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/`

Scuola Logica, Lago Garda, Agosto 2013

Capsule bio

- 1983-1986: undergraduate, Pisa University, Italy
- 1987: research assistant, Pisa University
  (A. Maggiolo-Schettini)
- 1988: military service
- 1989-91: PhD, Ednburgh University, UK
  (Robin Milner)
- 1991-1994: Research Fellow, Ednburgh University
  (Robin Milner)
- 1995-2002 INRIA, France
- 2003 - today, Bologna University

FOCUS

FOundations of Component-based Ubiquitous Systems

A joint effort between the Institut national de recherche en informatique et automatique (INRIA) (France) and the University of Bologna (Italy)

Semantic foundations of distributed systems (models, formalization and verification of properties, linguistic constructs, prototypes)

Algebra, logics, type theory

About 15/20 people, 8 permanents **INRIA**: 8 centers in France, ~$\mathbf{3,500}$ researchers

Why induction and coinduction in this school

- **logical concepts**
- **interdisciplinary concepts**
- **fundamental notions for programming languages**
  - $\ast$ defining structures, objects
  - $\ast$ 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**

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**
  - $\ast$ the complement set of the terminating ones
    (an indirect definition)
  - $\ast$ 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**
  - $\ast$ the complement set of the open ones
  - $\ast$ 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**

  * $\ast$ to define equality on processes (fundamental !!)
  * $\ast$ to prove equalities
  * $\ast$ to justify algebraic laws
  * $\ast$ to minimise the state space
  * $\ast$ 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

Main 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**

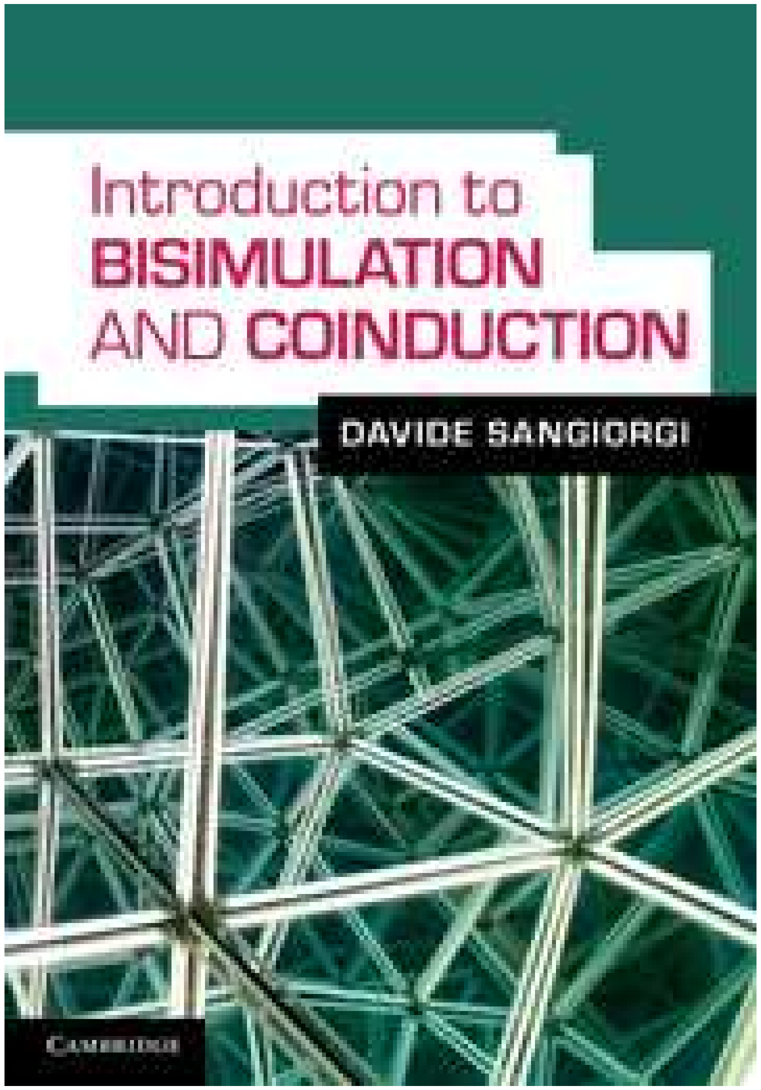
Other objectives

- know the interest of process calculi
- understand what is behavioural equality on processes
- see the possibilities of enhancements for the bisimulation proof method (if time permits)

References

This course is based on the book:

Davide Sangiorgi, *An introduction to bisimulation and coinduction*, Cambridge University Press, October 2012



Outline

- The success of bisimulation and coinduction
- Towards bisimulation, or: from functions to processes
- Bisimulation
- Induction and coinduction
- Process calculi and Concurrency theory
- Behavioral equivalences
- Bits of history
- (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:

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

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

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

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

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

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:
  $$(\ \mathtt{X} := \mathbf{1};\ \ \mathtt{X} :=\ \ \mathtt{X} + \mathbf{1}\ ) \mid\ \ \mathtt{X} := \mathbf{2}$$
  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 19

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

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

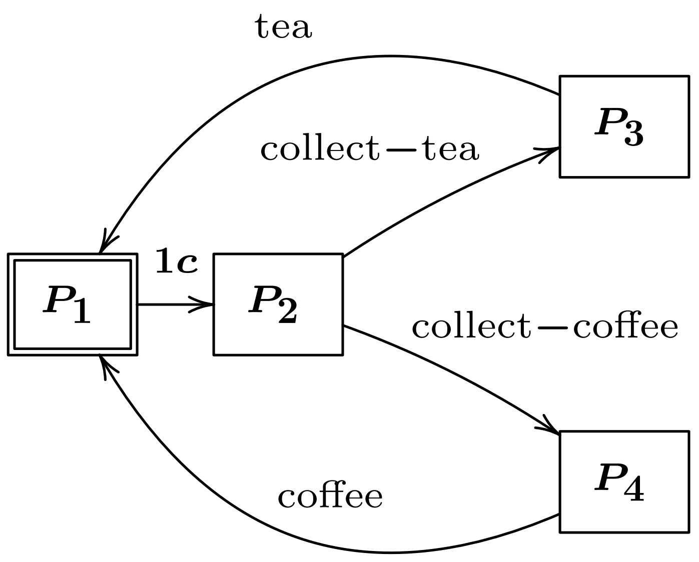
- $\mathcal{P}$ is the set of **states**, or **processes**;
- $\mathtt{Act}$ is the set of **actions**; (NB: can be infinite)
- $\mathcal{T} \subseteq (\mathcal{P}, \mathtt{Act}, \mathcal{P})$ is the **transition relation**.

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 $\boxed{\boxed{}}$ indicates the processes we are interested in (the “initial state”)

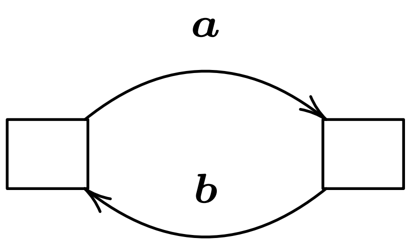
NB: the color of the machine is irrelevant

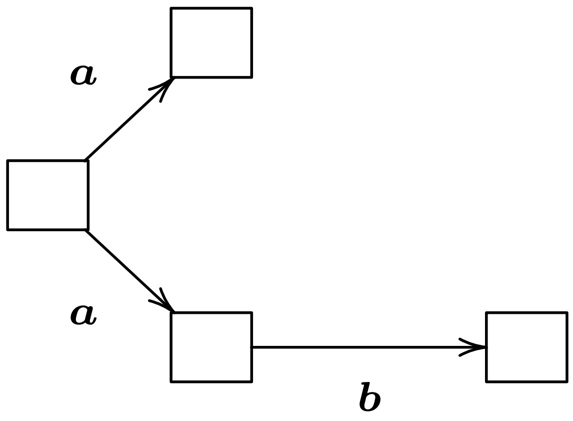
The behaviour of the first replacement machine

$$\begin{tikzcd}
 & \boxed{Q_2} \arrow[r, bend left=10, "\text{collect}-\text{tea}"] & \boxed{Q_3} \arrow[dll, "\text{tea}"'] \\
\boxed{\boxed{Q_1}} \arrow[ur, bend left, "\mathbf{1}c"] \arrow[dr, bend right, "\mathbf{1}c"'] & & \\
 & \boxed{Q_4} \arrow[r, bend right=10, "\text{collect}-\text{coffee}"'] & \boxed{Q_5} \arrow[ull, "\text{coffee}"]
\end{tikzcd}$$

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

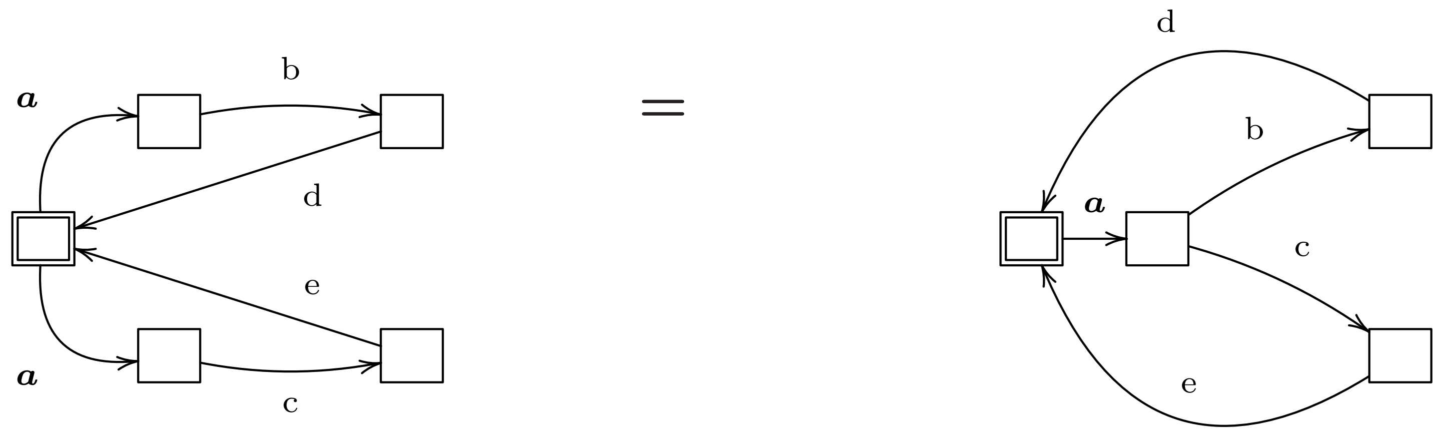
$$\begin{tikzcd} \boxed{P_1} \arrow[r, bend left, "a"] & \boxed{P_2} \arrow[l, bend left, "b"] & = & \boxed{Q_1} \arrow[r, "a"] & \boxed{Q_2} \arrow[r, bend left, "b"] & \boxed{Q_3} \arrow[l, bend left, "a"] \end{tikzcd}$$

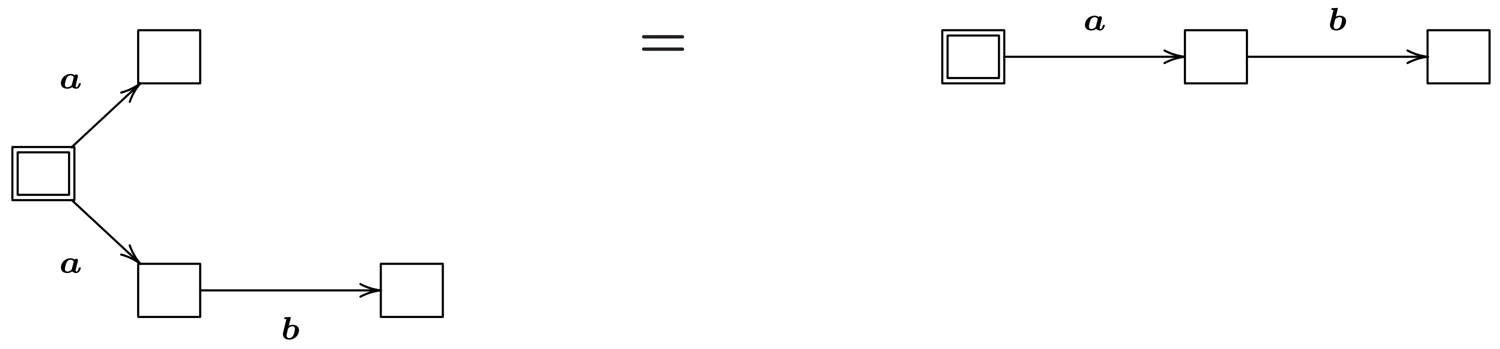
Can **graph theory** help? (equality is **graph isomorphism**)

... too strong (example above)

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

Examples of trace-equivalent processes:

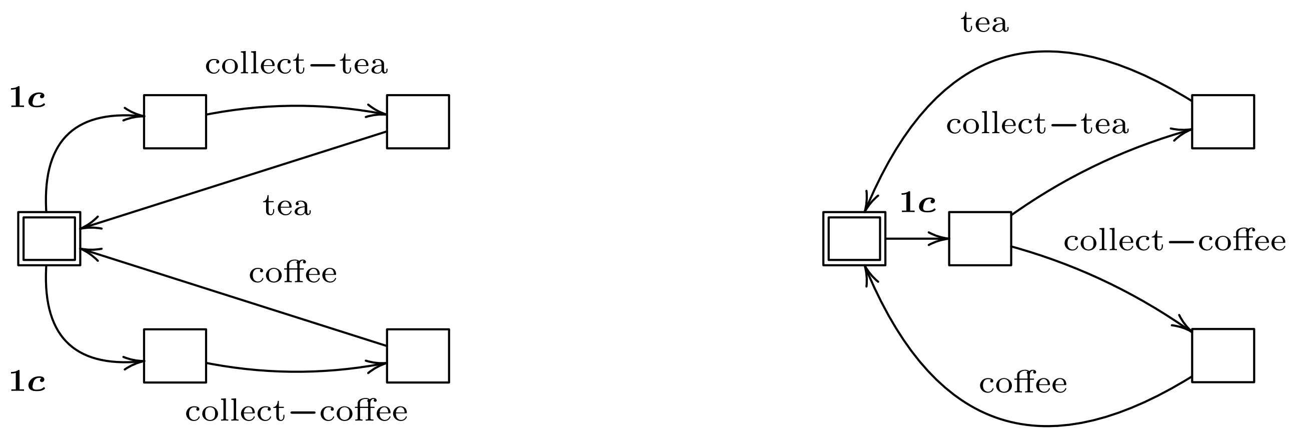




These equalities are OK on automata.

... 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:

$$\begin{tikzcd}
P \arrow[r, phantom, "\mathcal{R}"] \arrow[d, "\mu"'] & Q \arrow[d, "\mu"'] \\
P' \arrow[r, phantom, "\mathcal{R}"] & Q'
\end{tikzcd}$$

Examples

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

$$\begin{tikzcd} \boxed{\boxed{P_1}} \arrow[r, bend left, "a"] & \boxed{P_2} \arrow[l, bend left, "b"] \end{tikzcd} \qquad \begin{tikzcd} \boxed{\boxed{Q_1}} \arrow[r, "a"] & \boxed{Q_2} \arrow[r, bend left, "b"] & \boxed{Q_3} \arrow[l, bend left, "a"] \end{tikzcd}$$

Examples

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

$$\begin{tikzcd}
\boxed{\boxed{P_1}} \arrow[r, bend left, "a"] & \boxed{P_2} \arrow[l, bend left, "b"]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\boxed{\boxed{Q_1}} \arrow[r, "a"] & \boxed{Q_2} \arrow[r, bend left, "b"] & \boxed{Q_3} \arrow[l, bend left, "a"]
\end{tikzcd}$$

First attempt for a bisimulation:

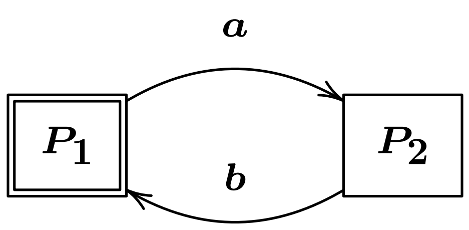
$$\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2)\}$$

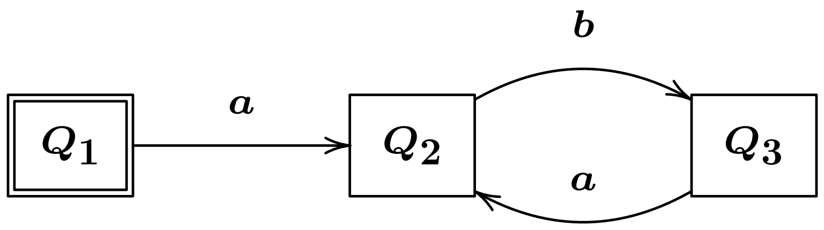
Bisimulation diagrams for (P_1, Q_1):

$$\begin{tikzcd}
P_1 \arrow[d, "a"'] & \mathcal{R} & Q_1 \\
P_2 & &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
P_1 & \mathcal{R} & Q_1 \arrow[d, "a"'] \\
& & Q_2
\end{tikzcd}$$

Examples

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





First attempt for a bisimulation:

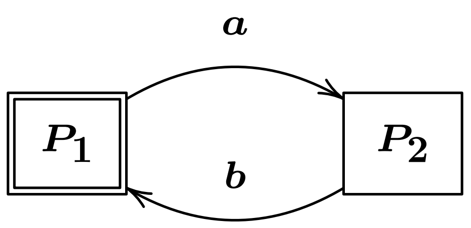
$$\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2)\}$$

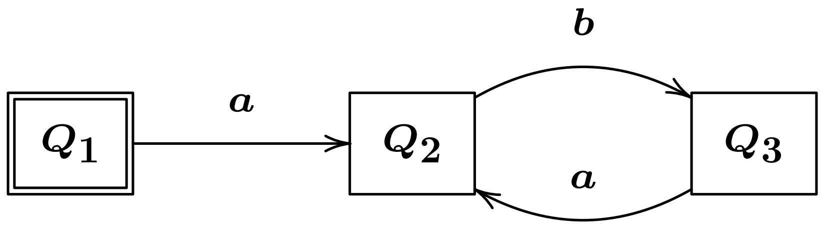
Bisimulation diagrams for (P_1, Q_1):

$$\begin{tikzcd} P_1 \arrow[d, "a"'] & \mathcal{R} & Q_1 \arrow[d, "a"'] \\ P_2 & \mathcal{R} & Q_2 \end{tikzcd} \qquad \begin{tikzcd} P_1 \arrow[d, "a"'] & \mathcal{R} & Q_1 \arrow[d, "a"'] \\ P_2 & \mathcal{R} & Q_2 \end{tikzcd}$$

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):

$$\begin{tikzcd} P_2 \arrow[d, "b"'] & \mathcal{R} & Q_2 \\ P_1 & & \end{tikzcd} \qquad \begin{tikzcd} P_2 & \mathcal{R} & Q_2 \arrow[d, "b"'] \\ & & Q_3 \end{tikzcd}$$

Examples

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

$$\begin{tikzcd}
\boxed{\boxed{P_1}} \arrow[r, bend left, "a"] & \boxed{P_2} \arrow[l, bend left, "b"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\boxed{\boxed{Q_1}} \arrow[r, "a"] & \boxed{Q_2} \arrow[r, bend left, "b"] & \boxed{Q_3} \arrow[l, bend left, "a"]
\end{tikzcd}$$

First attempt for a bisimulation:

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

Bisimulation diagrams for (P_2, Q_2):

$$\begin{tikzcd}
P_2 \arrow[d, "b"'] & \mathcal{R} & Q_2 \arrow[d, "b"] \\
P_1 & \cancel{\mathcal{R}} & Q_3
\end{tikzcd}
\qquad
\begin{tikzcd}
P_2 \arrow[d, "b"'] & \mathcal{R} & Q_2 \arrow[d, "b"] \\
P_1 & \cancel{\mathcal{R}} & Q_3
\end{tikzcd}$$

Examples

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

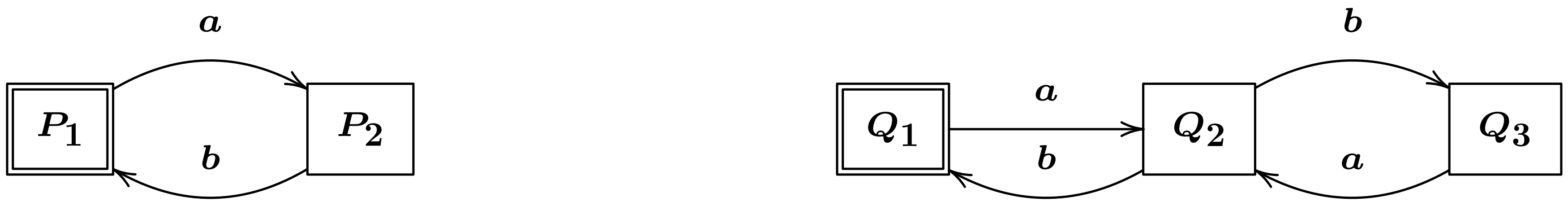
$$\begin{tikzcd} P_1 \arrow[r, bend left, "a"] & P_2 \arrow[l, bend left, "b"] \end{tikzcd} \qquad \begin{tikzcd} Q_1 \arrow[r, "a"] & Q_2 \arrow[r, bend left, "b"] & Q_3 \arrow[l, bend left, "a"] \end{tikzcd}$$

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:

$$\begin{tikzcd}
P_2 \arrow[d, "b"'] \arrow[r, phantom, "\mathcal{R}"] & Q_2 \arrow[d, "b"'] \\
P_1 \arrow[r, phantom, "\mathcal{R}"] & Q_1
\end{tikzcd}
\qquad
\begin{tikzcd}
P_2 \arrow[d, "b"'] \arrow[r, phantom, "\mathcal{R}"] & Q_2 \arrow[d, "b"'] \\
P_1 \arrow[r, phantom, "\mathcal{R}"] & Q_1
\end{tikzcd}$$

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

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

$$\begin{tikzcd}
\boxed{\boxed{Q_1}} \arrow[r, "a"] & \boxed{Q_2} \arrow[r, bend left, "b"] & \boxed{Q_3} \arrow[l, bend left, "a"]
\end{tikzcd}$$

$$\begin{tikzcd}
\boxed{\boxed{R_1}} \arrow[r, "a"] & \boxed{R_2} \arrow[r, "b"] & \boxed{R_3} \arrow[r, bend left, "a"] & \boxed{R_4} \arrow[l, bend left, "b"] \arrow[lll, bend left, "b"]
\end{tikzcd}$$

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):

$$\begin{tikzcd}
Q_2 \arrow[d, "b"'] & \mathcal{R} & R_4 \arrow[d, "b"'] \\
Q_3 & \mathcal{R} & R_3
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
Q_2 \arrow[d, "b"'] & \mathcal{R} & R_4 \arrow[d, "b"'] \\
Q_3 & \mathcal{R} & R_3
\end{tikzcd}$$

One diagram check is missing. Which one?

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

$$\begin{tikzcd}
\boxed{\boxed{Q_1}} \arrow[r, "a"] & \boxed{Q_2} \arrow[r, bend left, "b"] & \boxed{Q_3} \arrow[l, bend left, "a"']
\end{tikzcd}$$

$$\begin{tikzcd}
\boxed{\boxed{R_1}} \arrow[r, "a"] & \boxed{R_2} \arrow[r, "b"] & \boxed{R_3} \arrow[r, bend left, "a"] & \boxed{R_4} \arrow[l, bend left, "b"'] \arrow[lll, bend left, "b"]
\end{tikzcd}$$

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):

$$\begin{tikzcd}
Q_2 \arrow[d, "b"'] & \mathcal{R} & R_4 \arrow[d, "b"] \\
Q_3 & \mathcal{R} & R_3
\end{tikzcd}
\qquad
\begin{tikzcd}
Q_2 \arrow[d, "b"'] & \mathcal{R} & R_4 \arrow[d, "b"] \\
Q_3 & \mathcal{R} & R_3
\end{tikzcd}$$

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

We want to prove $M_1 \sim N_1$:

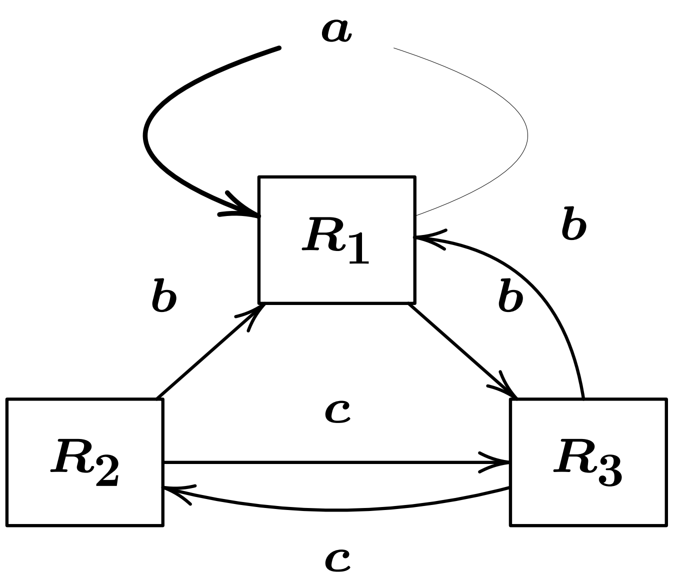
$$\begin{tikzcd}
\boxed{M_1} \arrow[d, bend right, "a"'] & & & \boxed{N_1} \arrow[dl, bend right, "a"'] \arrow[dr, bend left, "a"] & \\
\boxed{M_2} \arrow[u, bend right, "b"'] \arrow[d, bend right, "a"'] & & \boxed{N_2} \arrow[ur, bend right, "b"'] \arrow[d, bend right, "a"'] & & \boxed{N_3} \arrow[ul, bend left, "b"] \arrow[d, bend left, "a"] \\
\boxed{M_3} \arrow[u, bend right, "b"'] & & \boxed{N_4} \arrow[urr, "b"] & & \boxed{N_5} \arrow[ull, "b"']
\end{tikzcd}$$

A graphical representation of a bisimulation:

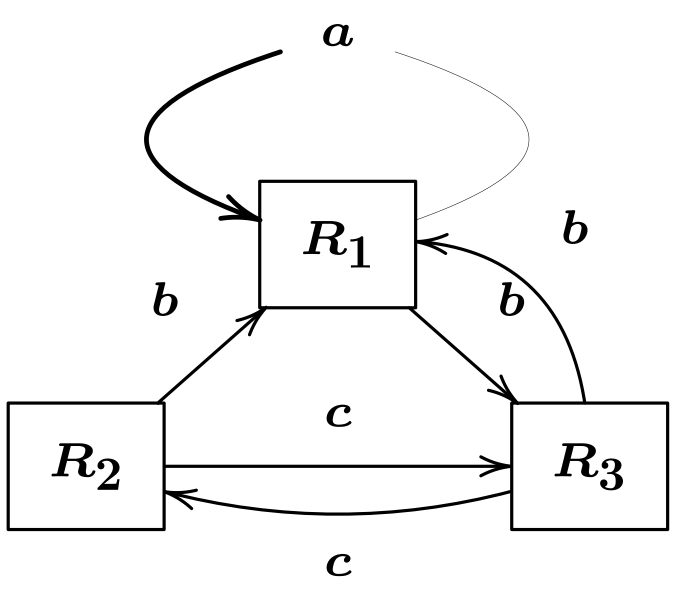
$$\begin{tikzcd}
\boxed{M_1} \arrow[d, bend right, "a"'] \arrow[rrr, dash, dotted, bend left] & & & \boxed{N_1} \arrow[dl, bend right, "a"'] \arrow[dr, bend left, "a"] & \\
\boxed{M_2} \arrow[u, bend right, "b"'] \arrow[d, bend right, "a"'] \arrow[rr, dash, dotted, bend left] \arrow[rrrr, dash, dotted, bend right] & & \boxed{N_2} \arrow[ur, bend right, "b"'] \arrow[d, bend right, "a"'] & & \boxed{N_3} \arrow[ul, bend left, "b"] \arrow[d, bend left, "a"] \\
\boxed{M_3} \arrow[u, bend right, "b"'] \arrow[rr, dash, dotted, bend left] \arrow[rrrr, dash, dotted, bend right] & & \boxed{N_4} \arrow[urr, "b"] & & \boxed{N_5} \arrow[ull, "b"']
\end{tikzcd}$$

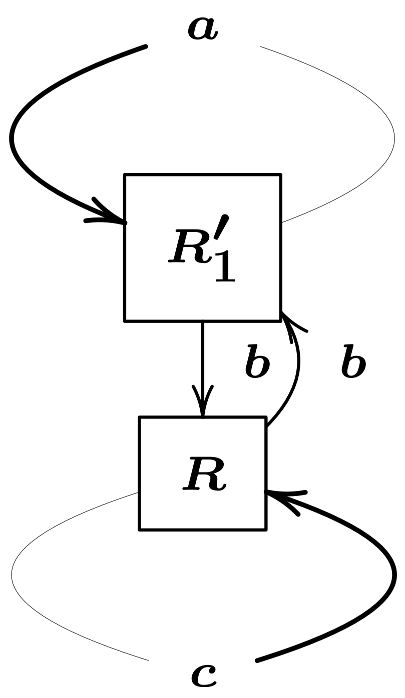
$$\{(M_1, N_1), (M_2, N_2), (M_2, N_3), (M_3, N_4), (M_3, N_5)\}$$

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

$$\begin{tikzcd}
& \bullet \arrow[d, "a"] & \\
& \bullet \arrow[dl, "b"'] \arrow[dr, "c"] & \\
\bullet & & \bullet
\end{tikzcd}$$

Q

$$\begin{tikzcd}
& \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
\bullet \arrow[d, "b"'] & & \bullet \arrow[d, "c"] \\
\bullet & & \bullet
\end{tikzcd}$$

R

$$\begin{tikzcd}
& & & \bullet \arrow[dlll, "a"'] \arrow[d, "a"] \arrow[drrr, "a"] & & & \\
\bullet \arrow[d, "b"'] & & & \bullet \arrow[dl, "b"'] \arrow[dr, "c"] & & & \bullet \arrow[d, "c"'] \\
\bullet & & \bullet & & \bullet & & \bullet
\end{tikzcd}$$

P

$$\begin{tikzcd}
& \bullet \arrow[d, "a"] & \\
& \bullet \arrow[d, "b"] & \\
& \bullet \arrow[dl, "c"'] \arrow[dr, "d"] & \\
\bullet & & \bullet
\end{tikzcd}$$

Q

$$\begin{tikzcd}
& \bullet \arrow[d, "a"] & \\
& \bullet \arrow[dl, "b"'] \arrow[dr, "b"] & \\
\bullet \arrow[d, "c"'] & & \bullet \arrow[d, "d"] \\
\bullet & & \bullet
\end{tikzcd}$$

R

$$\begin{tikzcd}
& \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
\bullet \arrow[d, "b"'] & & \bullet \arrow[d, "b"] \\
\bullet \arrow[d, "c"'] & & \bullet \arrow[d, "d"] \\
\bullet & & \bullet
\end{tikzcd}$$

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 $P \mathrel{\mathcal{R}} R$.

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

Set

$$\mathcal{R} = \{(A, C) \mid \text{there is } B \text{ with } A \mathrel{\mathcal{R}_1} B \text{ and } B \mathrel{\mathcal{R}_2} C\}$$

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

Take $(A, C) \in \mathcal{R}$, because $\exists\, B$ with $A \mathrel{\mathcal{R}_1} B$ and $B \mathrel{\mathcal{R}_2} C$, with $A \xrightarrow{a} A'$:

$$\begin{tikzcd}
A \arrow[r, phantom, "\mathcal{R}_1"] \arrow[d, "\mu"'] & B \arrow[r, phantom, "\mathcal{R}_2"] \arrow[d, "\mu"'] & C \arrow[d, "\mu"'] \\
A' \arrow[r, phantom, "\mathcal{R}_1"] & B' \arrow[r, phantom, "\mathcal{R}_2"] & C'
\end{tikzcd}$$

Proof of symmetry

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

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

Take

$$\mathcal{R}^{-1} = \{(B, A) \mid A \mathrel{\mathcal{R}} B\}$$

$\mathcal{R}^{-1}$ is a bisimulation.

We have $Q \mathrel{\mathcal{R}^{-1}} P$, thus $Q \sim P$.

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

2. $\forall \mu, Q'$ s.t. $Q \xrightarrow{\mu} Q'$, then $\exists P'$ 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. $\forall \mu, P'$ s.t. $P \xrightarrow{\mu} P'$, then $\exists Q'$ s.t. $Q \xrightarrow{\mu} Q'$ 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 28.)

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?

$$\begin{tikzcd}
& P & \\
& \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
\bullet \arrow[d, "b"'] & & \bullet \arrow[d, "c"] \\
\bullet & & \bullet
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
& Q & \\
& \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
\bullet \arrow[d, "b"'] & & \bullet \arrow[d, "c"] \\
\bullet & & \bullet
\end{tikzcd}$$

Other equivalences: examples

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

$$\begin{tikzcd}
 & \bullet \arrow[d, "a"] & & = & & \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
 & \bullet \arrow[dl, "b"'] \arrow[dr, "c"] & & & \bullet \arrow[d, "b"'] & & \bullet \arrow[d, "c"] \\
\bullet & & \bullet & & \bullet & & \bullet
\end{tikzcd}$$

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 $\boldsymbol{n}$, then it also holds at $\boldsymbol{n + 1}$

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 $n+1$)

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

A non-example of mathematical induction

Statement: **All men have eyes of the same color**

Basis: trivial (with only one man, there is only one color)

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

Order the $n+1$ men, in the positions $\{1, 2, \ldots, n+1\}$

Consider now the two subsets of men in positions $\{1, 2, \ldots, n\}$ and $\{2, \ldots, n+1\}$

Each subset has n elements.

By the inductive hypothesis, the men in each subset have the eyes of the same color

The two sets overlap, so there is only one color for the eyes of the $n+1$ men.

Rule induction: finite traces (may termination)

(assume only one label hence we drop it)

$$\begin{tikzcd}
 & \boxed{P_1} \arrow[r, bend left] \arrow[d] & \boxed{P_2} \arrow[l, bend left] \arrow[r] \arrow[d] & \boxed{P_7} \\
\boxed{P_5} & \boxed{P_3} \arrow[l] & \boxed{P_4} \arrow[r, bend left] & \boxed{P_6} \arrow[l, bend left]
\end{tikzcd}$$

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

P has a **finite trace**, written $P \downharpoonleft$, 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 $\downharpoonleft$) $P \downharpoonleft$ if
(1) P is stopped
(2) $\exists\, P'$ with $P \longrightarrow P'$ and $P' \downharpoonleft$

as rules:

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

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, informally

1. sets of elements with a certain **proof trees**
2. sets satisfying a certain **closure property**
   (from which we derive the familiar method of proofs by induction)
3. sets obtained via a certain **iteration schema**

We will do the same for coinduction

Then we formally justify the 3 readings, from fixed-point theory
First (2), then (3), then (1).

We will also see another reading, as **games**

Equivalent readings for $\Downarrow$

$$\frac{P \text{ stopped}}{P \Downarrow}\ (\textsc{Ax}) \qquad\qquad \frac{P \longrightarrow P' \qquad P' \Downarrow}{P \Downarrow}\ (\textsc{Inf})$$

– The processes obtained with a finite proof from the rules

Equivalent readings for $\downharpoonright$

$$\frac{P \text{ stopped}}{P \downharpoonright}\ (\textsc{ax}) \qquad\qquad \frac{P \longrightarrow P' \qquad P' \downharpoonright}{P \downharpoonright}\ (\textsc{inf})$$

– The processes obtained with a finite proof from the rules

Example

$$\frac{P_1 \longrightarrow P_2 \qquad \dfrac{P_2 \longrightarrow P_7 \qquad \dfrac{P_7 \text{ stopped}}{P_7 \downharpoonright}\ (\textsc{ax})}{P_2 \downharpoonright}\ (\textsc{inf})}{P_1 \downharpoonright}\ (\textsc{inf})$$

$$\begin{tikzcd}
 & \boxed{P_1} \arrow[r, bend left] \arrow[d] & \boxed{P_2} \arrow[l, bend left] \arrow[r] \arrow[d] & \boxed{P_7} \\
\boxed{P_5} & \boxed{P_3} \arrow[l] & \boxed{P_4} \arrow[r, bend left] & \boxed{P_6} \arrow[l, bend left]
\end{tikzcd}$$

Equivalent readings for $\downharpoonright$

$$\frac{P \text{ stopped}}{P \downharpoonright}\ (\textsc{Ax}) \qquad\qquad \frac{P \longrightarrow P' \qquad P' \downharpoonright}{P \downharpoonright}\ (\textsc{Inf})$$

– The processes obtained with a finite proof from the rules

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

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

$$\begin{tikzcd}
 & \boxed{P_1} \arrow[r, bend left] \arrow[d] & \boxed{P_2} \arrow[l, bend left] \arrow[r] \arrow[d] & \boxed{P_7} \\
\boxed{P_5} & \boxed{P_3} \arrow[l] & \boxed{P_4} \arrow[r, bend left] & \boxed{P_6} \arrow[l, bend left]
\end{tikzcd}$$

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 $\downarrow$

$$\frac{P \text{ stopped}}{P \downarrow}\ (\textsc{Ax}) \qquad\qquad \frac{P \longrightarrow P' \qquad P' \downarrow}{P \downarrow}\ (\textsc{Inf})$$

– 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
  - $\ast$ all stopped processes are in S;
  - $\ast$ 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:

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

(f can have any value, or even be undefined, if the set on which the `min` is taken is empty)

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

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

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 $\Downarrow$

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

– 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)

$$\begin{tikzcd}
 & \boxed{P_1} \arrow[r, bend left] \arrow[d] & \boxed{P_2} \arrow[l, bend left] \arrow[r] \arrow[d] & \boxed{P_7} \\
\boxed{P_5} & \boxed{P_3} \arrow[l] & \boxed{P_4} \arrow[r, bend left] & \boxed{P_6} \arrow[l, bend left]
\end{tikzcd}$$

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 $\upharpoonright$:

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

Equivalent readings for $\downharpoonright$

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

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

Equivalent readings for $\downharpoonright$

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

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

Example

$$\cfrac{P_1 \longrightarrow P_2 \qquad \cfrac{P_2 \longrightarrow P_1 \qquad \cfrac{P_1 \longrightarrow P_2 \qquad \cfrac{\vdots}{P_2 \upharpoonright}}{P_1 \upharpoonright}}{P_2 \upharpoonright}}{P_1 \upharpoonright}$$

$$\begin{tikzcd}
 & \boxed{P_1} \arrow[r, bend left] \arrow[d] & \boxed{P_2} \arrow[l, bend left] \arrow[r] \arrow[d] & \boxed{P_7} \\
\boxed{P_5} & \boxed{P_3} \arrow[l] & \boxed{P_4} \arrow[r, bend left] & \boxed{P_6} \arrow[l, bend left]
\end{tikzcd}$$

Equivalent readings for $\upharpoonright$

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

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

An invalid proof:

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

$$\begin{tikzcd}
 & \boxed{P_1} \arrow[r, bend left] \arrow[d] & \boxed{P_2} \arrow[l, bend left] \arrow[r] \arrow[d] & \boxed{P_7} \\
\boxed{P_5} & \boxed{P_3} \arrow[l] & \boxed{P_4} \arrow[r, bend left] & \boxed{P_6} \arrow[l, bend left]
\end{tikzcd}$$

Equivalent readings for $\downharpoonleft$

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

– 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 $\downharpoonleft$

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

- 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 $\upharpoonright$ (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

$$\begin{tikzcd}
 & P_1 \arrow[r, bend left] \arrow[d] & P_2 \arrow[l, bend left] \arrow[r] \arrow[d] & P_7 \\
P_5 & P_3 \arrow[l] & P_4 \arrow[r, bend left] & P_6 \arrow[l, bend left]
\end{tikzcd}$$

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

Suppose we want to prove $P_1 \upharpoonright$

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 $P \upharpoonright$, 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 $P \mathrel{\mathcal{R}} Q$:

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

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

Equivalent readings for $\downharpoonleft$

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

- 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

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

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

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

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:

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

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) $\boldsymbol{\lambda}$-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

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

$\wp(X)$ 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

$$\begin{tikzcd}
\bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \\
\bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[u] \\
\bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[u] \\
\bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[u] \\
\bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[r] \arrow[u] & \bullet \arrow[u]
\end{tikzcd}$$

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

$$\begin{tikzcd}
\boldsymbol{a} & \boldsymbol{b} \\
\boldsymbol{c} \arrow[u] \arrow[ur] & \boldsymbol{d} \arrow[l] \arrow[u] \\
\boldsymbol{e} \arrow[u] & \boldsymbol{f} \arrow[u]
\end{tikzcd}$$

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

Bounds, meets, and joins

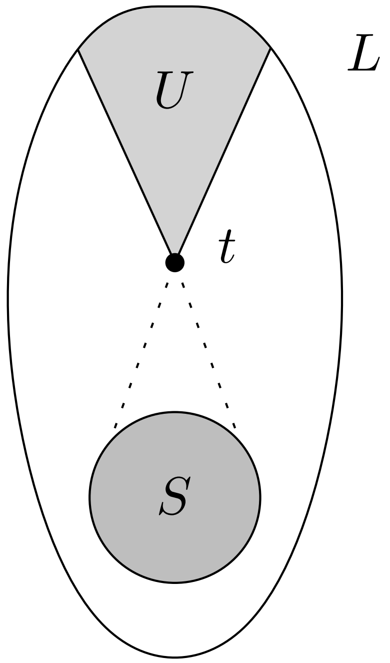
L poset, $S \subseteq L$: then $y \in L$ is an **upper bound of S** if $x \leq y \; \forall \; x \in S$.

The dual: a **lower bound of S**: (a $y \in L$ with $y \leq x$ for all $x \in S$)

The **least upper bound** (also called the **join**) of S: an upper bound y with $y \leq z \; \forall$ upper bounds z of S

The dual: of these concepts gives us the **greatest lower bound** (or **meet**)

Example:
U = the upper bounds for S
t = the least upper bound



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**

Example

the poset of the (positive) natural numbers with $n \leq m$ if n divides m

For $S = \{4, 8, 16\}$:

- $\{1, 2, 4\}$ = the set of lower bounds of S

- 4 is the least element in S and its meet;

- 16 is the greatest element and the join

For $S = \{2, 3, 4\}$:

- 1 is the only lower bound and the meet;

- there is no least element; any multiple of 12 is an upper bound, 12 is the join; no greatest element

The endofunction F where $F(n)$ is the sum of the factors of n that are different from n, with the exception of 1 that is mapped onto itself

eg: $F(1) = 1, F(2) = 1, F(3) = 1, F(4) = 3, F(6) = 6$.

Then $1, 2, 3, 6$ are pre-fixed points, and $1, 6$ fixed points.

Exercise

- Is the set of all natural numbers a complete lattice?
- Is it a lattice (that is, is a poset in which all *pairs* of elements have a join)?
- Can we add elements to the set of natural numbers so as to make it a complete lattice?

Exercise

1. Show that if F is a monotone endofunction on a complete lattice, and x and y are post-fixed points of F, then also $\cup\{x, y\}$ is a post-fixed point.
2. Generalise the previous point to an arbitrary set S of post-fixed points: $\cup S$ is also a post-fixed point. Then dualise the result to pre-fixed points.

For simplicity, we discuss the theorem on the complete lattices generated by the powerset construction

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

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

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

(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

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

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

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

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:

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

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

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

By the Fixed-point Theorem, when F monotone:

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

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

Next objective

We wish to derive the reading (2) of inductive sets
(the smallest sets ‘close forward’ wrt some rules)

And dually for coinductive sets

We have 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 $\dfrac{P \longrightarrow P' \qquad P' \upharpoonright}{P \upharpoonright}$ 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, $\texttt{lfp}(\Phi_{\mathcal{R}})$ and $\texttt{gfp}(\Phi_{\mathcal{R}})$, 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 $\texttt{lfp}(\Phi_{\mathcal{R}}) = \emptyset$.

The examples, revisited

and continued

- the previous examples of rule induction and coinduction reduced to the fixed-point format
- example of application of bisimulation outside concurrency

Finite traces

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

As ground rules, these become:

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

This yields the following functional:

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

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

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

ω-traces

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

As ground rules, this yields:

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

This yields the following functional:

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

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

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

Finite lists ($\mathrm{finLists}$)

The rule functional (from sets to sets) is:

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

$\boldsymbol{F}$ is monotone, and $\mathrm{finLists} = \mathtt{lfp}(\boldsymbol{F})$. (i.e., $\mathrm{finLists}$ 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 $\boldsymbol{\lambda}$-terms, thus they act on the set $\boldsymbol{\Lambda}^0 \times \boldsymbol{\Lambda}^0$. The rule functional for $\Downarrow$, written $\boldsymbol{\Phi}_{\Downarrow}$, is

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

In the case of $\Uparrow$, the rules are on $\boldsymbol{\Lambda}^0$. The rule functional for $\Uparrow$ is

$$\begin{aligned}\boldsymbol{\Phi}_{\Uparrow}(\boldsymbol{T}) \overset{\text{def}}{=}\ & \{e_1\, e_2 \mid e_1 \in \boldsymbol{T}, \} \\ & \bigcup \{e_1\, e_2 \mid e_1 \Downarrow \boldsymbol{\lambda} x.\, e_0 \text{ and } e_0\{{}^{e_2}\!/\!_{x}\} \in \boldsymbol{T}\}.\end{aligned}$$

Example (bisimulation outside concurrency)

Problem: reason about equality on infinite lists (streams), more generally on coinductively defined sets

Objects may be ‘infinite’, induction may not be applicable

We can prove equalities adapting the idea of bisimulation.

The coinductive definition tells us what can be observed

An LTS for lists:

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

$\sim$: the resulting bisimulation

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. $s =$ `nil` implies $t =$ `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:

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

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.

The coinduction/bisimulation proof method on lists

$$f : A \to A$$

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

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

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

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

For all $a \in A$: $\texttt{map}\ f\ (\texttt{iterate}\ f\ a) = \texttt{iterate}\ f\ f(a)$

An LTS for lists: $\langle a \rangle \bullet s \xrightarrow{a} s$

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 $\begin{array}{lll} P & \stackrel{\text{def}}{=} & \texttt{map}\ f\ (\texttt{iterate}\ f\ c) \\ Q & \stackrel{\text{def}}{=} & \texttt{iterate}\ f\ f(c) \end{array}$

Applying the definitions of `iterate`, and of LTS

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

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

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

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

Other induction and coinduction principles

- justification from fixed-point theory
- recursion and corecursion
- enhancements of the principles

Mathematical induction

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

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

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

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

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 $\{y \mid y \mathcal{R} x\}$ 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 $\alpha \in \mathcal{T}$ 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

...

Function definitions by recursion and corecursion

One often finds functions defined by means of systems of equations. Such definitions may follow the schema of **recursion** or **corecursion**.

In a definition by *recursion* the domain of the function is an inductive set.

Examples on the well-founded set of the natural numbers: the factorial function

$$f(0) = 1 \qquad f(n+1) = n \times f(n)$$

An example of structural recursion is the function f that defines the number of λ-abstractions in a λ-term:

$$f(x) = 0 \qquad f(\lambda x.\, e) = 1 + f(e) \qquad f(e\, e') = f(e) + f(e')$$

It is possible to define patterns of equations for well-founded recursion, and prove that whenever the patterns are respected the functions specified exist and are unique. The proof makes use of well-founded induction twice, to prove that such functions exist and to prove its unicity a function defined by **corecursion** produces an element of a coinductive set.

An equation for a corecursive function specifies the immediate observables of the element returned by the function

for instance, if the element is an infinite list, the equation should tell us specify the head of the list.

Examples are the definitions of the functions `map`, `iterate`

As in the case of recursion, so for corecursion one can produce general equation schemata, and prove that any system of equations satisfying the schemata defines a unique function (or unique functions, in case of mutually recursive equations)

Enhancements of the principles

Theorem Let F be a monotone endofunction on a complete lattice L, and y a post-fixed point of F (i.e., $y \leq F(y)$). Then

$$\mathtt{gfp}(F) = \bigcup \{x \mid x \leq F(x \cup y)\}$$

principle of coinduction up-to $\cup$:

Let F be a monotone endofunction on a complete lattice,
and suppose $y \leq F(y)$;
then $x \leq F(x \cup y)$ implies $x \leq \mathtt{gfp}(F)$.

Theorem Let F be a monotone endofunction on a complete lattice L, and $\bullet : L \times L \to L$ an associative function such that:

1. for all $x, y, x', y' \in L$, whenever both $x \leq F(x')$ and $y \leq F(y')$, then $x \bullet y \leq F(x' \bullet y')$;
2. for all x with $x \leq F(x)$ we have both $x \leq x \bullet \mathtt{gfp}(F)$ and $x \leq \mathtt{gfp}(F) \bullet x$.

Then

$$\mathtt{gfp}(F) = \bigcup \{x \mid x \leq F(\mathtt{gfp}(F) \bullet x \bullet \mathtt{gfp}(F))\}$$

principle of coinduction up-to $\mathtt{gfp}$:

Let F be a monotone endofunction on a complete lattice L, and $\bullet : L \times L \to L$ an associative function for which the assumptions (1) and (2) of Theorem above hold; then $x \leq F(\mathtt{gfp}(F) \bullet x \bullet \mathtt{gfp}(F))$ implies $x \leq \mathtt{gfp}(F)$.

Back to bisimulation

- bisimilarity as a fixed point

Bisimulation as a fixed-point

Definition Consider the following function $F_\sim : \wp(Pr \times Pr) \rightarrow \wp(Pr \times Pr)$.
$F_\sim(\mathcal{R})$ 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 $\mathcal{R} \subseteq F_\sim(\mathcal{R})$;
- $\sim = \mathtt{gfp}(F_\sim)$.

Least and greatest fixed points: approximations and constructive proofs

– objective: derive the reading (3) of inductive/coinductive sets (via iteration schema)

Continuity and cocontinuity

The proof of the Fixed-point Theorem is not constructive

We show now constructive proofs, by means of iterative schemata.

Pros:

- approximating/computing least fixed points and greatest fixed points. (at the heart of the algorithms used in tools)
- an alternative way for reasoning

Cons:

- requires properties on functions (continuity/cocontinuity) stronger than monotonicity.

Abbreviations: $\bigcup_i \alpha_i$ for $\bigcup_i \{\alpha_i\}$, and $\bigcup_i F(\alpha_i)$ for $\bigcup_i \{F(\alpha_i)\}$; similarly for $\bigcap_i \alpha_i$ and $\bigcap_i F(\alpha_i)$.

Definition An endofunction on a complete lattice is:

- **continuous** if for all sequences $T_0, T_1 \dots$ of increasing points in the lattice (i.e., $T_i \subseteq T_{i+1}$, for $i \geq 0$) we have $F(\bigcup_i T_i) = \bigcup_i F(T_i)$.
- **cocontinuous** if for all sequences $T_0, T_1 \dots$ of decreasing points in the lattice (i.e., $T_{i+1} \subseteq T_i$, for $i \geq 0$) we have $F(\bigcap_i T_i) = \bigcap_i F(T_i)$.

Example: the complete lattice made of the integers plus the points ω and $-\omega$, with the ordering $-\omega \leq n \leq \omega$ for all n.

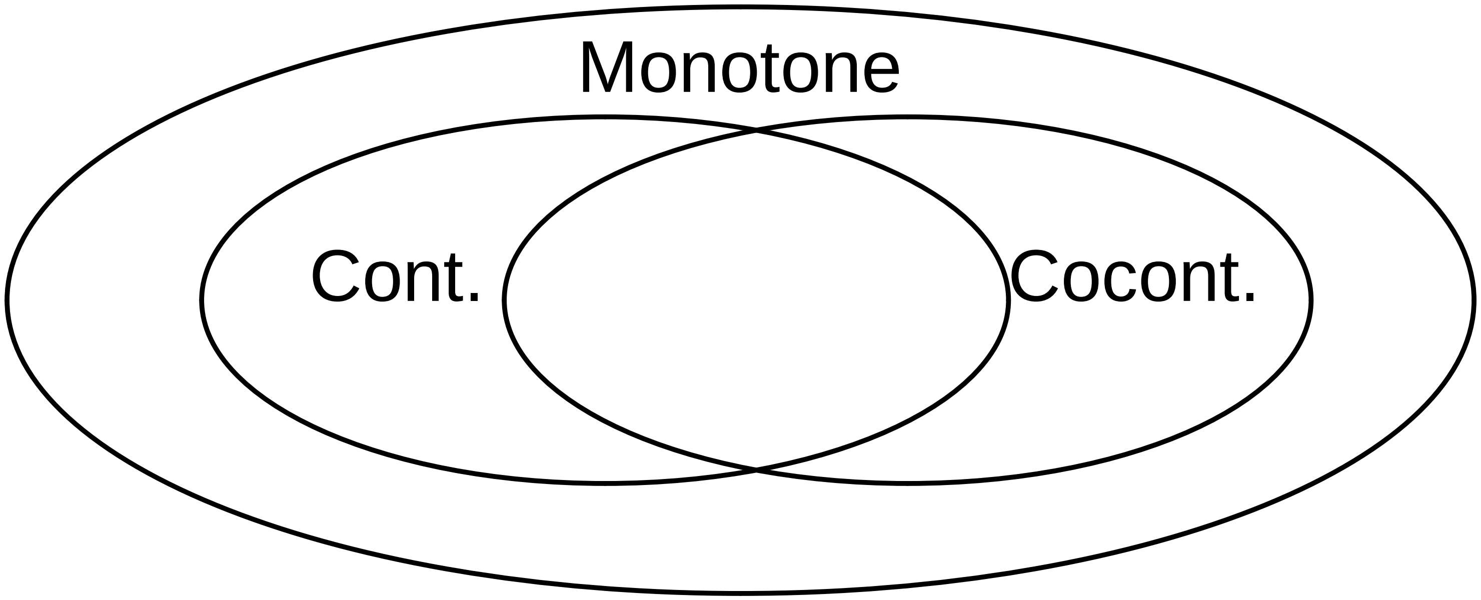
Take F with: $F(n) = n + 1$, $F(\omega) = \omega$, $F(-\omega) = -\omega$

Then $F(\cup\{3, 4, 6\}) = F(6) = 7 = \cup\{4, 5, 7\} = \cup\{F(3), F(4), F(6)\}$

For the increasing sequence of the positive integers, we have $F(\cup_i n_i) = F(\omega) = \omega = \cup_i n_{i+1} = \cup_i F(n_i)$.

Dually, for the decreasing sequence of the negative integers, we have $F(\cap_i - n_i) = -\omega = \cap_i F(-n_i)$.

Exercise If F is cocontinuous (or continuous), then it is also monotone. (Hint: Take $x \geq y$, and the sequence $x, y, y, y, \ldots$.) $\square$



$F^n(x)$ indicates the n-th iteration of F starting from the point x:

$$\begin{aligned} F^0(x) &\overset{\text{def}}{=} x \\ F^{n+1}(x) &\overset{\text{def}}{=} F(F^n(x)) \end{aligned}$$

Then we set:

$$\begin{aligned} F^{\cap\omega}(x) &\overset{\text{def}}{=} \bigcap_{n\geq 0} F^n(x) \\ F^{\cup\omega}(x) &\overset{\text{def}}{=} \bigcup_{n\geq 0} F^n(x) \end{aligned}$$

Theorem [Continuity/Cocontinuity] Let F be an endofunction on a complete lattice, in which $\bot$ and $\top$ are the bottom and top elements.

- If F is continuous, then $\mathtt{lfp}(F) = F^{\cup\omega}(\bot)$;
- if F is cocontinuous, then $\mathtt{gfp}(F) = F^{\cap\omega}(\top)$.

$F^0(\bot), F^1(\bot), \ldots$ is increasing, $F^0(\top), F^1(\top), \ldots$ is decreasing.

lfp and gfp of F are the join and meet of the two sequences.

Exercise Prove the Continuity/Cocontinuity Theorem. (Hint: Referring to the second part, first show that $F^{\cap\omega}$ is a fixed point, exploiting the definition of cocontinuity; then show that it is the greatest fixed point, exploiting the definition of meet.)

If F is not cocontinuous, and only monotone, we only have $\mathtt{gfp}(F) \leq F^{\cap\omega}(\top)$. The converse need not hold

However, when $F^{\cap\omega}(\top)$ is a fixed point, then surely it is the greatest fixed point.

Example Let L be the set of negative integers plus the elements $-\omega$ and $-(\omega+1)$, with the expected ordering $-n \geq -\omega \geq -(\omega+1)$, for all n. Let now F be the following function on L:

$$\begin{array}{rcl} F(-n) & = & -(n+1) \\ F(-\omega) & = & -(\omega+1) \\ F(-(\omega+1)) & = & -(\omega+1) \end{array}$$

The top and bottom elements are -1 and $-(\omega+1)$. Function F is monotone but not cocontinuous, and we have $F^{\cap\omega}(-1) = -\omega$ and $\mathtt{gfp}(F) = -(\omega+1)$.

Having only monotonicity, to reach the greatest fixed point using induction, we need to iterate over the transfinite ordinals.

(The dual statement, for continuity and least fixed points, also holds.)

Theorem Let F be a monotone endofunction on a complete lattice L, and define $F^{\lambda}(\top)$, where λ is an ordinal, as follows:

$$\begin{array}{rcll} F^{0}(\top) & \overset{\text{def}}{=} & \top & \\ F^{\lambda+1}(\top) & \overset{\text{def}}{=} & F(F^{\lambda}(\top)) & \text{for successor ordinals} \\ F^{\lambda}(\top) & \overset{\text{def}}{=} & F(\bigcap_{\beta<\lambda} F^{\beta}(\top)) & \text{for limit ordinals} \end{array}$$

and then $F^{\infty}(\top) \overset{\text{def}}{=} \bigcap_{\lambda} F^{\lambda}(\top)$. We have:

$$F^{\infty}(\top) = \mathtt{gfp}(F).$$

Continuity and cocontinuity, for rules

The functional given by a set of rules need not be continuous or cocontinuous.

Example: Take the rule

$$\frac{a_1 \quad \ldots \quad a_n \quad \ldots}{a}$$

and call ϕ its functional, $T_n = \{a_1, \ldots, a_n\}$.
Then $a \in \phi(\bigcup_n T_n)$, but $a \notin \bigcup_n \phi(T_n)$ (hence ϕ not continuous)

We can recover continuity and cocontinuity for rule functionals adding some conditions.

Definition A set $\mathcal{R}$ of rules is **finite in the premises**, briefly **FP**, if for each rule $(S, x) \in \mathcal{R}$ the premise set S is finite.

Exercise Show that if the set of rules $\mathcal{R}$ is FP, then $\Phi_{\mathcal{R}}$ is continuous; conclude that $\mathtt{lfp}(\Phi_{\mathcal{R}}) = \Phi_{\mathcal{R}}^{\cup\omega}(\emptyset)$.

FP does not work for cocontinuity

Example. $X = \{b\} \cup \{a_1, \ldots, a_n, \ldots\}$, and rules $\dfrac{a_i}{b} \quad \forall\, i$

call Φ the corresponding rule functional.

Thus $\Phi(T) = \{b\}$ if there is i with $a_i \in T$, otherwise $\Phi(T) = \emptyset$.

Take the sequence of decreasing sets $T_0, \ldots, T_n, \ldots$, where

$$T_i \overset{\text{def}}{=} \{a_j \mid j \geq i\}$$

We have $\Phi(\bigcap_n T_n) = \emptyset$, but $\bigcap_n \Phi(T_n) = \{b\}$.

To obtain cocontinuity we need some finiteness conditions on the conclusions of the rules (rather than on the premises as for continuity).

Definition A set of rules $\mathcal{R}$ is **finite in the conclusions**, briefly FC, if for each x, the set $\{S \mid (S, x) \in \mathcal{R}\}$ is finite (i.e., there is only a finite number of rules whose conclusion is x)

Each premise set S may itself be infinite

Theorem If a set of rules $\mathcal{R}$ is FC, then $\Phi_{\mathcal{R}}$ is cocontinuous.

Exercise Prove the theorem above.

Corollary If a set of rules $\mathcal{R}$ on X is FC, then $\mathtt{gfp}(\Phi_{\mathcal{R}}) = \Phi_{\mathcal{R}}^{\cap\omega}(X)$.

Without FC, and therefore without cocontinuity, we have nevertheless $\mathtt{gfp}(\Phi_{\mathcal{R}}) \subseteq \Phi_{\mathcal{R}}^{\cap\omega}(X)$.

With the FP or FC hypothesis we are thus able of applying the Continuity/Cocontinuity Theorem.

For FP and continuity, the theorem tells us that given some rules $\mathcal{R}$, the set inductively defined by $\mathcal{R}$ can be obtained as the limit of the increasing sequence of sets

$$\emptyset, \Phi_{\mathcal{R}}(\emptyset), \Phi_{\mathcal{R}}(\Phi_{\mathcal{R}}(\emptyset)), \Phi_{\mathcal{R}}(\Phi_{\mathcal{R}}(\Phi_{\mathcal{R}}(\emptyset))), \ldots .$$

That is, we construct the inductive set thus:

- start with the empty set
- add the conclusions of the axioms in $\mathcal{R}$ ($\Phi_{\mathcal{R}}(\emptyset)$),
- repeatedly add elements following the inference rules in $\mathcal{R}$ in a 'forward' manner

This corresponds to the usual constructive way of interpreting inductively a bunch of rules

As usual, the case for coinductively defined sets is dual.

The iterative reading, for the finite-trace example

Exercise The rules for finite traces:

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

Show that:

- $P \in \Phi^{n}_{\mathcal{R}_{\downarrow}}(\emptyset)$, for $0 \leq n$, if and only if there are $0 \leq m \leq n$, processes $P_0, \ldots, P_m$, and actions $\mu_1, \ldots, \mu_m$ with $P = P_0$ and such that $P_0 \xrightarrow{\mu_1} P_1 \ldots \xrightarrow{\mu_m} P_m$ and P_m is stopped.

At step 0 we have the empty set; then at step 1 we add the stopped processes; at step 2 we add the processes that have a stopped derivative; and so on.

In applications in which the set of all processes is finite, the sequence $\{\Phi^{n}_{\mathcal{R}_{\downarrow}}(\emptyset)\}_n$ will not increase forever

The iterative reading, for the ω-trace example

Recall that the ground rules are:

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

Exercise For *Pr* = all processes, show that:

- $P \in \Phi^n_{\mathcal{R}_{\upharpoonright}}$(*Pr*), for $0 \leq n$, if and only if there are processes $P_0, \ldots, P_n$ with $P = P_0$ and such that $P_0 \xrightarrow{\mu} P_1 \ldots \xrightarrow{\mu} P_n$.

In the sequence

$$\Phi^0_{\mathcal{R}_{\upharpoonright}}(Pr), \Phi^1_{\mathcal{R}_{\upharpoonright}}(Pr), \Phi^2_{\mathcal{R}_{\upharpoonright}}(Pr), \cdots$$

at step 0 we have the set *Pr* of all processes; at step 1 we remove the processes that do not have a μ-derivative; at step 2 the processes that cannot perform 2 consecutive μ-transitions; and so on.

If the set of processes is finite, the sequence will not decrease forever

Approximants of bisimilarity

Here is a natural definition of approximants of bisimilarity, where *Pr* = the states of an LTS

- $\sim_0 \overset{\text{def}}{=} \mathit{Pr} \times \mathit{Pr}$;
- $P \sim_{n+1} Q$, for $n \geq 0$, if for all μ:
  1. for all P' with $P \xrightarrow{\mu} P'$, there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \sim_n Q'$;
  2. the converse, i.e., for all Q' with $Q \xrightarrow{\mu} Q'$, there is P' such that $P \xrightarrow{\mu} P'$ and $P' \sim_n Q'$;
- $\sim_\omega \overset{\text{def}}{=} \bigcap_{n \geq 0} \sim_n$.

At stage n, we check transitions up to depth n.

There is an exact correspondence with the the sequence

$$F^0_\sim(\mathit{Pr} \times \mathit{Pr}), F^1_\sim(\mathit{Pr} \times \mathit{Pr}), F^2_\sim(\mathit{Pr} \times \mathit{Pr}), \cdots$$

where $F_\sim$ is the functional of bisimilarity

Recall that $F_\sim(\mathcal{R})$ 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 $P' \mathcal{R} Q'$.

Exercise

1. $\sim_0, \ldots, \sim_n, \ldots$ is a decreasing sequence of relations.
2. For all $0 \leq n < \omega$, we have $\sim_n = F_\sim^n(Pr \times Pr)$, and $\sim_\omega = F_\sim^{\bigcap\omega}(Pr \times Pr)$, where $F_\sim^n$ and $F_\sim^{\bigcap\omega}$ are the iterations of $F_\sim$ following the definitions used in the Cocontinuity Theorem

NB: Approximants can also be usefully employed to prove non-bisimilarity results

Bisimulation and cocontinuity

A counterexample to $\sim \, = \, \sim_\omega$. Take the states and transitions:

$$\begin{array}{lll} a^0 & \overset{\text{def}}{=} & 0 \\ a^\omega & \xrightarrow{a} & a^\omega \\ a^n & \xrightarrow{a} & a^{n-1} \quad \text{for } n \geq 1 \end{array}$$

Now let P, Q be states with transitions

$$P \xrightarrow{a} a^n \quad \text{for all } n \geq 0$$

and

$$\begin{array}{ll} Q \xrightarrow{a} a^n & \quad \text{for all } n \geq 0 \\ Q \xrightarrow{a} a^\omega & \end{array}$$

$P \sim_n Q$ for all n (simple induction), hence also $P \sim_\omega Q$.

$P \not\sim Q$, as $Q \xrightarrow{a} a^\omega$ can only be matched by $P \xrightarrow{a} a^n$, for some n

Exercise Show formally that the functional $F_\sim$ of bisimilarity is not cocontinuous.

A sufficient condition: finite branching

We can obtain $\sim$ by iteration over the natural numbers if we add some finiteness hypothesis on the branching structure of the LTS.

An LTS is **finitely-branching** if for each process the set of its (immediate) derivatives is finite.

Theorem On finitely-branching LTSs, $\sim = \sim_\omega$.

The thereom follows from the following exercise and the Cocontibuity Theorem.

Exercise Check that under the finitely-branching hypothesis the functional $\boldsymbol{F}_\sim$ is cocontinuous.

A direct proof is useful to understand the hypothesis

Proof The inclusion $\sim \subseteq \sim_\omega$ is easy: one proves that $\sim \subseteq \sim_n$ for all n, using the fact that $\sim$ is a bisimulation

(or, using the fact that $\sim$ is a fixed point of $F_\sim$, monotonicity of $F_\sim$, and $\sim_{n+1} = F_\sim(\sim_n)$; we can also directly derive it from fixed-point theory).

Now the converse. We show that the set

$$\mathcal{R} \overset{\text{def}}{=} \{(P, Q) \mid P \sim_\omega Q\}$$

is a bisimulation. Take $(P, Q) \in \mathcal{R}$ and suppose $P \xrightarrow{\mu} P'$.

$\forall\, n$, as $P \sim_{n+1} Q$, $\exists\, Q_n$ s.t. $Q \xrightarrow{\mu} Q_n$ and $P \sim_n Q_n$.

As the LTS is finitely-branching, the set $\{Q_i \mid Q \xrightarrow{\mu} Q_i\}$ is finite.

Hence $\exists\, Q_i$ s.t. $P' \sim_n Q_i$ for infinitely many n.

As $\{\sim_n\}_n$ is decreasing with n, we have $P' \sim_n Q_i \ \forall\, n$.

Hence $P' \sim_\omega Q_i$ and $(P', Q_i) \in \mathcal{R}$. $\square$

Algorithms for bisimilarity

The stratification of bisimilarity given by continuity is also the basis for **algorithms** for mechanically checking bisimilarity and for minimisation of the state-space of a process

These algorithms work on processes that are **finite-state** (ie, each process has only a finite number of possible derivaties)

They proceed by progressively refining a partition of all processes

In the initial partition, all processes are in the same set

Bisimulation: P-complete **[Alvarez, Balcazar, Gabarro, Santha, '91]**

With m transitions, n states:
O(m log n) time and O(m + n) space **[Paige, Tarjan, '87]**

Trace equivalence, testing: PSPACE-complete
[Kannelakis, Smolka, '90; Huynh, Tian, 95]

Other views on the meaning of induction and coinduction

- derivation proofs
  (cf: the informal reading (1) of inductive and coinductive sets)
- games

Proof tree interpretations

A set of ground rules is used to derive elements, via proof trees

The example of lists. The rules are $\dfrac{}{\texttt{nil}} \quad \dfrac{s \qquad a}{\langle a\rangle \bullet s} \qquad (\forall s \in X, a \in A)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{nil}}}{\langle b\rangle \bullet \texttt{nil}}}{\langle a\rangle \bullet \langle b\rangle \bullet \texttt{nil}}}{\langle a\rangle \bullet \langle a\rangle \bullet \langle b\rangle \bullet \texttt{nil}} \qquad \dfrac{\dfrac{\dfrac{\ldots}{\langle a\rangle \bullet \langle b\rangle \bullet \langle a\rangle \bullet \langle b\rangle \bullet \ldots}}{\langle b\rangle \bullet \langle a\rangle \bullet \langle b\rangle \bullet \langle a\rangle \bullet \langle b\rangle \bullet \ldots}}{\langle a\rangle \bullet \langle b\rangle \bullet \langle a\rangle \bullet \langle b\rangle \bullet \langle a\rangle \bullet \langle b\rangle \bullet \ldots}$$

(in general — but not with lists — a node of a tree may have several children)

$\langle a\rangle \bullet \langle a\rangle \bullet \langle b\rangle \bullet \texttt{nil}$ is both in the inductive and in the coinductive set

$\langle a\rangle \bullet \langle b\rangle \bullet \langle a\rangle \bullet \langle b\rangle \bullet \ldots$ in only in the coinductive set

What is the difference between induction and coinduction on the meaning of 'correct' proof tree?

Trees (informally)

The set of **trees over X** is the set of all trees, possibly infinite both in depth and in breadth, in which each node is labelled with an element from the set X and, moreover, the labels of the children of a node are pairwise distinct.

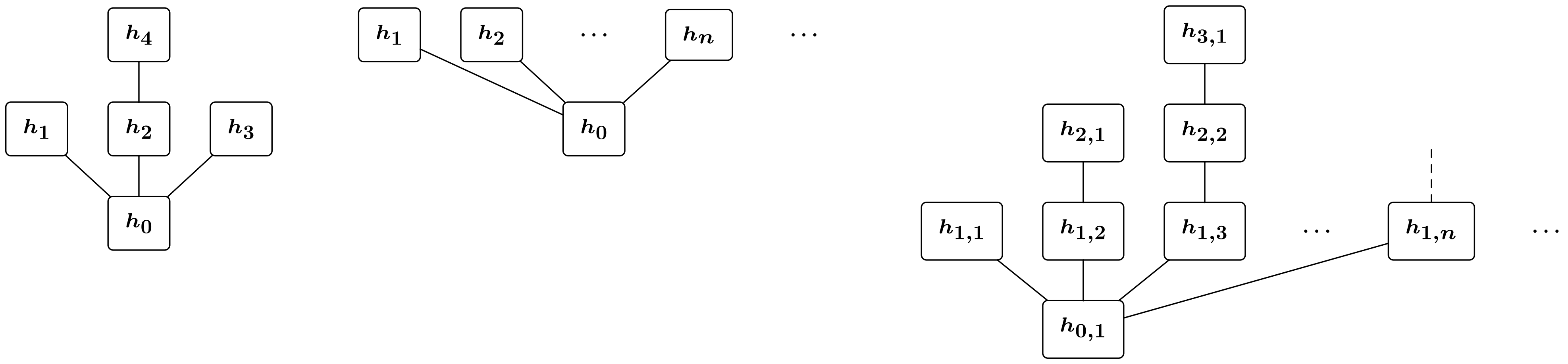
If $\mathcal{T}$ is such a tree, then the **root** of $\mathcal{T}$ is the only node without a parent.

Let $\mathcal{R}$ be a set of ground rules.

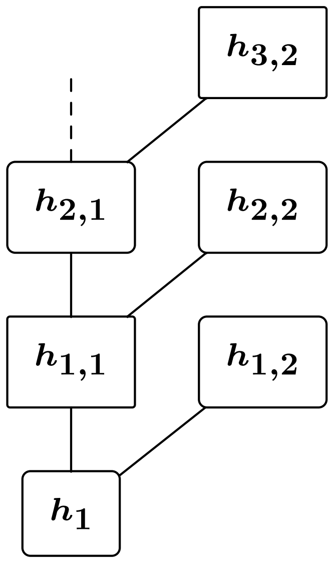
A tree $\mathcal{T}$ is **a proof tree for $x \in X$ under $\mathcal{R}$** if x is the label of the root of $\mathcal{T}$ and, for each node h with label y, if S is the set of the labels of all children of h, then (S, y) is a rule in $\mathcal{R}$.

A tree is *non-well-founded* if it has paths of infinite length. It is *well-founded* otherwise.

Some well-founded trees:



A non-well-founded tree:



Theorem $x \in \mathtt{lfp}(\Phi_{\mathcal{R}})$ iff there is a well-founded proof tree for x under $\mathcal{R}$.

Proof: reason on the approximants

With FP hypothesis, each node only has finitely many children, and therefore a well-founded proof tree has a finite height (hence it is finite)

In the examples of traces, λ-calculus, lists, the rules are FP, hence the inductive objects are those with a ‘finite derivation proof’.

Without FP, a well-founded proof tree need not have a finite height.

Theorem $x \in \mathtt{gfp}(\Phi_{\mathcal{R}})$ iff there is a proof tree for x under $\mathcal{R}$.

Proof First, the direction from left to right.

If $x \in \mathtt{gfp}(\Phi_{\mathcal{R}})$, then x is in some post-fixed point of $\Phi_{\mathcal{R}}$; that is, there is T with $x \in T$ and $T \subseteq \Phi_{\mathcal{R}}(T)$.

Now, as $T \subseteq \Phi_{\mathcal{R}}(T)$, by definition of $\Phi_{\mathcal{R}}$, for each $y \in T$ there is at least one rule (S, y) in $\mathcal{R}$ with $S \subseteq T$; we pick one of these rules and call it R_y.

The proof tree for x is defined as follows. The root is x. The children of a node y in the tree are the nodes $y_1, \ldots, y_n$ that form the premise of the rule R_y chosen for y.

Theorem $x \in \mathtt{gfp}(\Phi_{\mathcal{R}})$ iff there is a proof tree for x under $\mathcal{R}$.

Proof Now the converse direction (right to left).

Suppose there is a proof tree for x.

Let T be the set of all the (labels of) nodes in the tree. We show that T is a post-fixed point of $\Phi_{\mathcal{R}}$.

We have to show that any $y \in T$ is in $\Phi_{\mathcal{R}}(T)$.

If $y \in T$ then there is a node in the tree that is labelled y.

Let $\{y_1, \dots, y_n\}$ be the set of the labels of the children of such node.

By definition of proof tree, $(\{y_1, \dots, y_n\}, y)$ is a rule in $\mathcal{R}$ and, by definition of T, we have $(\{y_1, \dots, y_n\} \subseteq T$.

Hence $y \in \Phi_{\mathcal{R}}(T)$.

Game interpretations

A game-theoretic characterisation of induction and coinduction, exploiting some of the ideas in the ‘proof-tree’ presentation

Consider a set $\mathcal{R}$ of ground rules (on X).

A **game in $\mathcal{R}$** has:

- two players (the verifier V and the refuter R)
- an element $x_0 \in X$

V tries to show that a proof tree for x_0 exists; R tries to dispute that

A play

Thus **a play for $\mathcal{R}$ and x_0** is a sequence

$$x_0, S_0, \ldots, x_n, S_n, \ldots$$

which can be finite or infinite.

It goes thus:

- V chooses a set S_0 s.t. $(S_0, x_0) \in \mathcal{R}$ (i.e., x_0 can be derived from S_0)
- R chooses an element $x_1 \in S_0$ (thus challenging V to continue)
- V has to find a set S_1 with $(S_1, x_1) \in \mathcal{R}$
- R picks $x_2 \in S_1$
- and so on.

Example from the λ-calculus

The ground rules $\mathcal{R}$ for the divergence predicate $\Uparrow$ of the λ-calculus are

$$\frac{e}{e\ e'} \qquad\qquad \frac{e}{e_1\ e_2} \text{ with } e_1 \Downarrow \lambda x.\,e_0 \text{ for some } e_0 \text{ with } e_0\{{}^{e_2}\!/\!_x\} = e$$

For $e_1 = \lambda x.\,xx$, a play is

$$e_1\,e_1,\ \{e_1\,e_1\},\ e_1\,e_1,\ \ldots$$

For $e_2 = \lambda x.\,xxx$, a play is

$$e_2\,e_2,\ \{(e_2\,e_2)\,e_2\},\ (e_2\,e_2)\,e_2,\ \{e_2\,e_2\},\ \ldots$$

Both plays, in the coinductive game, represent win for V. A finite play for $e_2\,e_2$ is

$$e_2\,e_2,\ \{e_2\},\ e_2$$

and it is a win for R.

Example with finite traces

The ground rules $\mathcal{R}_{\downharpoonleft}$ for the finite-trace predicate $\downharpoonleft$ are

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

$$\begin{tikzcd} P_4 & P_1 \arrow[l, "a"'] \arrow[r, bend left, "b"] & P_2 \arrow[l, bend left, "b"] \arrow[r, "a"] & P_3 \end{tikzcd}$$

A play for $\mathcal{R}_{\downharpoonleft}$ and P_1 is

$$P_1,\ \{P_2\},\ P_2,\ \{P_1\},\ P_1, \ldots$$

where V follows the b-transitions; another play is

$$P_1,\ \{P_2\},\ P_2,\ \emptyset$$

where V follows an a-transition. The latter play is a win for V. In the inductive game, the first play is a win for R.

Inductive vs coinductive games

The game is **finite** if at some point one of the players is unable to make the move; then the other player wins.

(e.g., V's last move was the empty set $\emptyset$; R's last move was an element x that does not appear in conclusions of the rules $\mathcal{R}$)

An **infinite** game:

- in the inductive world it is a win for R
  (with induction, proofs should must be well-founded)
- in the coinductive world it is a win for V
  (with coinduction, non-well-founded paths in proof trees are allowed)

$\mathcal{G}^{\mathtt{ind}}(\mathcal{R}, x_0)$: the inductive game.

$\mathcal{G}^{\mathtt{coind}}(\mathcal{R}, x_0)$: the coinductive game.

Strategies

A **winning strategy**: a systematic way of playing that always produces a win

Definition In a game $\mathcal{G}^{\mathtt{ind}}(\mathcal{R}, x_0)$ or $\mathcal{G}^{\mathtt{coind}}(\mathcal{R}, x_0)$:
a **strategy for** V is a partial function that associates to each play

$$x_0, S_0, \ldots, x_n, S_n, x_{n+1}$$

a set S_{n+1}, with $(S_{n+1}, x_{n+1}) \in \mathcal{R}$, to be used for the next move for V;
similarly, a **strategy for** R in $\mathcal{G}^{\mathtt{ind}}(\mathcal{R}, x_0)$ or $\mathcal{G}^{\mathtt{coind}}(\mathcal{R}, x_0)$ is a partial function that associates to each play

$$x_0, S_0, \ldots, x_n, S_n$$

an element $x_{n+1} \in S_n$. The strategy of a player is **winning** if that player wins every play in which he/she has followed the strategy.

The strategies for induction and coinduction can actually be history-free

Exercise Analise the winning stragegies for the previous examples

Theorem

1. $x_0 \in \mathtt{lfp}(\Phi_{\mathcal{R}})$ iff player V has a winning strategy in the game $\mathcal{G}^{\mathtt{ind}}(\mathcal{R}, x_0)$;

2. $x_0 \in \mathtt{gfp}(\Phi_{\mathcal{R}})$ iff player V has a winning strategy in the game $\mathcal{G}^{\mathtt{coind}}(\mathcal{R}, x_0)$.

The bisimulation game

As we have seen, there is a standard construction to turn any monotone operator on a complete lattice $\wp(X)$ into a set of rules (and vice versa)

This construction gives us these rules for bisimulation:

$$\frac{\mathtt{Der}(P, Q, f, g)}{(P, Q)}$$

where

- function f maps a pair (μ, P') such that $P \xrightarrow{\mu} P'$ into a process Q' such that $Q \xrightarrow{\mu} Q'$
  Conversely, function g maps a pair (μ, Q') such that $Q \xrightarrow{\mu} Q'$ into a process P' such that $P \xrightarrow{\mu} P'$
- $\mathtt{Der}(P, Q, f, g)$ is the set of process pairs

$$\{(P', f(\mu, P')) \mid P \xrightarrow{\mu} P'\} \cup \{(g(\mu, Q'), Q') \mid Q \xrightarrow{\mu} Q'\}\,.$$

(With non-determinism, there may be several rules with the same conclusion)

In the resulting game interpretation, given a pair (P, Q), the verifier V chooses the functions f and g that determine the pairs $\texttt{Der}(P, Q, f, g)$ needed in the premise.

The refuter R then picks up one of the pairs in $\texttt{Der}(P, Q, f, g)$ to continue the game.

If functions f and g for V do not exist, then V cannot continue and R wins.

When $\texttt{Der}(P, Q, f, g)$ is empty (which happens if both P and Q are stopped), R cannot continue and V wins.

As the game is coinductive, an infinite play represents a win for V.

Example

$$\begin{tikzcd}
 & P_2 & \\
P_1 \arrow[ur, "a"] \arrow[dr, "a"'] & & \\
 & P_3 \arrow[r, "b"'] & P_4
\end{tikzcd}$$

$$\begin{tikzcd}
Q_1 \arrow[r, "a"] & Q_2 \arrow[r, "b"] & Q_3
\end{tikzcd}$$

A play in which V wins:

$$(P_1, Q_1),\ \{(P_2, Q_2), (P_3, Q_2)\},\ (P_3, Q_2),\ \{(P_4, Q_3)\},\ (P_4, Q_3),\ \emptyset$$

A play with a win for R:

$$(P_1, Q_1),\ \{(P_2, Q_2), (P_3, Q_2)\},\ (P_2, Q_2)$$

R has a winning strategy, which consists in following the latter play, thereby always selecting, in the first move, the pair (P_2, Q_2).

A simpler bisimulation game

We formulate the bisimulation game a bit differently, letting R move first

R first chooses a transition, say $P \xrightarrow{\mu} P'$ or $Q \xrightarrow{\mu} Q'$

then V has to find a matching derivative from Q or P

A **play for (P_0, Q_0)** in the new game is a finite or infinite sequence of pairs

$$(P_0, Q_0),\ (P_1, Q_1),\ \cdots,\ (P_i, Q_i),\ \cdots$$

Given (P_i, Q_i), the following pair (P_{i+1}, Q_{i+1}) is determined thus:

- R makes the challenge by choosing either a transition $P_i \xrightarrow{\mu} P'$ or a transition $Q_i \xrightarrow{\mu} Q'$;
- V has to answer, in the former case with a transition $Q_i \xrightarrow{\mu} Q'$, in the latter case with a transition $P_i \xrightarrow{\mu} P'$;
- the pair (P', Q') is $(i + 1)$th one of the play.

If V is unable to answer, then R wins.

If this situation never occurs (at some point R cannot formulate a challenge, or the play is infinite) then V is the winner.

Again, we can define the notion of strategy

Theorem $P \sim Q$ if and only if V has a winning strategy for (P, Q).

Theorem $P \not\sim Q$ if and only if R has a winning strategy for (P, Q).

The game interpretation is also useful to reason about bisimulation, especially to prove non-bisimilarity results.

Example: a winning strategy for R

$$\begin{tikzcd}
 & P_3 \arrow[dl, "a"'] \arrow[dr, "a"] & \\
P_3^1 \arrow[d, "b"'] & & P_3^2 \arrow[d, "b"] \\
P_3^3 \arrow[d, "c"'] & & P_3^4 \arrow[d, "d"] \\
P_3^5 & & P_3^6
\end{tikzcd}$$

$$\begin{tikzcd}
 & Q_3 \arrow[d, "a"] & \\
 & Q_3^1 \arrow[dl, "b"'] \arrow[dr, "b"] & \\
Q_3^2 \arrow[d, "c"'] & & Q_3^3 \arrow[d, "d"] \\
Q_3^4 & & Q_3^5
\end{tikzcd}$$

- The initial transition chosen by R is $P_3 \xrightarrow{a} P_3^1$.
- The only answer for V can be via the transition $Q_3 \xrightarrow{a} Q_3^1$, and the resulting pair is (P_3^1, Q_3^1).
- Now R chooses the transition $Q_3^1 \xrightarrow{b} Q_3^3$, and V has only the transition $P_3^1 \xrightarrow{b} P_3^3$, resulting in the new pair (P_3^3, Q_3^3).
- Finally, R makes the challenge on the transition $Q_3^3 \xrightarrow{d} Q_3^5$, and V cannot answer.

A process calculus: CCS

Imposing structure: an algebraic language of processes

We introduce some common process operators, which impose a structure on processes and bring in concepts from algebra.

Analogy with automata theory: The language of an automata can be described as a regular expression:

$$L ::= L^\star \;\Big|\; L ; L \;\Big|\; L + L \;\Big|\; \emptyset \;\Big|\; a \qquad \text{where } a \in \mathtt{Act}$$

Theorem 1 A language is regular iff it is the language accepted by some automata

The description of automata as regular languages is important: regular languages have an algebra, which can be used for synthesis, analysis, reasoning

For instance, the reduction of non-deterministism to determinism for trace equivalence can be proved using the law

$$(a.\,P) + (a.\,Q) = a.\,(P + Q) \tag{1}$$

Of course, algebra is not the only means for reasoning on automata!
(Example of another means: automata minimisation).

Similarly, in a **process calculus** algebra is very important; but it should not be the only tool
(other tools: logics, induction, coinduction, etc.).

That is why **process calculus** is a better terminology than **process algebra**

Motivation for this part

– introduction to the concept of process calculus

– the Structured Operational Semantics (SOS) style

– robustness of bisimilarity: compositionality, axiomatisation

Towards a process calculus

We wish to isolate a set of basic constructors for concurrency

This is hard: we do not have something like Theorem 1 to guide us.

LTS's tell us what are the processes; but what are the 'computable' processes?

Another analogy: Church's λ-calculus:

functions	LTS's
computable functions	???

The behaviour of proceses: operationally, following Plotkin's **Structured Operational Semantics**.

Robin Milner (the Calculus of Communicating Systems, CCS, end 70s);
Tony Hoare (Communicating Sequential Processes, CSP, similar dates)

Names and transitions in CCS

Interaction: handshaking between two processes.

No value passing, for simplicity.

In an interaction: a process performs an action a and a parallel process the complementary action $\overline{a}$.

P, Q, R range over *processes*; $a, b \ldots$ over *names*

$\overline{a}, \overline{b}$ are *co-names*. Names, co-names and τ form the set `Act` of *actions*.
μ, λ range over `Act`

$P \xrightarrow{a} P'$: P offers an input at port a, thus evolving into P'

$P \xrightarrow{\overline{a}} P'$: similar, but P offers an output at a.

$P \xrightarrow{\tau} P'$: P can internally do some work and then become P' (eg, an interaction between components)

We assume $\overline{\overline{a}} = a$, and τ different from any name or co-name.

The CCS operators

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

where K is a **constant**

Nil, written 0 : a terminated process, no transitions

Prefixing (action sequentialisation)

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

Example: $a.\, b.\, 0 \xrightarrow{a} b.\, 0 \xrightarrow{b} 0$

Parallel composition Complex systems are composed of components, that interact *both* with each other *and* with the environment.

A component can be a register, a bus, a memory, a program, a wire etc.

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

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

$P \overset{\text{def}}{=} (a.\,0 \mid b.\,0) \mid \overline{a}.\,0$ has the transitions

$$\begin{array}{lll} P & \xrightarrow{a} & (0 \mid b.\,0) \mid \overline{a}.\,0 \\ P & \xrightarrow{b} & (a.\,0 \mid 0) \mid \overline{a}.\,0 \end{array} \qquad \begin{array}{lll} P & \xrightarrow{\tau} & (0 \mid b.\,0) \mid 0 \\ P & \xrightarrow{\overline{a}} & (a.\,0 \mid b.\,0) \mid 0 \end{array}$$

Other communication mechanisms (eg, shared variables, or buffered communications) can be modeled

Choice

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

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

Example. $P \overset{\text{def}}{=} (\overline{a}.\, Q_1 \mid a.\, Q_2) + b.\, R$ has the transitions

$$\begin{array}{lll} P & \xrightarrow{\tau} & Q_1 \mid Q_2 \\ P & \xrightarrow{\overline{a}} & Q_1 \mid a.\, Q_2 \end{array} \qquad \begin{array}{lll} P & \xrightarrow{a} & \overline{a}.\, Q_1 \mid Q_2 \\ P & \xrightarrow{b} & R \end{array}$$

Restriction

$$\frac{P \xrightarrow{\mu} P'}{(\nu a)\, P \xrightarrow{\mu} (\nu a)\, P'} \;\textsc{Res}\; \mu \notin \{a, \overline{a}\}$$

In $(\nu a)\, P$: a is private to P, hidden to the external environment

It is a binder (cf: $\lambda a.\, M$ in the λ-calculus)

Example: $P \overset{\text{def}}{=} (\nu a)\, ((a.\, Q_1 \mid \overline{a}.\, Q_2) + b.\, R)$ has transitions

$$P \xrightarrow{\tau} (\nu a)\, (Q_1 \mid Q_2) \qquad \text{and} \qquad P \xrightarrow{b} (\nu a)\, R\, .$$

Exercise

1. Which transitions can the process $P \overset{\text{def}}{=} (\nu a)\, ((a.\, 0 + b.\, 0) \mid \overline{a}.\, 0)$ make?
2. Find a process Q in which there is no parallel composition and restriction and with $P \sim Q$. $\square$

Constants (Recursive process definitions)

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

Example. $K \xrightarrow{a} K$

Example. Let K_1 and K_2 have the transitions

$$\begin{aligned} K_1 &\xrightarrow{a} \overline{c}.\, K_1 \\ K_2 &\xrightarrow{b} c.\, K_2 \end{aligned}$$

The LTS for $(\nu c)\ (K_1 \mid K_2)$ is

$$\begin{tikzcd}
& (\nu c)\ (K_1 \mid K_2) \arrow[dl, "a"'] \arrow[dr, "b"] & \\
(\nu c)\ (\overline{c}.\, K_1 \mid K_2) \arrow[dr, "b"'] & & (\nu c)\ (K_1 \mid c.\, K_2) \arrow[dl, "a"] \\
& (\nu c)\ (\overline{c}.\, K_1 \mid c.\, K_2) \arrow[uu, "\tau"'] &
\end{tikzcd}$$

$\mathrm{CCS}(\mathtt{Act}, \mathcal{C}ons, \mathcal{T}_{\mathcal{C}ons})$

The set of all constants is $\mathcal{C}ons$, and the set of all transitions for the constants in $\mathcal{C}ons$ is $\mathcal{T}_{\mathcal{C}ons} \subseteq \mathcal{C}ons \times \mathtt{Act} \times Pr$.

The transitions for the processes in *Pr* are determined by $\mathcal{T}_{\mathcal{C}ons}$ plus the inferences rules for the operators seen above

Such a process language is called $\mathrm{CCS}(\mathtt{Act}, \mathcal{C}ons, \mathcal{T}_{\mathcal{C}ons})$, abbreviated as CCS, since either the specific sets of actions and constants are not important, or anyhow there are no risks of ambiguities.

Examples bisimilarities

Write Ω_μ for the constant whose only transition is

$$\Omega_\mu \xrightarrow{\mu} \Omega_\mu$$

Thus Ω_μ perpetually performs the action μ. Having two, or more, copies of Ω_μ has no visible effect, that is, for all $n \geq 1$, we have

$$\underbrace{\Omega_\mu \mid \ldots \mid \Omega_\mu}_{n \text{ times}} \sim \Omega_\mu$$

Proof: for each $n \geq 1$, $\{(\underbrace{\Omega_\mu \mid \ldots \mid \Omega_\mu}_{n \text{ times}}, \Omega_\mu)\}$ is a bisimulation.

Exercise

- $a.\, P \mid b.\, Q \sim a.\, (P \mid b.\, Q) + b.\, (a.\, P \mid Q)$
- $a.\, P \mid \overline{a}.\, Q \sim a.\, (P \mid \overline{a}.\, Q) + \overline{a}.\, (a.\, P \mid Q) + \tau.\, (P \mid Q)$

Exercise Show that $(\nu a)\ (a.\, P \mid \overline{a}.\, Q) \sim \tau.\, (\nu a)\ (P \mid Q)$. The restriction forces an interaction between the two parallel components. $\square$

Semaphores

Semaphores are widely used, for instance in operating systems, to protect access to resources.

An n-ary semaphore is used to guarantee that at most n processes have concurrent access to a resource.

A semaphore provides two operations, p and v: the former is invoked in order to obtain access, the latter to signal the completion of the activity

A binary semaphore:

$$K_2^0 \xrightarrow{p} K_2^1 \quad \text{and} \quad K_2^1 \xrightarrow{v} K_2^0$$

where K_2^0 is the initial state, and K_2^1 an auxiliary state indicating that one instance of the resource is active.

A 3-ary semaphore:

$$\begin{array}{ccccccc} K_3^0 & \xrightarrow{p} & K_3^1 & \qquad & K_3^1 & \xrightarrow{p} & K_3^2 \\ K_3^2 & \xrightarrow{v} & K_3^1 & \qquad & K_3^1 & \xrightarrow{v} & K_3^0 \end{array}$$

where the superscript indicates the number of resources active.

We obtain a 3-ary semaphore by composing 2 binary semaphores:

$$K_2^0 \mid K_2^0 \sim K_3^0$$

A bisimulation that proves the equality:

$$\mathcal{R} \stackrel{\text{def}}{=} \{(K_2^0 \mid K_2^0, K_3^0), (K_2^1 \mid K_2^0, K_3^1), (K_2^0 \mid K_2^1, K_3^1), (K_2^1 \mid K_2^1, K_3^2)\}$$

A specification and an implementation of a counter

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

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

and, for $n > 0$,

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

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

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

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

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

Proof

$$\mathcal{R} \overset{\text{def}}{=} \{(\mathtt{C} \mid \Pi_1^n\, \mathtt{down}.\, \mathbf{0}, \mathtt{Counter}_n) \ \mid\ n \geq 0\}\,,$$

is a bisimulation up-to $\sim$

Take $(\mathtt{C} \mid \Pi_1^n\, \mathtt{down}.\, \mathbf{0}, \mathtt{Counter}_n)$ in $\mathcal{R}$.

Suppose $\mathtt{C} \mid \Pi_1^n\, \mathtt{down}.\, \mathbf{0} \xrightarrow{\mu} P$.

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

$\mu = \mathtt{up}$. the transition from $\mathtt{C} \mid \Pi_1^n\, \mathtt{down}.\, \mathbf{0}$ originates from $\mathtt{C}$, which performs the transition $\mathtt{C} \xrightarrow{\mathtt{up}} \mathtt{C} \mid \mathtt{down}.\, \mathbf{0}$, and $P = \mathtt{C} \mid \Pi_1^{n+1}\, \mathtt{down}.\, \mathbf{0}$. 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: $(\mathtt{C} \mid \Pi_1^n\, \mathtt{down}.\,\mathbf{0}, \mathtt{Counter}_n)$

Action: $\mathtt{C} \mid \Pi_1^n\, \mathtt{down}.\,\mathbf{0} \xrightarrow{\mu} P$

$\mu = \mathtt{down}$. It must be $n > 0$. The action must originate from one of the $\mathtt{down}.\,\mathbf{0}$ components of $\Pi_1^n\, \mathtt{down}.\,\mathbf{0}$, which has made the transition $\mathtt{down}.\,\mathbf{0} \xrightarrow{\mathtt{down}} \mathbf{0}$.

Therefore $P = \mathtt{C} \mid \Pi_1^n\, P_i$, where exactly one P_i is $\mathbf{0}$ and all the others are $\mathtt{down}.\,\mathbf{0}$.

we have: $P \sim \mathtt{C} \mid \Pi_1^{n-1}\, \mathtt{down}.\,\mathbf{0}$.

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

This closes the diagram, for $P' \overset{\text{def}}{=} \mathtt{C} \mid \Pi_1^{n-1}\, \mathtt{down}.\,\mathbf{0}$ and $Q \overset{\text{def}}{=} \mathtt{Counter}_{n-1}$, as $P' \mathcal{R}\, Q$.

The case when $\mathtt{Counter}_n$ moves first and $\mathtt{C} \mid \Pi_1^n\, \mathtt{down}.\,\mathbf{0}$ has to answer is similar.

Example algebraic characterisation: the axiom system $\mathcal{SB}$

Summation	**S1**		$P + 0$	$=$	P
	S2		$P + Q$	$=$	$Q + P$
	S3		$P + (Q + R)$	$=$	$(P + Q) + R$
	S4		$P + P$	$=$	P
Restriction	**R1**		$(\nu a)\, 0$	$=$	0
	R2	if $\mu \in \{a, \overline{a}\}$	$(\nu a)\, \mu. P$	$=$	0
	R3	if $\mu \notin \{a, \overline{a}\}$	$(\nu a)\, \mu. P$	$=$	$\mu. (\nu a)\, P$
	R4		$(\nu a)\, (P + Q)$	$=$	$(\nu a)\, P + (\nu a)\, Q$
Expansion	**E**		<laws for parallel composition>		

Theorem For P, Q finite CCS processes, $P \sim Q$ iff $\mathcal{SB} \vdash P = Q$.

Some key features of CCS (... and its family)

1. Minimality (set of operators, taxonomy)
2. Specificity of each operator
3. The algebra of the operators
4. Practical interest

A good example for 1-2: in CCS there is no construct $P; Q$ for sequential composition of processes

Sequential composition is a special special form of interaction, which can be modeled using the basic mechanisms of interaction in the calculus.

For instance, take $P \overset{\text{def}}{=} a.\, b + c$, and suppose d does not appear in P and Q. Then

$$P; Q \overset{\text{def}}{=} (\nu d)\, ((a.\, b.\, d + c.\, d) \mid \overline{d}.\, Q).$$

A lot has been learnt with CCS (and similar calculi such as CSP), especially in the 80s:

- behavioural equivalences
- algebras for concurrency
- logics for expressising behavioural properties
- techniques and algorithms for reasoning
- tools to assist mechanically in the specification and verification
- design of concurrent languages

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: $(\lambda x.\, x)3$ and 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 $\xrightarrow{\tau}$.

(ii) $\overset{\mu}{\Longrightarrow}$ as $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$ (relational composition).

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

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

1. if $P \xRightarrow{\mu} P'$, then there is Q' s.t. $Q \xRightarrow{\widehat{\mu}} Q'$ 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 $K \stackrel{\circ}{=} \tau.\,(a \mid K)$; 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 $P \xrightarrow{\mu} P'$, 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.

Exercises

Exercise Prove these equivalences (all cases can be done purely algebraically or using the bisimulation technique):

$$\begin{array}{rcll} (\tau.\, P) \mid Q & \approx & \tau.\,(P \mid Q) & \quad \text{for all } P, Q \\ (\nu a)\,(\tau.\, \overline{b} \mid c) & \approx & \overline{b} \mid c & \\ (\nu a)\,(b.\, \overline{a} \mid a.\, c) & \approx & b.\, c & \end{array}$$

Exercise Explain why $\tau.\,(\tau.\, a + \ b) + \tau.\, b \not\approx \tau.\, a + \tau.\, b$

Exercise Is it true that $K \approx a$, where $K \stackrel{\circ}{=} \tau.\, K + a$?

Exercise Let $K_1 \stackrel{\circ}{=} a.\,(b.\, K_1 \mid c)$, $K_1' \stackrel{\circ}{=} a.\,(b.\, c.\, K_1' + c.\, b.\, K_1')$, $K_2 \stackrel{\circ}{=} \overline{a}.\, K_2$, $H_1 \stackrel{\circ}{=} a.\, b.\, H_1$, $H_2 \stackrel{\circ}{=} \overline{a}.\, c.\, H_2$, $H \stackrel{\circ}{=} a.\, b.\, H + b.\, a.\, H$. Explain why $(\nu a)\,(K_1 \mid K_2) \not\approx (\nu a)\,(H_1 \mid H_2)$. Prove that $(\nu a)\,(K_1' \mid K_2) \approx H \approx (\nu a)\,(H_1 \mid H_2)$. (Hint: for $\approx$, you may find Exercises **??** useful; then either use algebraic laws and unique solutions of equations (page **??**), or define an appropriate weak bisimulation.)

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 $P \xrightarrow{\mu} P'$, then there is Q' s.t. $Q \overset{\widehat{\mu}}{\Longrightarrow} Q'$ 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 $P \overset{\mu}{\Longrightarrow} P'$, then there is Q' s.t. $Q \overset{\widehat{\mu}}{\Longrightarrow} Q'$ 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 $\approx$:

$$\begin{tikzcd}
\tau.\,a.\,0 \arrow[d] & \mathcal{R} & 0 \arrow[d, Rightarrow] \\
a.\,0 & & 0 \\
\approx & & \approx \\
\tau.\,a.\,0 & \mathcal{R} & 0
\end{tikzcd}$$

Exercises

Exercise Let $K_1 \doteq f.\,a.\,\overline{d}.\,K_1$, $K_2 \doteq d.\,b.\,\overline{e}.\,K_2$, $K_3 \doteq \overline{f}.\,e.\,c.\,K_3$, $H \doteq a.\,b.\,c.\,H$. Prove that $((\nu d, e, f)\,)(K_1 \mid K_2 \mid K_3) \approx H$.

Exercise Let $H_1 \doteq a.\,\overline{c_1}.\,e_1.\,d$, $H_2 \doteq b.\,\overline{c_2}$, $\mathrm{Sync} \doteq c_1.\,c_2.\,\overline{e_1}$. Prove that
$((\nu c_1, c_2, e_1)\,)(H_1 \mid H_2 \mid \mathrm{Sync}) \approx a.\,b.\,d + b.\,a.\,d$.

Exercise Let $H_1 \doteq a.\,\overline{c_1}.\,e_1.\,H_1$, $H_2 \doteq b.\,\overline{c_2}.\,e_2.\,H_2$, $\mathrm{Sync} \doteq c_1.\,c_2.\,\overline{e_1}.\,\overline{e_2}.\,\mathrm{Sync}$. Prove that

$$((\nu c_1, c_2, e_1, e_2)\,)(H_1 \mid H_2 \mid \mathrm{Sync}) \approx H$$

for $H \doteq a.\,b.\,H + b.\,a.\,H$.

Note: process Sync is used to synchronise 2 processes. Similarly synchronisers for $n > 2$ processes can be defined.

Exercise We proved that strong bisimulation can also be defined on sequences of actions. Prove something analogous for weak bisimulation. (The crux is to find the right definition of weak bisimulation on sequences of actions).

Other equivalences

Concurrency theory: models of processes

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

Process calculi

- CCS [$\longrightarrow$ π-calculus $\longrightarrow$ 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: $P \overset{\text{def}}{=} a.(b.c.0 + b.d.0)$ 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

$$\begin{tikzcd}
 & & \bullet \arrow[r, "\boldsymbol{c}"] & \bullet \\
\bullet \arrow[r, "\boldsymbol{a}"] & \bullet \arrow[ur, "\mathrm{b}"] \arrow[dr, "\mathrm{b}"'] & & \\
 & & \bullet \arrow[r, "\boldsymbol{d}"'] & \bullet
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \bullet \arrow[r, "\mathrm{b}"] & \bullet \arrow[r, "\boldsymbol{c}"] & \bullet \\
\bullet \arrow[ur, "\boldsymbol{a}"] \arrow[dr, "\boldsymbol{a}"'] & & & \\
 & \bullet \arrow[r, "\mathrm{b}"'] & \bullet \arrow[r, "\boldsymbol{d}"'] & \bullet
\end{tikzcd}$$

A law valid for failure but not for bisimilarity:

$$\boldsymbol{a.\,(b.\,P + b.\,Q) = a.\,b.\,P + a.\,b.\,Q}$$

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 `Act`

Bisimulation in a testing scenario

Language for testing:

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

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

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:

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

Outcomes

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

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

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

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

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

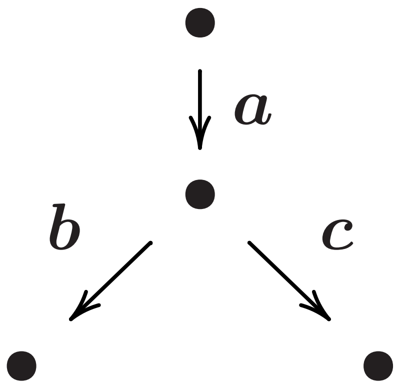
$$\mathcal{O}(T_1 \vee T_2, P) = \mathcal{O}(T_1, P) \vee^\star \mathcal{O}(T_1, P)$$

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

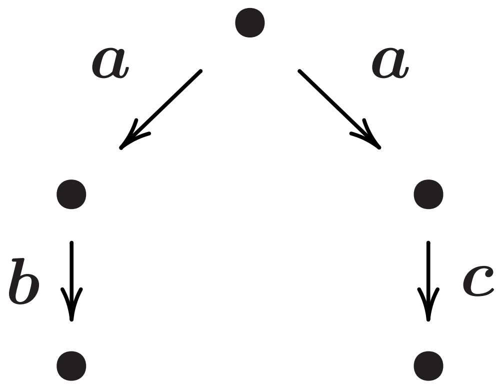
$$\mathcal{O}(\exists T, P) = \begin{cases} \{\top\} & \text{if } \top \in \mathcal{O}(T, P) \\ \{\bot\} & \text{otherwise} \end{cases}$$

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)

$$\begin{tikzcd}
& \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
\bullet \arrow[d, "b"'] & & \bullet \\
\bullet & &
\end{tikzcd}$$

P_3

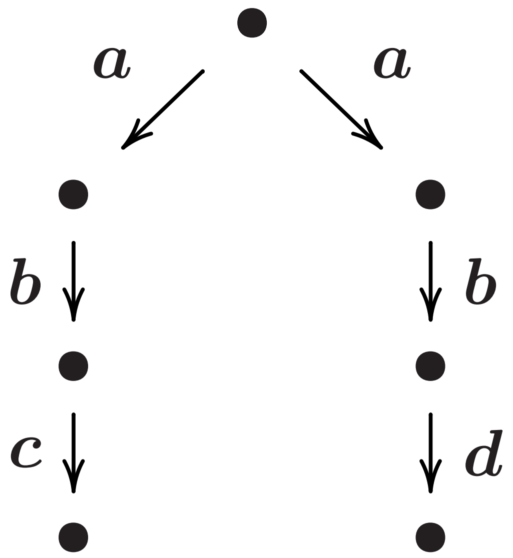
$$\begin{tikzcd}
\bullet \arrow[d, "a"] \\
\bullet \arrow[d, "b"] \\
\bullet
\end{tikzcd}$$

P_4

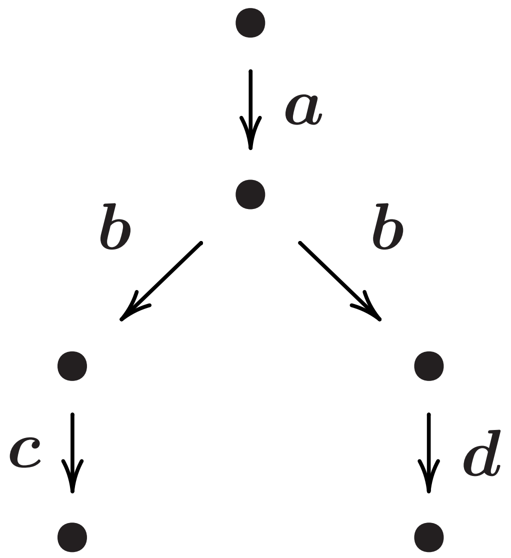
For $T_3 = a.\,b.\,\mathrm{SUCC}$, we have $\mathcal{O}(T_3, P_3) = \{\bot, \top\}$ and $\mathcal{O}(T_3, P_4) = \{\top\}$

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)

$$\begin{tikzcd}
& & & \bullet \arrow[dll, "a"'] \arrow[drr, "a"] & & & \\
& \bullet \arrow[dl, "b"'] \arrow[dr, "a"] & & & & \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
\bullet & & \bullet & & \bullet \arrow[d, "a"'] & & \bullet \\
& & & & \bullet & &
\end{tikzcd}$$

P_7

$$\begin{tikzcd}
& & & \bullet \arrow[dll, "a"'] \arrow[dr, "a"] & \\
& \bullet \arrow[dl, "b"'] \arrow[dr, "a"] & & & \bullet \arrow[d, "a"] \\
\bullet & & \bullet & & \bullet \arrow[d, "a"'] \\
& & & & \bullet
\end{tikzcd}$$

P_8

Exercise: Define tests that distinguish between P_7 and P_8.

Note: Every test has an inverse:

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

We have:

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

The equivalence induced by these tests:

$P \sim_{\mathrm{T}} Q \overset{\text{def}}{=}$ 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:

$$E ::= \langle T, P\rangle \;\Big|\; \top$$

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

$$\begin{tikzcd}\top \arrow[d, dash] \\ \bot\end{tikzcd}$$

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

$$\begin{tikzcd}[row sep=small]
\wp_{\mathtt{May}} & \wp_{\mathtt{Must}} & \wp_{\mathtt{Testing}} \\
\{\top\} = \{\top, \bot\} \arrow[d, dash] & \{\top\} \arrow[d, dash] & \{\top\} \arrow[d, dash] \\
\{\bot\} & \{\bot\} = \{\top, \bot\} & \{\top, \bot\} \arrow[d, dash] \\
 & & \{\bot\}
\end{tikzcd}$$

$\wp_{\mathtt{May}}$: 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**)

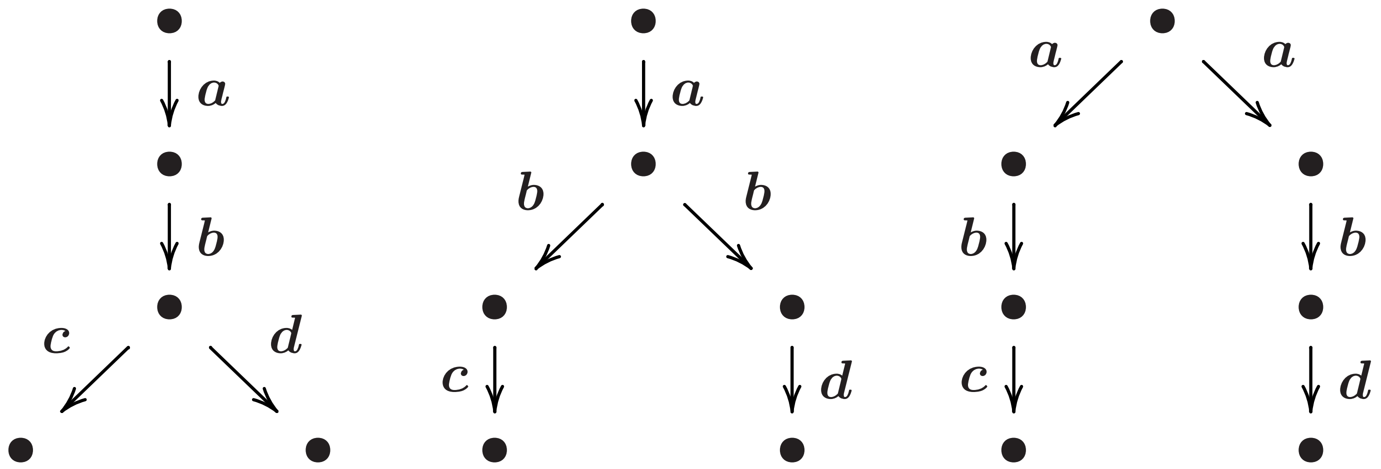
Note: $\simeq_{\mathtt{Testing}}$ is $\simeq$

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

$$
\begin{array}{ccccc}
P_9 & \simeq_{\mathtt{May}} & P_5 & \simeq_{\mathtt{May}} & P_6 \\
P_9 & \not\simeq_{\mathtt{Must}} & P_5 & \simeq_{\mathtt{Must}} & P_6 \\
P_9 & \not\simeq & P_5 & \simeq & P_6 \\
P_9 & \not\sim & P_5 & \not\sim & P_6
\end{array}
$$

$$\begin{tikzcd}
\bullet \arrow[r, "a"] \arrow[loop below, "\tau"] & \bullet & & \bullet \arrow[r, "a"] & \bullet
\end{tikzcd}$$

$$Q_1 \qquad\qquad\qquad Q_2$$

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

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

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**

Bits of history

3 lines, beginning mid 70s

- **Computer Science**
- **Philosophical logic** (modal logic)
- **Set theory**

Common basis: (weak) homomorphism between algebraic structures

Details:

Davide Sangiorgi, On the origins of bisimulation and coinduction,
ACM Trans. Program. Lang. Syst., 31(4), 2009.

The origins of bisimulation

and coinduction in Mathematics

Overview

- Foundations of theories for non-well-founded sets
- problem: what is equality for such sets?
  (non-trivial question, as these sets may have an infinite depth)
- bisimulation derived from isomorphism (and homomorphism)
  goal: obtaining relations coarser than isomorphism but still relating sets with “the same” internal structure
- main names: M. Forti, F. Honsell, P. Aczel, J. Barwise

Non-well-founded sets

The **axiom of foundation**: the membership relation on sets does not give rise to infinite descending sequences

$$\dots A_n \in A_{n-1} \in \dots \in A_1 \in A_0 \; .$$

Non-well-founded sets may violate it

Example (circular set) Ω that satisfies $\Omega = \{\Omega\}$

A set can also be non-well-founded without being circular

Equality on sets

- Equality on well-founded sets (Zermelo's **extensionality axiom**): two sets are equal if they have exactly the same elements.
- induction to reason on equality
  eg. to prove that the relation of equality is unique.
- On non-well-founded sets inductive arguments may not be applicable
  Example: the sets A and B s.t. $A = \{B\}$ and $B = \{A\}$
  If we apply the extensionality axiom we end up with a tautology
  ("A and B are equal iff A and B are equal")

Set Theory, first half 20th century

- The first axiomatisation of set theory by Ernst Zermelo in 1908
  - seven axioms, among which extensionality
  - no axioms of foundation (the possibility of circular is left open)
- In the same years, Bertrand Russell strongly rejects all definitions that involve forms of circularity
- Russel's **theory of types** ("stratification") **[1903,1908,1913]**
  - followed by the main logicians of the first half of the 20th century
  (Ernst Zermelo, Abraham Fraenkel, Thoralf Skolem, Johann von Neumann, Kurt Gödel, Paul Bernays)
- fear of paradoxes (eg, Burali-Forti's, Russell's)
- **common sense and perception**
- axiom of foundation deemed necessary for a "canonical" universe

Appearance of non-well-founded sets

Dimitry Mirimanoff [1917]: distinguish between well-founded and non-well-founded sets (the ‘ensembles *ordinaires* et *extraordinaires*’)

(we will come back to this later)

Paul Finsler [1926]: first attempt to an extensionality more powerful than Zermelo’s

(NB: use of graph theory)

Finsler and Mirimanoff work: remarkable but completely isolated and little known, until at least the 1960s (Specker, Scott, Boffa)

Main developments

Marco Forti and Furio Honsel [1980-83]

- work spurred by Ennio De Giorgi
- a number of anti-foundation axioms,
  some already appeared,
  a new one, X_1, that gives the strongest extensionality properties

Peter Aczel [1980s and later]

- re-discovers Forti and Honsell's anti-foundation axiom X_1, called AFA
- use of graph theory
- a book [1988] that makes bisimulation and non-well-founded sets popular in Mathematics
- motivations: mathematical foundations of processes, prompted by Milner's work on CCS and bisimulation

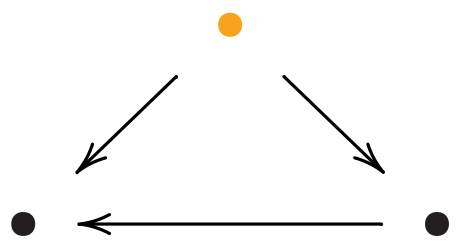
Sets via graphs

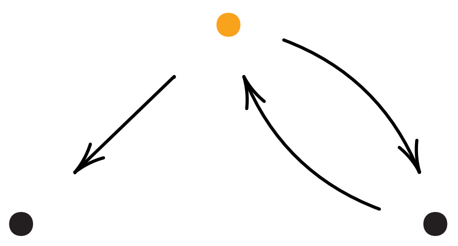
sets as (pointed) graphs where

- the nodes represent sets,
- the edges represent the converse membership relation (e.g., $x \to y$ if the set y is a member of x)
- the root of the graph indicates the node that represents the set under consideration.

Examples (roots in orange):

graphs





sets $\{\emptyset, \{\emptyset\}\}$ $\quad$ $D = \{\emptyset, \{D\}\}$

The graphs for the well-founded sets: no infinite paths or cycles

Decorations

A **decoration**: an assignment of sets to nodes that respects the structure of the edges

Examples

$$\begin{tikzcd}
& \{\emptyset, \{\emptyset\}\} \arrow[dl] \arrow[dr] & \\
\emptyset & & \{\emptyset\} \arrow[ll]
\end{tikzcd}$$

$$\begin{tikzcd}
& \{\emptyset, \{\mathrm{D}\}\} \arrow[dl] \arrow[dr, bend left] & \\
\emptyset & & \{D\} \arrow[ul, bend left]
\end{tikzcd}$$

[recall $D = \{\emptyset, \{D\}\}$]

The AFA axiom

AFA: every graph has a unique decoration

- **existence of the decoration**

  it tells us that the non-well-founded sets we need do exist.

- **uniqueness of the decoration**

  it tell us what is equality

Example: equality for the sets $\Omega = \{\Omega\}$, $A = \{B\}$, $B = \{A\}$.

$$\begin{tikzcd} \Omega \arrow[r, bend left] & \Omega \arrow[l, bend left] \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, bend left] & B \arrow[l, bend left] \end{tikzcd}$$

Bisimulation on sets

Bisimulation comes out when one tries to extract the meaning of equality.

A bisimulation relates sets A and B such that

– for all $A_1 \in A$ there is $B_1 \in B$ with A_1 and B_1 related; and the converse, for the elements of B_1.

Two sets are equal precisely if there is a bisimulation relating them.

The bisimulation proof method can then be used to prove equalities between sets (eg, the sets A and B in the previous slide).

Exercise AFA postulates that graphs have unique decoration. Show that the converse is false, i.e., there are (non-well-founded) sets that decorate different graphs.

Further developments

Aczel formulates AFA towards end 1983

he does not publish it immediately having then discovered the earlier work of Forti and Honsell and the equivalence between AFA and $\boldsymbol{X}_1$

he goes on developing the theory of non-well-founded sets, which leads to his ’88 book

and he develops the coalgebraic approach to semantics (Final Semantics)

with Aczel non-well-founded sets and bisimulation become popular in Mathematics for two main reasons:

– the elegant theory that he develops

– the concrete motivations for studying non-well-founded sets that he brings up.

Jon Barwise

Jon Barwise brings up other applications for non-well-founded sets

the study of paradoxes such as the Liar paradox in Philosophical Logic

more broadly the meaning in natural (i.e., human spoken) languages

Barwise develops a theory of non-well-founded sets based on systems of equations.

The axiom AFA becomes a requirement that appropriate systems of equations have a unique solution.

Intuition: Ω can be seen as the solution to the equation $x = \{x\}$, so all non-well-founded sets arise from systems of equations with variables on the left-hand side, and well-founded sets possibly containing such variables on the right-hand side.

(In Aczel '88 this property was expressed as the Solution Lemma; in Barwise it is the base assumption)

Barwise makes it the base assumption from which all the theory of sets is derived.

Historical curiosities

Other bisimulation-like notions

Earlier, or at the same time as Forti and Honsell, bisimulation-like relations are used to obtain extensional quotient models:

- **Roland Hinnion** [80,81]
- Harvey Friedman [73] and Lev Gordeev [82]
- **Jon Barwise, Robin O. Gandy, and Yiannis N. Moschovakis** [71]
  (cf: admissible sets; see also Moschovakis's book [74])

However these works do not isolate or study the concept of bisimulation (let alone bisimilarity)

Mirimanoff’s isomorphism

– Dimitry Mirimanoff [1917] (“ensembles extraordinaires”)
  Isomorphism between two non-well-founded sets E and E':

  A perfect correspondence can be established between the elements of E and E', in such a way that:

  1. all atoms $e \in E$ corresponds to an atom $e \in E'$ and conversely;
  2. all sets $F \in E$ corresponds to a set $F' \in E'$ so that the perfect correspondence can also be established on F and F' (ie, all atoms in F corresponds to an atom in F', and so forth)

For Mirimanoff: **isomorphism is not equality**
(cf: Zermelo's extensionality axiom)
Hence **isomorphism remains different from bisimilarity**

Example:

$A = \{B\}$ and $B = \{A\}$ isomorphic, not equal
$\{A, B\}$ not isomorphic to $\{A\}$ or $\{B\}$

Had one investigated the impact of isomorphism on extensionality, bisimulation and bisimilarity would have been discovered

Surprisingly, We have to wait 65 years for that

Why bisimulation discovered so late?

(in Math and elsewhere)

- Dangers of circularity and paradoxes (like Burali-Forti’s and Russel’s)
- Russel’s stratified approach
- Common sense
- Lack of concrete motivations
  (eg: Aczel’s work triggered by Milner’s on processes)

NB: Russel’s stratified approach has of course inspired type theories, including type theory in Computer Science, (Church, Scott, and Martin-Löf)

first disputed by Jean-Yves Girard and John Reynolds, in the 1970s, with impredicative polymorphism

Sets, functions, and the λ-calculus

– in the early 1900s a lot of of work in logic devoted to the building of systems meant to be the foundations for the whole of Mathematics

– some based on concepts of sets, others on concepts of functions

– on the line on functions, a very significant outcome for Computer Science (and logics) is the λ-calculus by Alonzo Church [1930s] (also Combinatory Logics, Ilyich Schonfinkel [1920s])

– Church's goal: a foundation for logic which would be more natural than Russell's type theory or Zermelo's set theory, and based on functions

– at the end mathematicians preferred (axiomatic) set theory.

– Church retreated to the less ambitious goal of re-formulating simple type theory on a λ-calculus base

- Church used the λ-calculus to provide the first solution to Hilbert’s Entscheidungsproblem

  (’is there a decision method to solve all problems in first order logic?’)

  - independently and shortly afterwards, same result by Alan Turing, who invented Turing machines for this
  - Church and Turing needed a formalism to reason on the intuitive notion of ’decidable’ (ie., ’computable’)

- λ-calculus was re-discovered for Computer Science by people like McCarthy, Strachey, Landin, and Scott in the 1960s.

  - imperative programming languages follow the λ-calculus paradigm
  - imperative programming languages follow the Turing machines paradigm