these may lead to different semantic notions.
- In this section: processes are (image-finte) LTSs (ie, finitely-branching labelled trees), with labels from a given alphabet of actions $\mathtt{Act}$

Bisimulation in a testing scenario

Language for testing:

$$T ::= \mathrm{SUCC} \mid \mathrm{FAIL} \mid a.T \mid \widetilde{a}.T \mid T_1 \wedge T_2 \mid T_1 \vee T_2 \mid \forall T \mid \exists T$$

$(a \in \mathtt{Act})$

The outcomes of an **experiment**, testing a process P with a test T:

$$\mathcal{O}(T, P) \subseteq \{\top, \bot\}$$

$\top$:success

$\bot$: lack of success (failure, or success is never reached)

Notation:

$P\ \mathtt{ref}(a) \stackrel{\text{def}}{=} P$ cannot perform a (ie, there is no P' st $P \xrightarrow{a} P'$)

Outcomes

$$\mathcal{O}(\mathrm{SUCC}, P) = \top$$

$$\mathcal{O}(\mathrm{FAIL}, P) = \bot$$

$$\mathcal{O}(a.T, P) = \begin{cases} \{\bot\} & \text{if } P \ \mathtt{ref}(a) \\ \bigcup\{\mathcal{O}(T, P') \mid P \xrightarrow{a} P'\} & \text{otherwise} \end{cases}$$

$$\mathcal{O}(\widetilde{a}.T, P) = \begin{cases} \{\top\} & \text{if } P \ \mathtt{ref}(a) \\ \bigcup\{\mathcal{O}(T, P') \mid P \xrightarrow{a} P'\} & \text{otherwise} \end{cases}$$

$$\mathcal{O}(T_1 \wedge T_2, P) = \mathcal{O}(T_1, P) \wedge^\star \mathcal{O}(T_1, P)$$

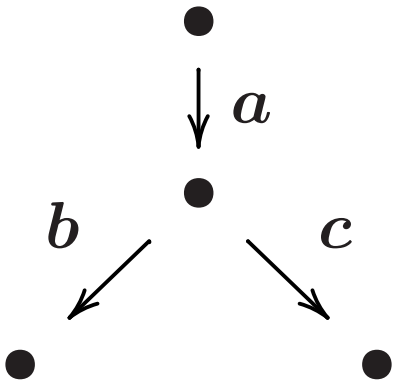
$$\mathcal{O}(T_1 \vee T_2, P) = \mathcal{O}(T_1, P) \vee^\star \mathcal{O}(T_1, P)$$

$$\mathcal{O}(\forall T, P) = \begin{cases} \{\top\} & \text{if } \bot \notin \mathcal{O}(T, P) \\ \{\bot\} & \text{otherwise} \end{cases}$$

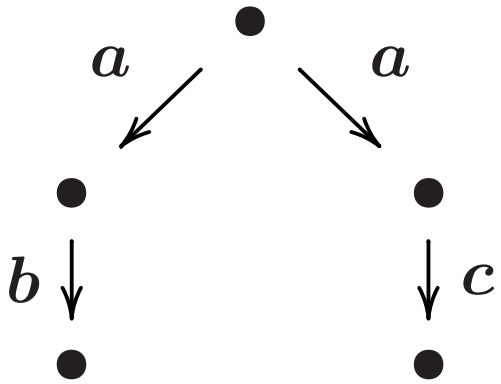
$$\mathcal{O}(\exists T, P) = \begin{cases} \{\top\} & \text{if } \top \in \mathcal{O}(T, P) \\ \{\bot\} & \text{otherwise} \end{cases}$$

where $\wedge^\star$ and $\vee^\star$ are the pointwise extensions of $\wedge$ and $\vee$ to powersets

Examples (a)



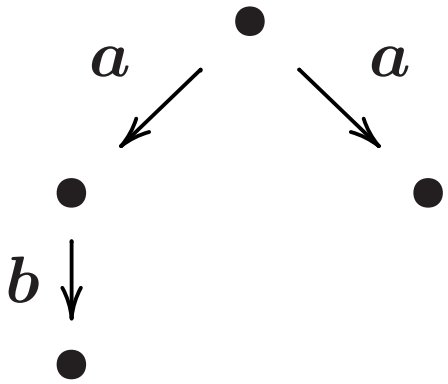
P_1



P_2

For $T_1 = a.\,(b.\,\mathrm{SUCC} \wedge c.\,\mathrm{SUCC})$, we have $\mathcal{O}(T_1, P_1) = \{\top\}$ and $\mathcal{O}(T_1, P_2) = \{\bot\}$

Examples (b)



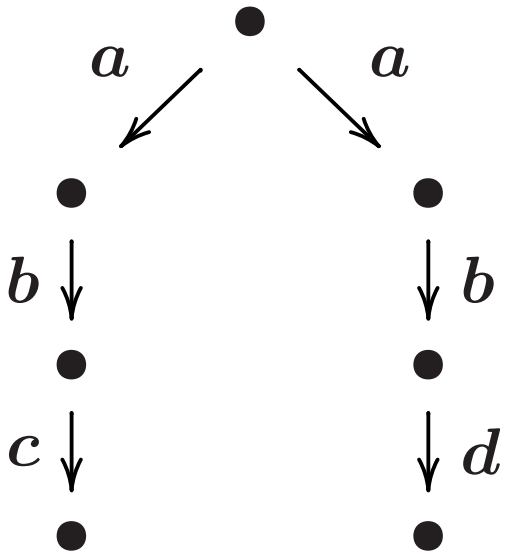
P_3

P_4

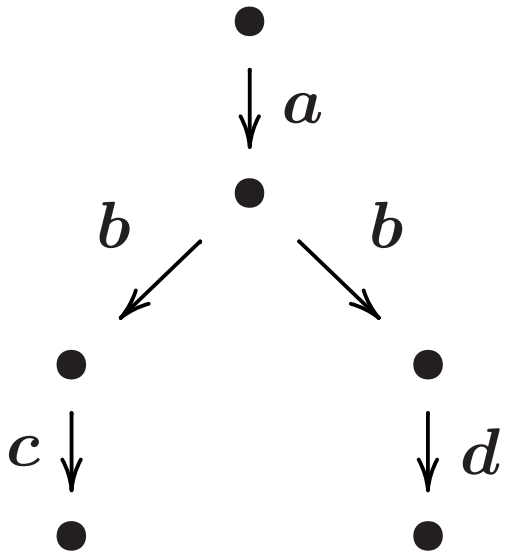
For $T_3 = a.\, b.\, \mathrm{SUCC}$, we have $\mathcal{O}(T_3, P_3) = \{\bot, \top\}$ and $\mathcal{O}(T_3, P_4) = \{\top\}$

For $T_4 = a.\, \widetilde{b}.\, \mathrm{FAIL}$, we have $\mathcal{O}(T_4, P_3) = \{\bot, \top\}$ and $\mathcal{O}(T_4, P_4) = \{\bot\}$

Examples (c)



P_5

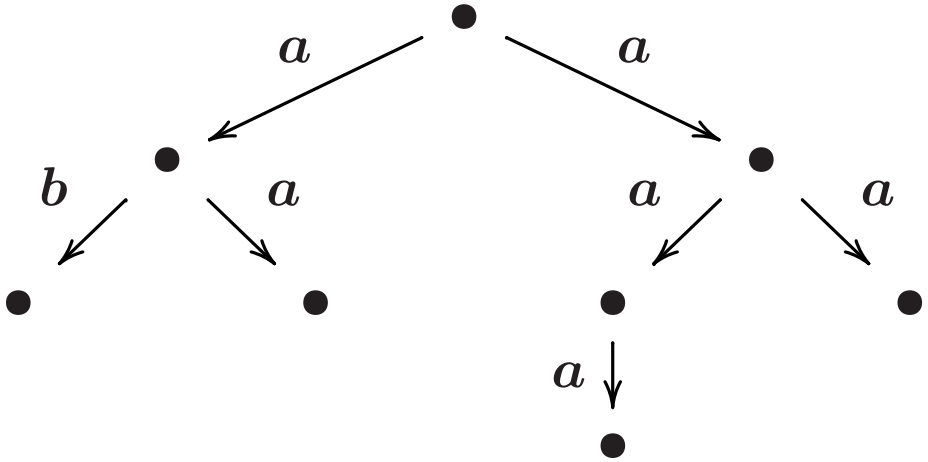


P_6

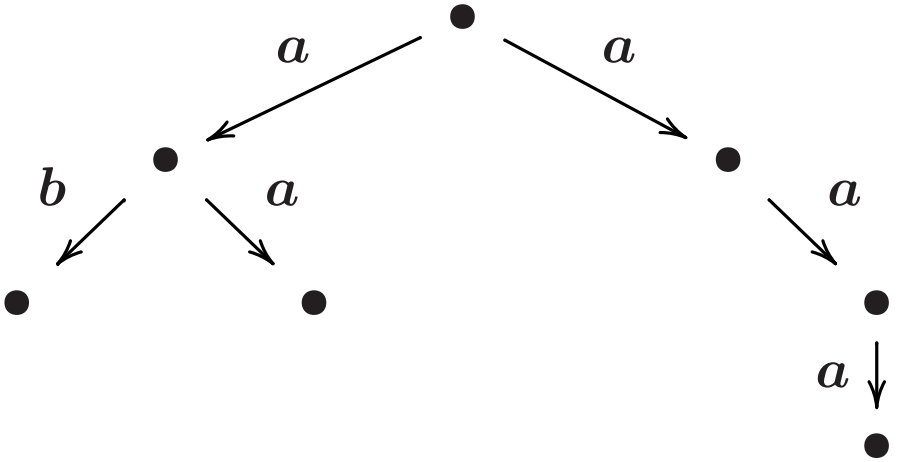
For $T = \exists a.\, \forall b.\, c.\, \mathrm{SUCC}$, we have $\mathcal{O}(T, P_5) = \{\top\}$ and $\mathcal{O}(T, P_6) = \{\bot\}$

Exercise: define other tests that distinguish between P_5 and P_6.

Examples (d)



P_7



P_8

Exercise: Define tests that distinguish between P_7 and P_8.

Note: Every test has an inverse:

$$
\begin{aligned}
\overline{\mathrm{SUCC}} &= \mathrm{FAIL} \\
\overline{\mathrm{FAIL}} &= \mathrm{SUCC} \\
\overline{\boldsymbol{a.T}} &= \widetilde{\boldsymbol{a}}.\overline{\boldsymbol{T}} \\
\overline{\widetilde{\boldsymbol{a}}.\boldsymbol{T}} &= \boldsymbol{a}.\overline{\boldsymbol{T}} \\
\overline{\boldsymbol{T_1} \wedge \boldsymbol{T_2}} &= \overline{\boldsymbol{T_1}} \vee \overline{\boldsymbol{T_2}} \\
\overline{\boldsymbol{T_1} \vee \boldsymbol{T_2}} &= \overline{\boldsymbol{T_1}} \wedge \overline{\boldsymbol{T_2}} \\
\overline{\forall \boldsymbol{T}} &= \exists \overline{\boldsymbol{T}} \\
\overline{\exists \boldsymbol{T}} &= \forall \overline{\boldsymbol{T}}
\end{aligned}
$$

We have:

1. $\bot \in \mathcal{O}(\boldsymbol{T}, \boldsymbol{P})$ iff $\top \in \mathcal{O}(\overline{\boldsymbol{T}}, \boldsymbol{P})$
2. $\top \in \mathcal{O}(\boldsymbol{T}, \boldsymbol{P})$ iff $\bot \in \mathcal{O}(\overline{\boldsymbol{T}}, \boldsymbol{P})$

The equivalence induced by these tests:

$P \sim_{\mathrm{T}} Q \stackrel{\text{def}}{=}$ for all T, $\mathcal{O}(T, P) = \mathcal{O}(T, Q)$.

Theorem $\sim \; = \; \sim_{\mathrm{T}}$

- The proof is along the lines of the proof of characterisation of bisimulation in terms of modal logics (Hennessy-Milner's logics and theorem)
- A similar theorem holds for weak bisimilarity (with internal actions, the definition of the tests may need to be refined)

Testing equivalence

- The previous testing scenario requires considerable control over the processes (eg: the ability to copy their state at any moment)
  One may argue that this is too strong
- An alternative: the tester is a process of the same language as the tested process (in our case: an LTS)
- Performing a test : the two processes attempt to communicate with each other.
- Thus most of the constructs in the previous testing language are no longer appropriate (for instance, because they imply the ability of copying a process)
- To signal success, the tester process uses a special action $w \notin \mathtt{Act}$

Outcomes of running a test

Experiments:

$$E ::= \langle T, P \rangle \mid \top$$

A *run* for a pair $\langle T, P \rangle$: a (finite or infinite) sequence of esperiments E_i such that

1. $E_0 = \langle T, P \rangle$
2. a transition $E_i \xrightarrow{a} E_{i+1}$ is defined by the following rules:

$$\frac{T \xrightarrow{a} T' \qquad P \xrightarrow{a} P'}{\langle T, P \rangle \longrightarrow \langle T', P' \rangle}$$

$$\frac{T \xrightarrow{w} T'}{\langle T, P \rangle \longrightarrow \top}$$

3. the last element of the sequence, say E_k, is such that there is no E' such that $E_k \longrightarrow E'$.

We now set:

$\top \in \mathcal{O}(T, P)$ if $\langle T, P \rangle$ has a run in which $\top$ appears (ie, $\langle T, P \rangle \Longrightarrow \top$)

$\bot \in \mathcal{O}(T, P)$ if there is a run for $\langle T, P \rangle$ in which $\top$ never appears

Testing equivalence ($\simeq$): the equivalence on processes so obtained

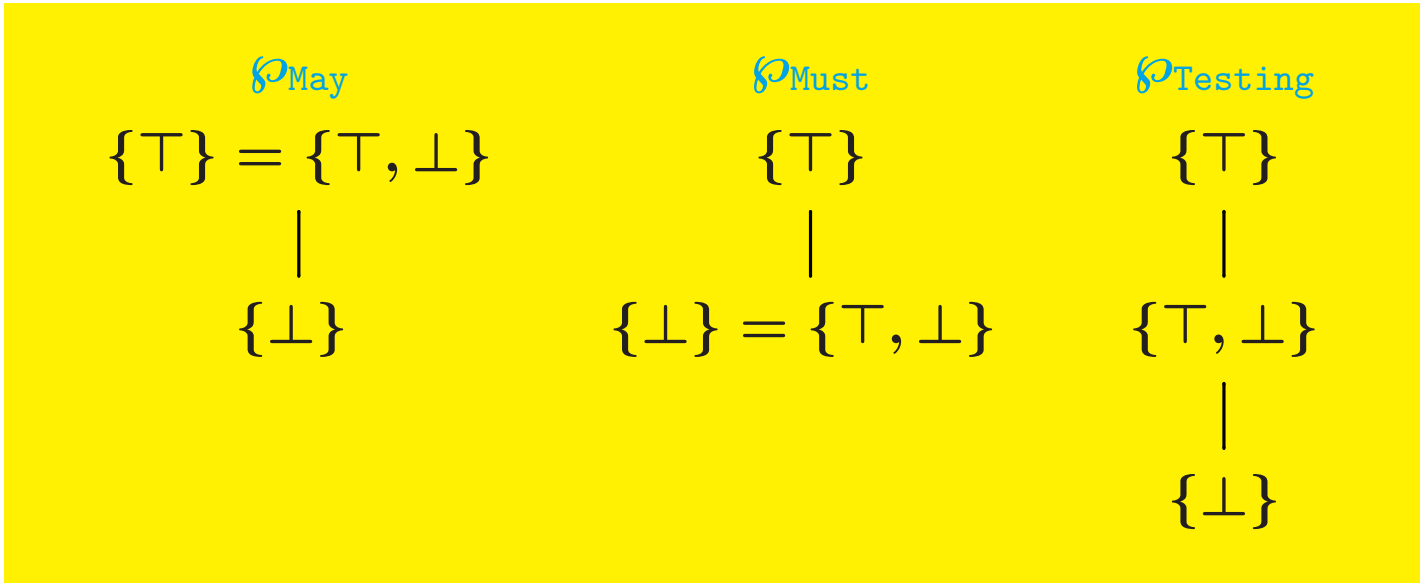
Note: If processes could perform internal actions, then other rules would be needed:

$$\frac{T \xrightarrow{\tau} T'}{\langle T, P \rangle \longrightarrow \langle T', P \rangle}$$

$$\frac{P \xrightarrow{\tau} P'}{\langle T, P \rangle \longrightarrow \langle T, P' \rangle}$$

$\mathcal{O}(T, P)$ is a non-empty subset of the 2-point lattice

However, there are 3 ways of lifting such lattice to its non-empty subsets:



$\wp_{\mathtt{May}}$: the possibility of success is essential

$\wp_{\mathtt{Must}}$: failure is disastrous

The resulting equivalences are $\simeq_{\mathtt{May}}$ (**may testing**) and $\simeq_{\mathtt{Must}}$ (**must testing**)

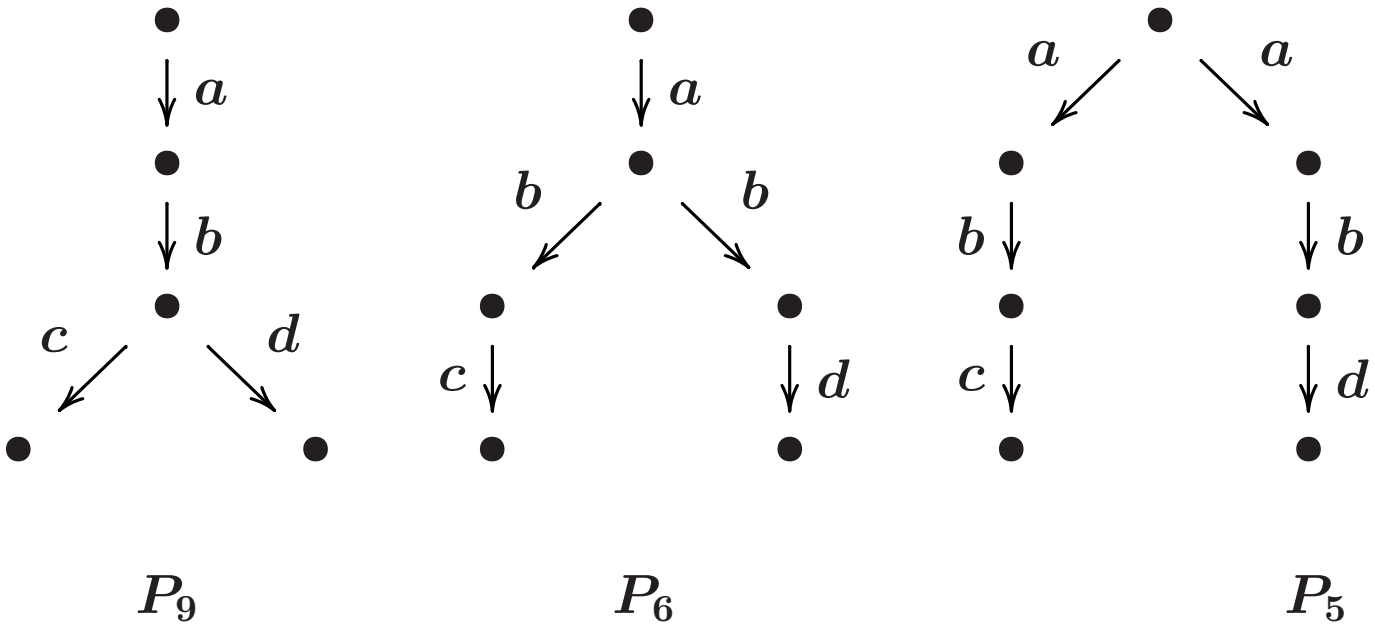
Note: $\simeq_{\mathtt{Testing}}$ is $\simeq$

Results for the test-based relations

Theorem

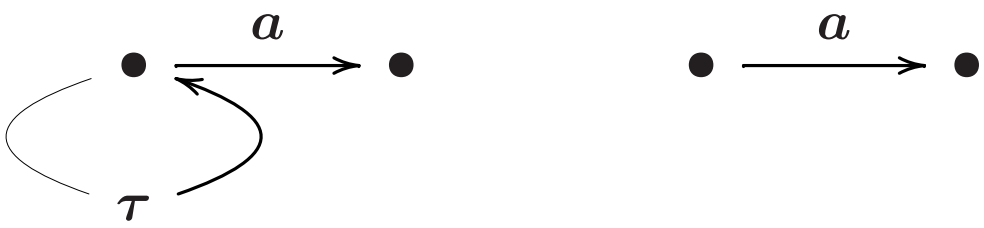
1. $\simeq = (\simeq_{\mathtt{May}} \cap \simeq_{\mathtt{Must}})$
2. $\simeq_{\mathtt{May}}$ coincides with trace equivalence
3. $\simeq$ coincides with failure equivalence

Example



P_9 P_6 P_5

P_9	$\simeq_{\mathtt{May}}$	P_5	$\simeq_{\mathtt{May}}$	P_6
P_9	$\not\simeq_{\mathtt{Must}}$	P_5	$\simeq_{\mathtt{Must}}$	P_6
P_9	$\not\simeq$	P_5	$\simeq$	P_6
P_9	$\not\asymp$	P_5	$\not\asymp$	P_6



Q_1 Q_2

In CCS: $Q_1 = \tau^\omega \mid a$, and $Q_2 = a.0$

Q_1 and Q_2 are **weakly bisimilar**, but **not testing equivalent**

Justification for testing: **bisimulation is insensitive to divergence**

Justification for bisimulation: **testing is not "fair"**

(notions of fair testing have been proposed, and then bisimulation is indeed strictly included in testing)

All equivalences discussed in these lectures reduce parallelism to interleaving, in that

$$a.0 \mid b.0 \text{ is the same as } a.b.0 + b.a.0$$

Not discussed in these lectures: equivalences that refuse the above equality (called true-concurrency, or non-interleaving)

Bisimulation and Coinduction:

examples of research problems

- **Bisimulation in higher-order languages**
- **Enhancements of the bisimulation proof method**
- **Combination of inductive and coinductive proofs (eg, proof that bisimilarity is a congruence)**
- **Languages with probabilistic constructs**
- **Unifying notions**