Coinduction in Programming Languages

Davide Sangiorgi

University of Bologna

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

Theme of the talk

- **what is coinduction?**
- **what has been achieved?**

Equality on processes, coinductively

Bisimulation:

A relation $\mathcal{R}$ s.t.

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

Bisimilarity ($\sim$) :

$\bigcup \{\mathcal{R} \; : \; \mathcal{R}$ is a bisimulation $\}$

(coind. definition)

Hence:

$$\frac{x \; \mathcal{R} \; y \qquad \mathcal{R} \text{ is a bisimulation}}{x \sim y}$$

(coind. proof principle)

Equality on processes, coinductively

Bisimulation:

A relation $\mathcal{R}$ s.t.

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

– local, unordered
– infinity in space

Bisimilarity ($\sim$) :

$$\bigcup \; \{\mathcal{R} \; : \; \mathcal{R} \text{ is a bisimulation } \}$$

(coind. definition)

Hence:

$$\frac{x \; \mathcal{R} \; y \qquad \mathcal{R} \text{ is a bisimulation}}{x \sim y}$$

(coind. proof principle)

Equality on processes, coinductively

Bisimulation:

A relation $\mathcal{R}$ s.t.
$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \alpha\downarrow & & \downarrow\alpha \\ P' & \mathcal{R} & Q' \end{array}$$

– local, unordered
– infinity in space

Bisimilarity ($\sim$) :

$\bigcup \ \{\mathcal{R} \ : \ \mathcal{R}$ is a bisimulation $\}$

includes $\sim$ (impredicative)

(coind. definition)

Hence:

$$\frac{x \ \mathcal{R} \ y \qquad \mathcal{R} \text{ is a bisimulation}}{x \sim y}$$

(coind. proof principle)

Coincidence

Locality $+$ **no order** :

– “simple” proofs

– avoid the dangers of circularity

cf: paradoxes, like Russel’s

– effective on infinite objects

direct, natural modelling

Another examples of coinduction: divergence

A Rewriting System

D-invariant: a predicate $\mathcal{P}$ s.t.

$$\frac{M \in \mathcal{P} \qquad M \longrightarrow M'}{M' \in \mathcal{P}}$$

Divergence ($\Uparrow$) $\triangleq \bigcup \{\mathcal{P} \;:\; \mathcal{P}$ is a D-invariant $\}$

(coind. definition)

Hence:

$$\frac{M \in \mathcal{P} \qquad \mathcal{P} \text{ is a D-invariant}}{M \Uparrow}$$

(coind. proof principle)

Tarski

On a **complete lattice** (i.e., a partial order $<$ with all joins)

Theorem On a complete lattice, a monotone endofunction $\boldsymbol{F}$ has a complete lattice of post-fixed points

$\boldsymbol{x}$ is **post-fixed point** of $\boldsymbol{F}$ if $\boldsymbol{x} < \boldsymbol{F}(\boldsymbol{x})$

Corollary **[coinduction proof principle, à la Tarski]**

$$\frac{\boldsymbol{F} \text{ monotone} \qquad \boldsymbol{x} < \boldsymbol{F}(\boldsymbol{x})}{\boldsymbol{x} < \operatorname{gfp}(\boldsymbol{F})}$$

CONTENTS

Bisimulation

– One of the most important contributions of concurrency theory to CS

– It has spurred the study of coinduction

– In concurrency: the most studied equivalence

  * ... in a plethora of equivalences **(see van Glabbeek 93)**
  * Why?

Bisimulation in concurrency

– **Clear** meaning of equality

– **Natural**

– The **finest** extensional equality

Extensional: – "whenever it does an output at b it will also do an input at a"

Non-extensional: – "Has 8 states"

– "Has an Hamiltonian circuit"

– An associated powerful **proof technique**

– **Robust**

Characterisations: logical, algebraic, set-theoretical, categorical, game-theoretical, ...

– Several **separation results** from other equivalences

Basic Process Algebra (BPA)

(cf: context-free grammars)

$$P ::= a \;\Big|\; P_1 \cdot P_2 \;\Big|\; P_1 + P_2 \;\Big|\; X$$

where $X \triangleq P$

Bisimilarity: **decidable** (with norm, in polynomial time!)

[Hirshfeld, Jerrum, Moller]

All equivalences "below" bisimilarity (trace, testing, etc.):
undecidable

[Bar-Hillel, Perles, Shamir; Friedman; Groote; Huttel]

Basic Parallel Processes (BPP)

$$P ::= a \;\Big|\; P_1 \mid P_2 \;\Big|\; P_1 + P_2 \;\Big|\; X$$

where $X \triangleq P$

The picture is the same as for BPA

Proofs very unrelated (cf: model checking is decidable for BPA and undecidable for BPP, Esparza)

Finite-state processes

Bisimulation: P-complete

[Alvarez, Balcazar, Gabarro, Santha]

With m transitions, n states:

O(m log n) time and O(m + n) space [Paige, Tarjan]

Trace equivalence, testing: PSPACE-complete

[Kannelakis, Smolka; Huynh, Tian]

More generally:

- **“local” equivalences: in P**
- **“non-local” equivalences: PSPACE-complete**

Unique decomposition

(Natural numbers, $*$, 1)

p **prime** if $p \neq 1$ and $p \neq q * r$ for $Q \wedge R \neq 1$

(Processes, $|$, 0, and an equivalence $=$)

P **prime** if $P \neq 0$ and $P \neq Q \mid R$ where $Q \wedge R \neq 0$

Examples: a

$b.\, a$

Non-examples (for $\sim$): $a.\, a$ (since $\sim a \mid a$)

$a.\, b + b.\, a$ (since $\sim a \mid b$)

Theorem Any process can be expressed as the parallel composition of primes up to bisimilarity [Moller, ...]

Valid for: finite, live finite-state, normed BPA, normed BPP, etc.

Not possible for other equivalences

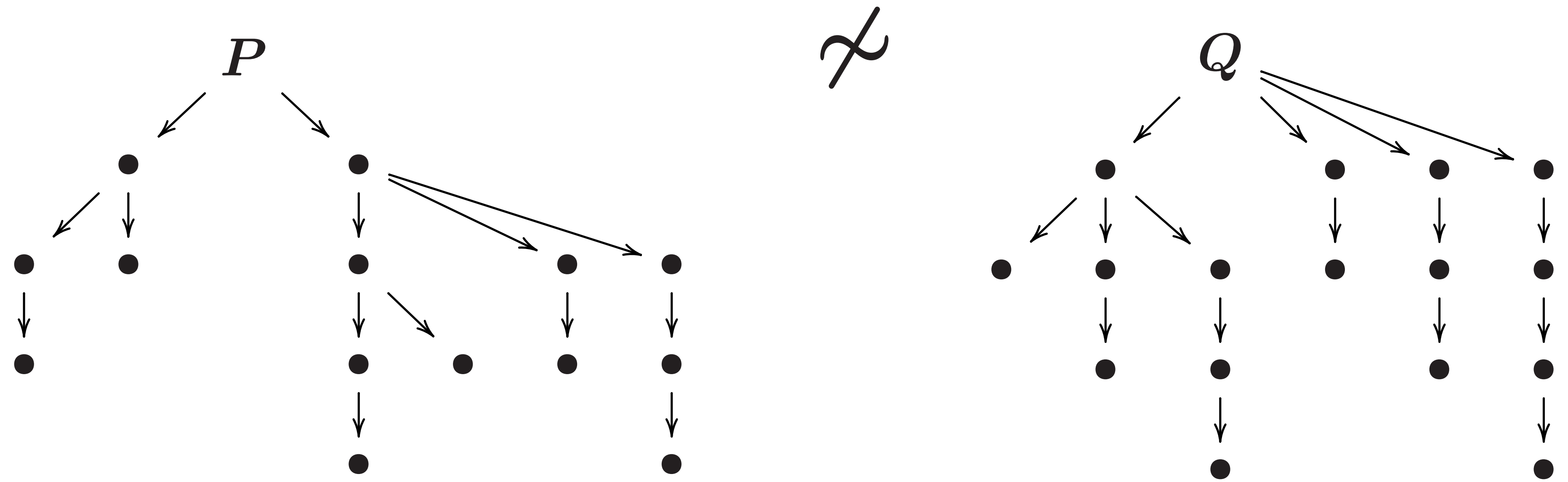
$$a^n \triangleq \overbrace{a.\cdots.a}^{n}$$

Counterexample

$$P \triangleq (a + a^2) \mid (a + a^2)$$
$$Q \triangleq a \mid (a + a^2 + a^3)$$

Their traces: a^2, a^3, a^4

Their trees:



Bisimulation in concurrency, today

- To **define equality** on processes (fundamental!!)
- To **prove equalities**
  - ∗ even if bisimilarity is not the chosen equivalence
    - · trying bisimilarity first
    - · coinductive characterisations of the chosen equivalence
- To **justify algebraic laws**
- To **minimise** the state space
- To **abstract** from certain details

Coinduction in programming languages

– **Bisimilarity in functional languages and OO languages**

[Abramsky, Ong]

A major factor in the movement towards operationally-based techniques in PL semantics in the 90s

– **Program analysis** **(see Nielson, Nielson, Hankin 's book)**

Noninterference (security) properties

– **Verification tools**: algorithms for computing gfp (for modal and temporal logics), tactics and heuristics

- **Types** [Tofte]
  - type soundness
  - coinductive types and definition by corecursion

    Infinite proofs in Coq [Coquand, Gimenez]
  - recursive types (equality, subtyping, ...)

    A coinductive rule:

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

- **Recursively defined data types and domains** [Fiore, Pitts]
- **Databases** [Buneman]

CONTENTS

Two concepts here to be tracked down:

– **bisimulation**

– **coinductive principles**

(only here the issues of circularity arise)

$$
\begin{tikzcd}
P \arrow[d, "\alpha"'] & \sim & Q \arrow[d, "\alpha"] \\
P' & \sim & Q'
\end{tikzcd}
\qquad\qquad
\frac{\Gamma, \langle p_1, q_1 \rangle \sim \langle p_2, q_2 \rangle \vdash p_i \sim q_i}{\Gamma \vdash \langle p_1, q_1 \rangle \sim \langle p_2, q_2 \rangle}
$$

3 lines, beginning early 70s

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

Common basis:

(weak) homomorphism between algebraic structures

From homomorphism to bisimulation in Philosophical logic

Modal logics interpreted on Kripke structures

Homomorphism (on models with a single relation $\rightarrow$) **:**

a function $\mathcal{F}$ s.t.

$$\begin{tikzcd} s \arrow[r, mapsto, "\mathcal{F}(s)"] \arrow[d] \arrow[dr, phantom, "\Nearrow"] & t \arrow[d] \\ s' \arrow[r, mapsto, "\mathcal{F}(s')"'] & t' \end{tikzcd}$$

Differences with bisimulation

– **functional**

– **no back condition**

When is the truth of a formula preserved when the model changes?

Theorem Modal formulas are not invariant under homomorphism

– **require an "iff" in the clause** (strong homomorphism)

– **require a back condition** (p-morphism)

[Jongt and Troelstra 1966, Segerberg 1971]

$$\begin{tikzcd}
s \arrow[r, mapsto, "\mathcal{F}(s)"] \arrow[d] \arrow[dr, phantom, "\Updownarrow"] & t \arrow[d] \\
s' \arrow[r, mapsto, "\mathcal{F}(s')"'] & t'
\end{tikzcd}$$

Theorem Modal formulas are invariant under p-morphism

A better invariance: **p-relations** (bisimulations)

[Van Benthem, 76]

Theorem **(Van Benthem)** A first-order logic formula (over Kripke structures) is equivalent to a modal formula iff it is bisimulation invariant

cf: logical characterisation of bisimilarity **[Hennessy-Milner, 85]**

No coinduction, really

Computer Science

- **Homomorphism** (well-known in the 60s)
- **Milner, '71**: Simulation between sequential imperative programs
- **Park, '81**: "Concurrency and Automata on Infinite Sequences"
  * Bisimulation as a proof technique for language equivalence on (mild variants of) Buchi automata
  * A celebrated paper
  * Still, no coinduction
- **Coinduction:** Park, while staying at Milner's
- Immediately adopted by Milner for **CCS**

Set-theory

Foundations of set theory (cf: non-well-founded sets)

- **Forti, Honsell** ’80-83, **Hinnion** ’80-81

  Bisimulations: f-conservative relations, contractions

  Coinduction?

  * yes
  * a little hidden (more attention to bisimulation equivalences than bisimulations)

- **Aczel** ’85-89

  nwf sets popular, motivated by Milner’s work on CCS

  the basis of the coalgebraic approach to semantics

Under its most general connotation (as from Tarski) coinduction in CS existed even earlier

(but not recognised as such)

– Unification

  Structural equivalence of graphs **[Hopcroft and Ullman '71]**

– Invariant properties (60s, 70s)

  A huge literature

  Hoare logic: while-statements and weakest preconditions

  Connections to fixed-point theory

  [Clarke '77; also De Bakker, De Rover in the 70s]

CONTENTS

Final semantics

[Aczel, Rutten, Turi, Jacobs, Fiore, Plotkin, Lenisa, Honsell, ...]

- **Operationally-driven**
- The meaning of a program: an **F-coalgebra**

  (F: a functor in a category)
- The mapping onto the **final F-coalgebra**: the equivalence induced on terms
  - Canonical representatives
  - Universal domain of observations
- If F well-behaved: **coinductive** techniques
- Dual to the "**initial algebra**" approach" [ADJ group]
- **Insights** into the nature of coinduction and the duality with induction

Algebras

$$\begin{array}{c} F(S) \\ \Big\downarrow f \\ S \end{array}$$

Intuition:

object	S	$\leftrightarrow$	a set
arrow	f	$\leftrightarrow$	a function over sets
functor	F	$\leftrightarrow$	a signature

An algebra over S tells us how to construct new elements of S

Example:

$$F(S) = 1 + A \times S \qquad (A \text{ is a given set})$$

Then $f \triangleq \langle f_1, f_2 \rangle$

with $f_1 : 1 \mapsto S$

$f_2 : A \times S \mapsto S$

The initial algebra: (finite) lists over A

Coalgebras

$$\begin{array}{c} S \\ \Big\downarrow f \\ F(S) \end{array}$$

A coalgebra over S tells us how to decompose elements in S (cf: the observations)

Example

$$F(S) = A \times S$$

$$\begin{array}{ccccccc} f: & s & \to & \langle & \textbf{head}, & \textbf{tail} & \rangle \\ & \Cap & & & \Cap & \Cap & \\ & S & & & A & S & \end{array}$$

A coalgebra:

$$\begin{array}{c} S \\ \Big\downarrow f \\ A \times S \end{array}$$

- From any $s \in S$ we can extract an infinite sequence of elements in A
- $s_1 \neq s_2$ may give us the same sequences
- An element of S need not be infinite:

$$* \mapsto a \times * \qquad \begin{array}{c} * \\ \Big\downarrow \\ A \times * \end{array}$$

- **The final coalgebra** : streams over A

Corecursion

Example: $F(S) = A \times S$

$$\begin{aligned} A^+ &\triangleq \text{streams (final coalgebra)} \\ \text{const}(a) &\triangleq * \to a \times a \times a \ldots \end{aligned}$$

$$\begin{tikzcd}[column sep=huge, row sep=large]
* \arrow[r, "\text{const}(a)"] \arrow[d, "* \mapsto a \times *"'] & A^+ \arrow[d, "\langle \text{head,tail} \rangle"] \\
A \times * \arrow[r, "\text{id}\times\ \text{const}(a)"'] & A \times A^+
\end{tikzcd}$$

Bisimulation, coalgebraically

From homomorphim to bisimulation:

$$\begin{tikzcd} S \arrow[d, "\alpha"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\gamma"'] & T \arrow[d, "\beta"'] \\ F(S) & F(R) \arrow[l, "F(\pi_1)"] \arrow[r, "F(\pi_2)"'] & F(T) \end{tikzcd} \qquad R \subseteq S \times T$$

Bisimulation proof principle, coalgebraically

$x \ \mathrm{fin\_coalg}(F) \ y \ \triangleq$ equated via the mapping into the final F-coalgebra

The category has terminal object and limits (of ω-chains)

F is a functor (and "behaves well")

$$\frac{x \ \mathcal{R} \ y \qquad \mathcal{R} \text{ is an } F\text{-bisimulation}}{x \ \mathrm{fin\_coalg}(F) \ y}$$

Bisimulation proof principle, coalgebraically

$x \; \text{fin\_coalg}(F) \; y \;\triangleq$ equated via the mapping into the final F-coalgebra

The category has terminal object and limits (of ω-chains)

F is a functor (and "behaves well")

$$\frac{x \,\mathcal{R}\, y \qquad \mathcal{R} \text{ is an } F\text{-bisimulation}}{x \; \text{fin\_coalg}(F) \; y}$$

Compare it with the results à la Tarski :

A complete lattice

F monotone

$$\frac{x \,\mathcal{R}\, y \qquad \mathcal{R} \subseteq F(\mathcal{R})}{x \; \text{gfp}\,(F) \; y}$$

The duality: summary

constructors	destructors
inductive def	coinductive def
def by recursion	def by corecursion
induction technique	coinductive technique
congruence	bisimulation
inductive predicates	invariants
least fixed-points	greatest fixed-points
initiality	finality
sets	non-well-founded sets
strengthening inductive hypothesis	coinduction ‘up-to’

Canonical representatives

Final coalgebras: a domain of canonical representatives

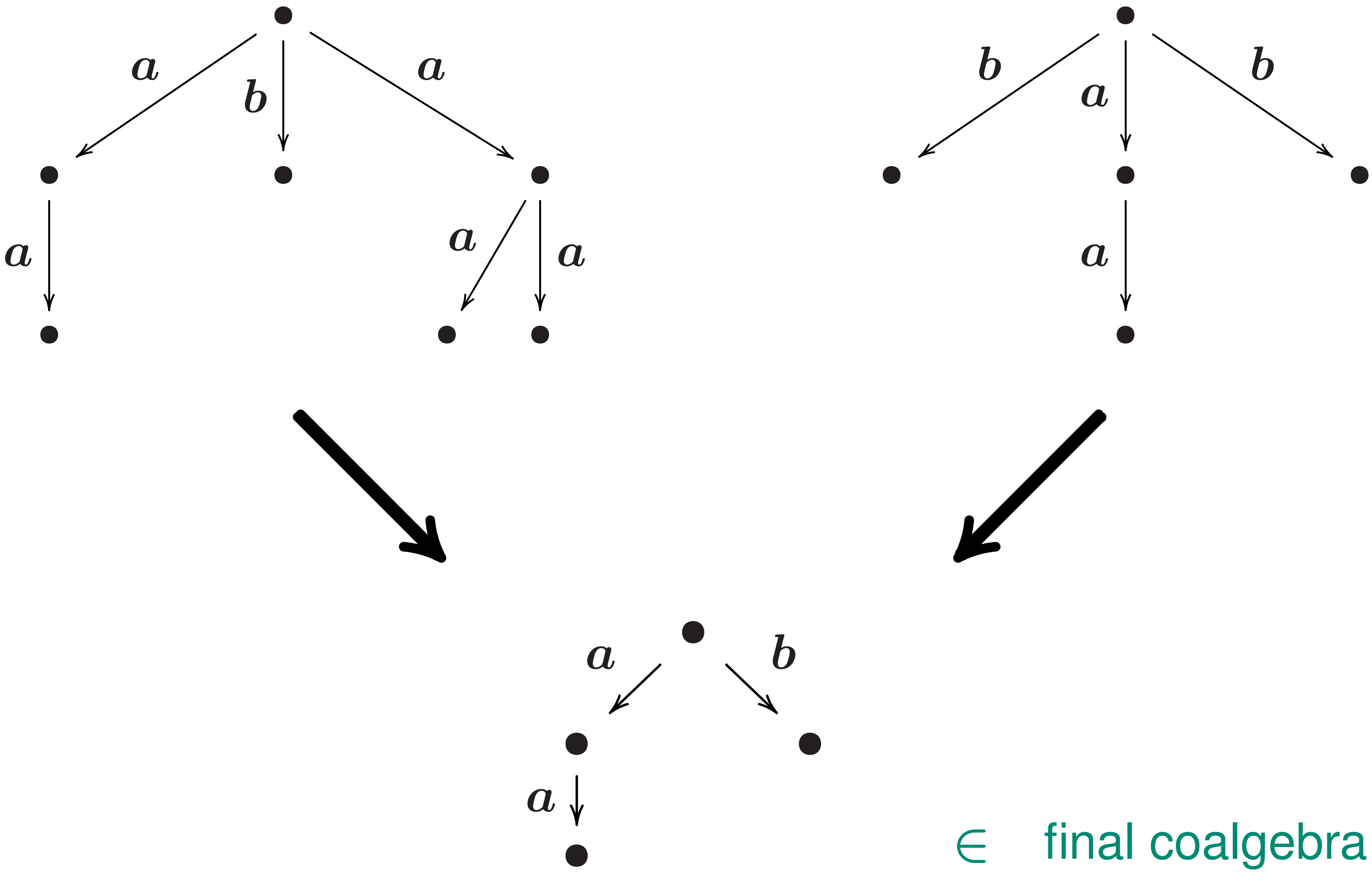
Example:

$$F(X) = \wp_{\mathrm{fin}}(A \times X)$$

Final coalgebra: the set of all finitely-branching trees, A-labeled, minimal wrt bisimilarity

Any finitely-branching tree, A-labeled tree can be made into a coalgebra

The mapping of a tree onto the final coalgebra picks up the canonical representative for the equivalence class of that tree


a
b
a
a
a
a
b
a
b
a
a
b
a
∈ final coalgebra

Coalgebras: main drawbacks

- Compositionality

  but see Plotkin and Turi

- General coinduction (ie, non-relational)

- What are observations ?

CONTENTS

Redundancies: example

The perfect firewall equation in Ambients

P: a process with n not free in it

$$\nu n \;\; n\langle P\rangle \sim \mathbf{0}$$

Proof: **Let's find a bisimulation...**

Is this a bisimulation?

$$\mathcal{R} \triangleq \{\ (\nu n\ n\langle P\rangle\ ,\ 0)\ \}$$

Is this a bisimulation?

$$\mathcal{R} \triangleq \{\ (\nu n\ n\langle P\rangle\ ,\ 0)\ \}$$

No!

$$\begin{array}{ccc}
\nu n\ n\langle P\rangle & \mathcal{R} & 0 \\
\Big\downarrow{\scriptstyle \mathtt{enter_}k\langle Q\rangle} & & \Big\downarrow{\scriptstyle \mathtt{enter_}k\langle Q\rangle} \\
k\langle Q \mid \nu n\ n\langle P\rangle\ \rangle & \cancel{\mathcal{R}} & k\langle Q\rangle \mid 0
\end{array}$$

Try again...

Is this a bisimulation?

$$\begin{aligned} \mathcal{R} \triangleq & \quad \{\, (\nu n\ n\langle P\rangle\ ,\ 0)\,\} \\ \textstyle\bigcup_{k,Q} & \quad \{\, (k\langle Q \mid \nu n\ n\langle P\rangle\,\rangle\ ,\ k\langle Q\rangle \mid 0)\,\} \end{aligned}$$

Is this a bisimulation?

$$\mathcal{R} \triangleq \{ (\nu n\ n\langle P\rangle\ ,\ 0) \}$$
$$\quad \cup_{k,Q}\ \{ (k\langle Q \mid \nu n\ n\langle P\rangle\ \rangle\ ,\ k\langle Q\rangle \mid 0) \}$$

No!

. . .

Try again...

The bisimulation:

$$\mathcal{R} \triangleq \bigcup_{C \text{ is a static contexts}} \{(S,T) : \begin{array}{lll} S & \sim & C[\nu n\ n\langle P\rangle\ ] \\ T & \sim & C[0] \end{array} \}$$

$$C ::= k\langle C\rangle \ \big|\ P \mid C \ \big|\ \nu a\, C \ \big|\ [\,]$$

We started with the **singleton** relation

$$\{(\nu n\ n\langle P\rangle\ ,\ 0)\}$$

The added pairs: **redundant**? (derivable, laws of $\sim$)

Can we work with relations smaller than bisimulations?

Advantage: fewer and simpler bisimulation diagrams

Redundant pairs

What we would like to do:

$\mathcal{R} \triangleq \mathcal{R}^* - \{$some redundant pairs$\}$

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \alpha\downarrow & & \downarrow\alpha \\ P' & \mathcal{R}^* & Q' \end{array}$$

implies $\mathcal{R} \subseteq \sim$

Redundant pairs

What we would like to do:

$\mathcal{R} \triangleq \mathcal{R}^* - \{\text{some redundant pairs}\}$

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \alpha\downarrow & & \downarrow\alpha \\ P' & \mathcal{R}^* & Q' \end{array} \quad \text{implies } \mathcal{R} \subseteq \sim$$

A wrong definition of redundant:

$\mathcal{S} \triangleq$ a set of inference rules valid for $\sim$

(P, Q) is redundant in $(P, Q) \cup \mathcal{R}$ if

$$\mathcal{S} \quad \frac{\mathcal{R} \subseteq \sim}{P \sim Q}$$

In some cases it works

– **Rules for transitivity of $\sim$ (up-to $\sim$)** [Milner]

$$\begin{array}{ccccccc} P & & & \mathcal{R} & & & Q \\ \alpha\downarrow & & & & & & \downarrow\alpha \\ P' & \sim & P'' & \mathcal{R} & Q'' & \sim & Q' \end{array}$$

implies $\mathcal{R} \subseteq \sim$

Warning: in some cases it does not work, even though $\sim$ is transitive

In some cases it works

- Rules for transitivity of $\sim$ (up-to $\sim$)
- **rules for substitutivity of $\sim$ (up-to context)**

[Sangiorgi]

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \alpha\downarrow & & \downarrow\alpha \\ \cancel{C}\,[P'] & \mathcal{R} & \cancel{C}\,[Q'] \end{array}$$ implies $\mathcal{R} \subseteq \sim$

Warning: in some cases it does not work, even though the contexts preserve $\sim$

In some cases it works

- Rules for transitivity of $\sim$ (up-to $\sim$)
- rules for substitutivity of $\sim$ (up-to context)
- **rules for invariance of $\sim$ under injective substitutions (up-to injective substitutions)**

$$\begin{tikzcd}
P \arrow[d, "\alpha"'] & \mathcal{R} & Q \arrow[d, "\alpha"] \\
P'\sigma & \mathcal{R} & Q'\sigma
\end{tikzcd}$$

σ: an injective function σ

implies $\mathcal{R} \subseteq \sim$

Proof of the firewall, composition up-to techniques

We can prove $\nu n \;\; n\langle P\rangle \;\; \sim \;\; \mathbf{0}$ using the singleton relation

$$\begin{array}{ccc}
\nu n \;\; n\langle P\rangle & \mathcal{R} & \mathbf{0} \\
\mathtt{enter_}k\langle Q\rangle \downarrow & & \downarrow \mathtt{enter_}k\langle Q\rangle \\
k\langle Q \mid \nu n \;\; n\langle P\rangle \;\rangle & & k\langle Q\rangle \mid \mathbf{0}
\end{array}$$

Proof of the firewall, composition up-to techniques

We can prove $\nu n \ n\langle P \rangle \ \sim \ \mathbf{0}$ using the singleton relation

$$
\begin{tikzcd}
\nu n \ n\langle P\rangle \arrow[d, "\mathtt{enter}\_k\langle Q\rangle"'] & \mathcal{R} & \mathbf{0} \arrow[d, "\mathtt{enter}\_k\langle Q\rangle"] \\
k\langle Q \mid \nu n \ n\langle P\rangle \ \rangle \arrow[d, phantom, "\sim"] & & k\langle Q\rangle \mid \mathbf{0} \arrow[d, phantom, "\sim"] \\
k\langle Q \mid \nu n \ n\langle P\rangle \ \rangle & & k\langle Q \mid \mathbf{0}\rangle
\end{tikzcd}
$$

Proof of the firewall, composition up-to techniques

We can prove $\nu n\ n\langle P\rangle \ \sim\ \mathbf{0}$ using the singleton relation

$$
\begin{array}{ccc}
\nu n\ n\langle P\rangle & \mathcal{R} & \mathbf{0} \\
\quad\downarrow{\scriptstyle \mathtt{enter\_}k\langle Q\rangle} & & \downarrow{\scriptstyle \mathtt{enter\_}k\langle Q\rangle} \\
k\langle Q \mid \nu n\ n\langle P\rangle\ \rangle & & k\langle Q\rangle \mid \mathbf{0} \\
\sim & & \sim \\
\cancel{k\langle Q} \mid \nu n\ n\langle P\rangle\ \rangle & \mathcal{R} & \cancel{k\langle Q} \mid \mathbf{0}\rangle
\end{array}
$$

[Merro, Zappa Nardelli '03]

"up-to $\sim$" **and** "up-to context" (NB: need also: "up-to injective substitutions", with a **different composition**)