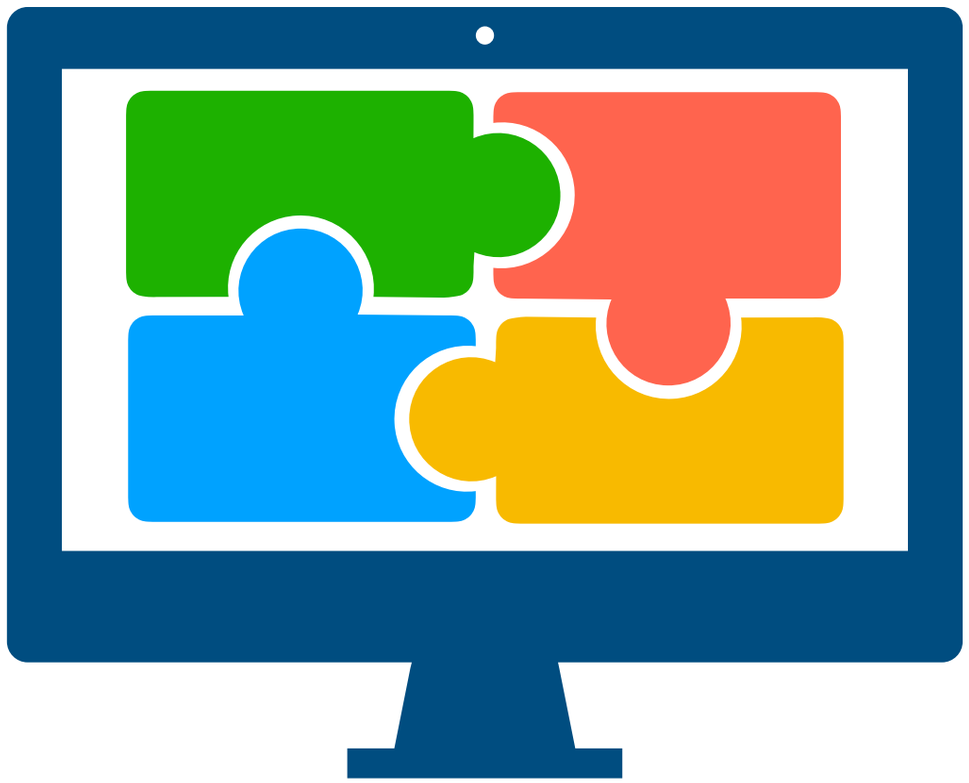


http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/

PSC 2020/21 (375AA, 9CFU)

Principles for Software Composition

Roberto Bruni
http://www.di.unipi.it/~bruni/

18a - CCS abstract semantics

CCS

Graph isomorphism

CCS syntax

p, q	$::=$	$\mathbf{nil}$	inactive process
	$\mid$	x	process variable (for recursion)
	$\mid$	$\mu.p$	action prefix
	$\mid$	$p\backslash\alpha$	restricted channel
	$\mid$	$p[\phi]$	channel relabelling
	$\mid$	$p + q$	nondeterministic choice (sum)
	$\mid$	$p \mid q$	parallel composition
	$\mid$	$\mathbf{rec}\ x.\ p$	recursion

(operators are listed in order of precedence)

CCS op. semantics

Act) $$\frac{}{\mu.p \xrightarrow{\mu} p}$$

Res) $$\frac{p \xrightarrow{\mu} q \quad \mu \notin \{\alpha, \overline{\alpha}\}}{p\backslash\alpha \xrightarrow{\mu} q\backslash\alpha}$$

Rel) $$\frac{p \xrightarrow{\mu} q}{p[\phi] \xrightarrow{\phi(\mu)} q[\phi]}$$

SumL) $$\frac{p_1 \xrightarrow{\mu} q}{p_1 + p_2 \xrightarrow{\mu} q}$$

SumR) $$\frac{p_2 \xrightarrow{\mu} q}{p_1 + p_2 \xrightarrow{\mu} q}$$

ParL) $$\frac{p_1 \xrightarrow{\mu} q_1}{p_1|p_2 \xrightarrow{\mu} q_1|p_2}$$

Com) $$\frac{p_1 \xrightarrow{\lambda} q_1 \quad p_2 \xrightarrow{\overline{\lambda}} q_2}{p_1|p_2 \xrightarrow{\tau} q_1|q_2}$$

ParR) $$\frac{p_2 \xrightarrow{\mu} q_2}{p_1|p_2 \xrightarrow{\mu} p_1|q_2}$$

Rec) $$\frac{p\left[{}^{\mathbf{rec}\ x.\ p}/_{x}\right] \xrightarrow{\mu} q}{\mathbf{rec}\ x.\ p \xrightarrow{\mu} q}$$

Isomorphic LTS

syntactically different processes
can exhibit exactly the same behaviour

$\mathbf{nil}$ $\quad$ $\mathbf{nil}\backslash\alpha$ $\quad$ $\mathbf{nil}[\phi]$

p $\quad$ $p + \mathbf{nil}$ $\quad$ $p + p$ $\quad$ $p|\mathbf{nil}$

$p + q$ $\quad$ $q + p$

$p|q$ $\quad$ $q|p$

their LTS are different (states syntactically different)

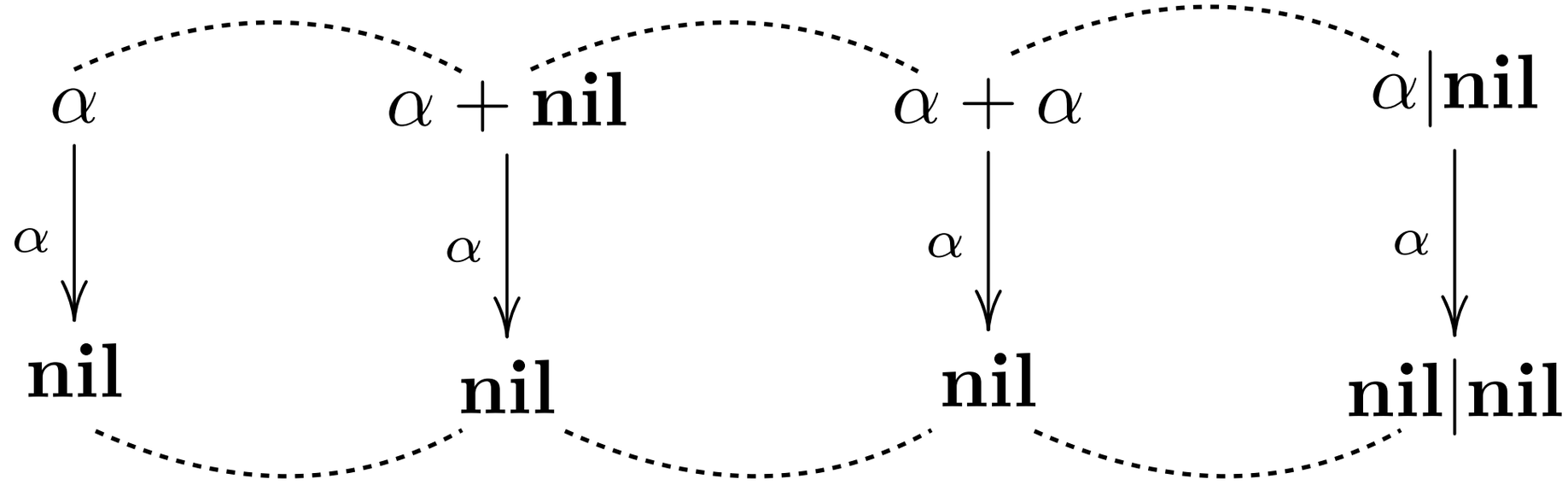
graph isomorphism abstracts away from states

Graph isomorphism

$$G = (V, E) \qquad G' = (V', E')$$

$$v \xrightarrow{\mu} w \quad \Leftrightarrow \quad f(v) \xrightarrow{\mu} f(w)$$

$f : V \to V'$ bijective

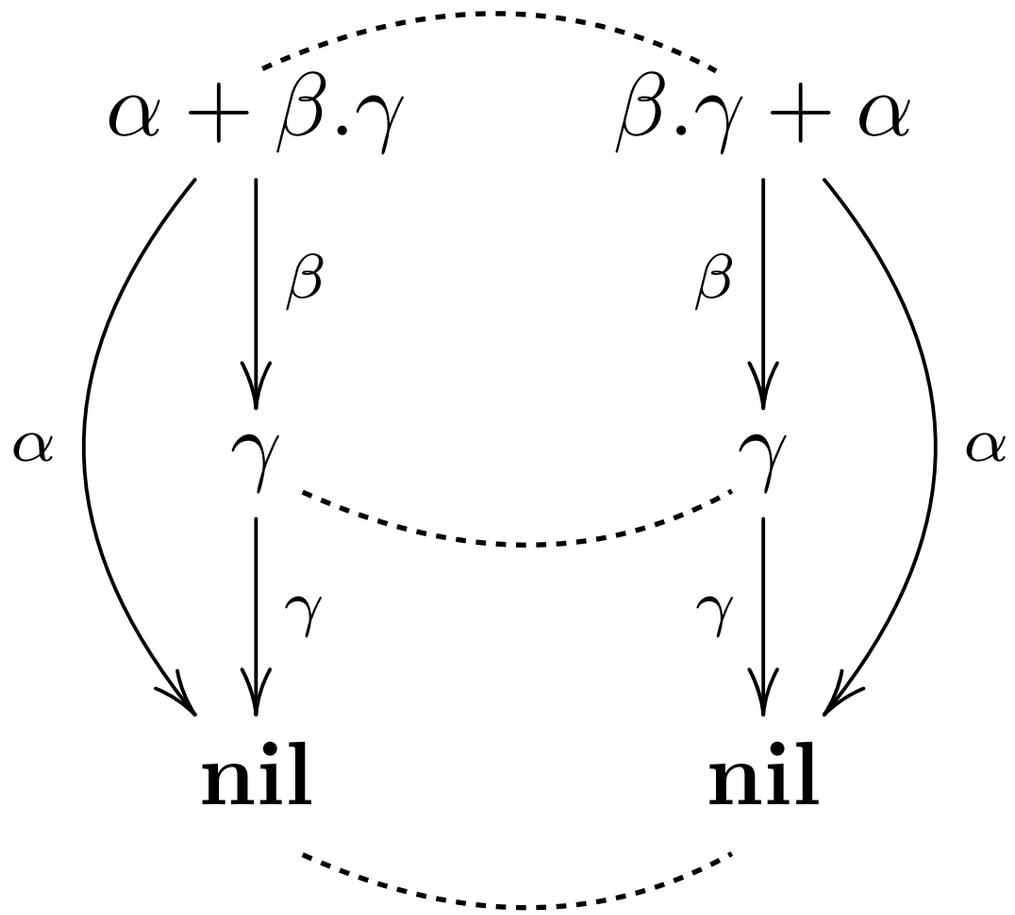


Graph isomorphism

$$G = (V, E) \qquad G' = (V', E')$$

$$v \xrightarrow{\mu} w \quad \Leftrightarrow \quad f(v) \xrightarrow{\mu} f(w)$$

$$f : V \to V' \quad \text{bijective}$$

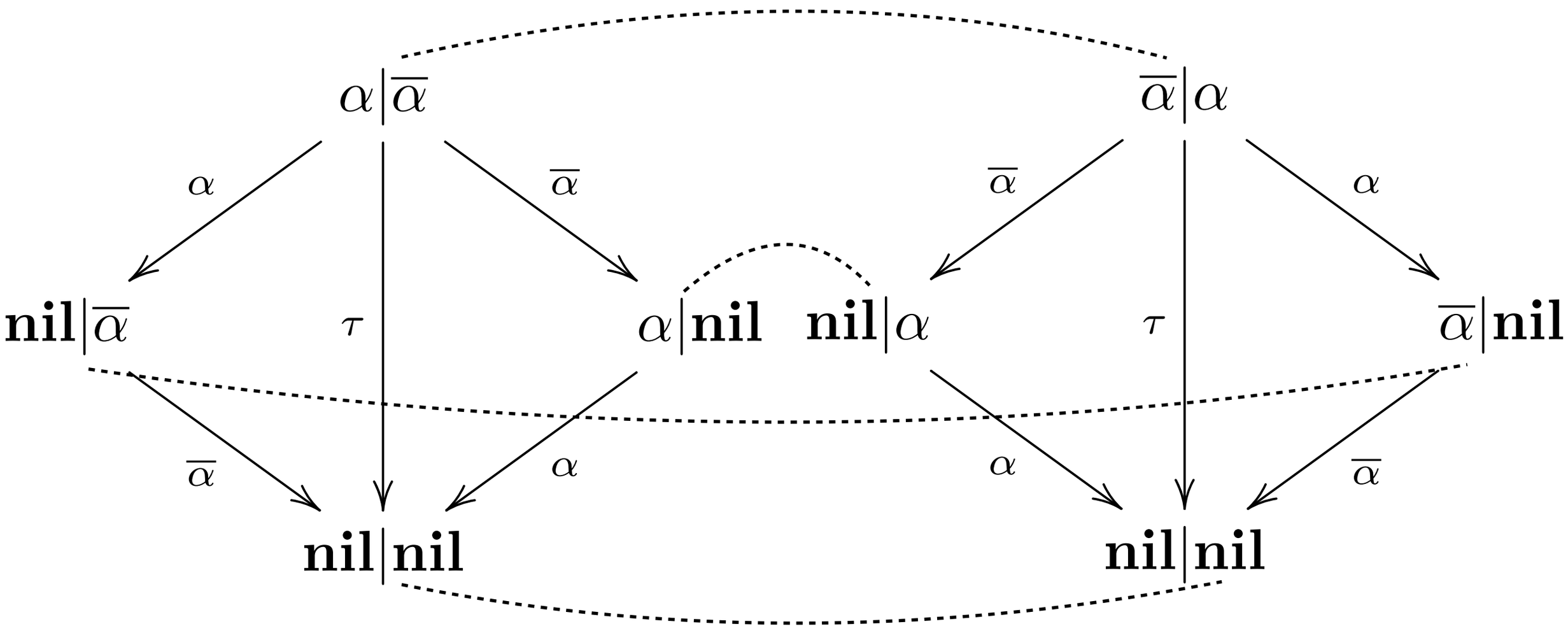


Graph isomorphism

$G = (V, E)$ $\qquad$ $G' = (V', E')$

$f : V \to V'$ bijective

$v \xrightarrow{\mu} w \quad \Leftrightarrow \quad f(v) \xrightarrow{\mu} f(w)$

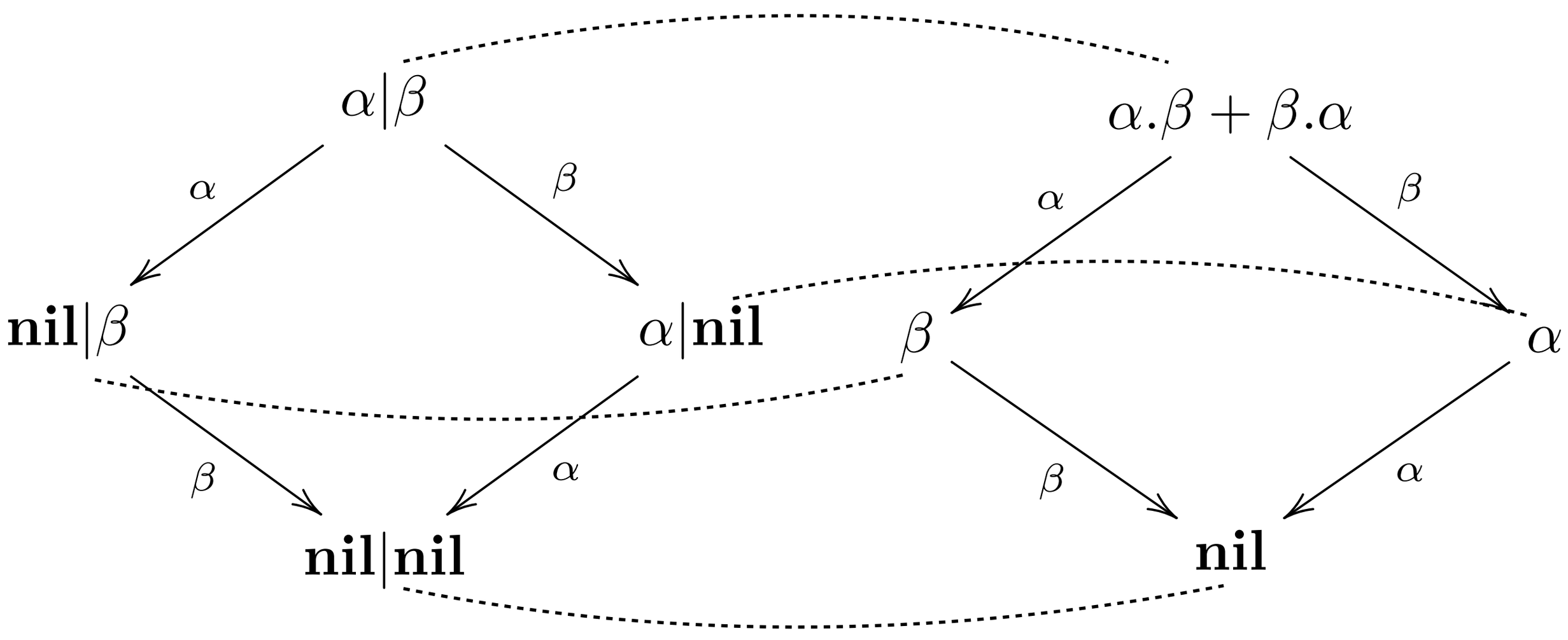


Graph isomorphism

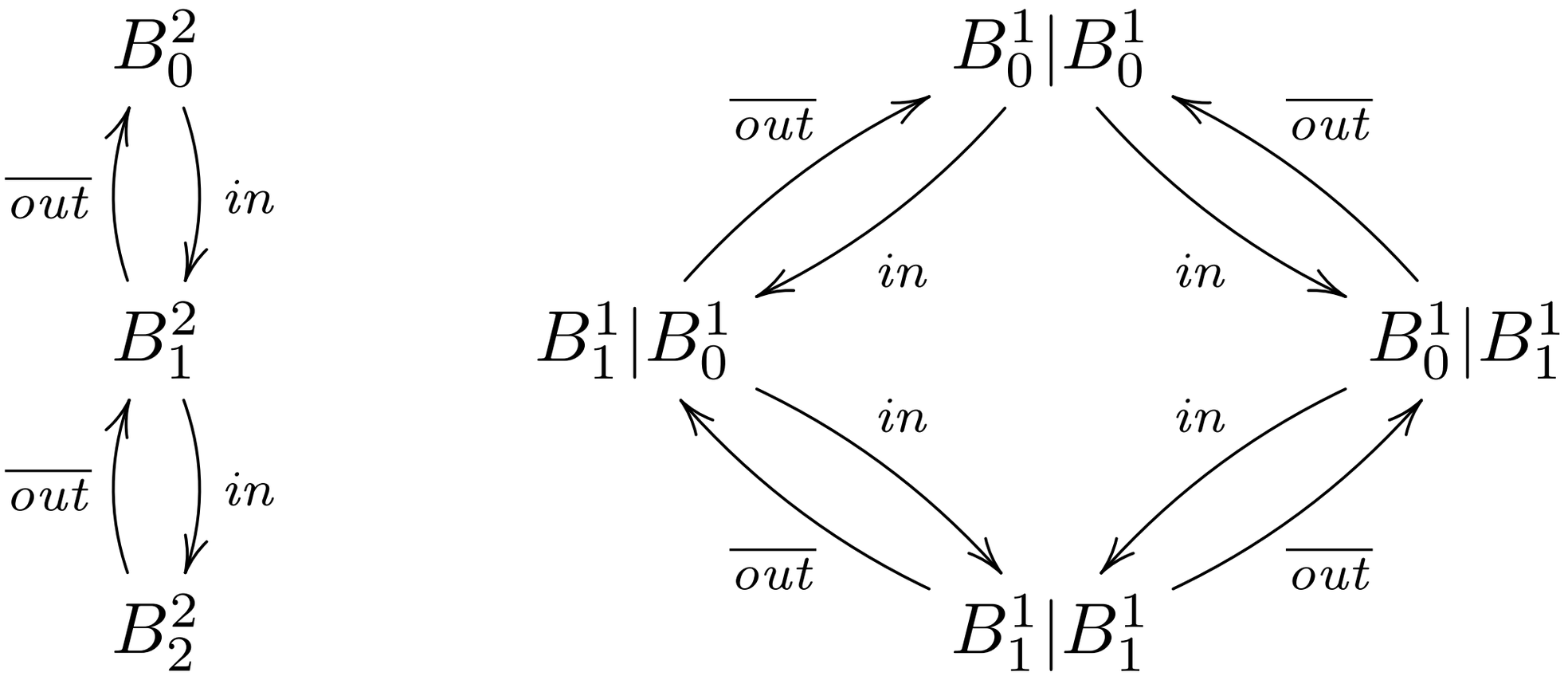
$$G = (V, E) \qquad G' = (V', E')$$

$$v \xrightarrow{\mu} w \quad \Leftrightarrow \quad f(v) \xrightarrow{\mu} f(w)$$

$f : V \to V'$ bijective



Equivalent or not?



intuitively equivalent, but not isomorphic!

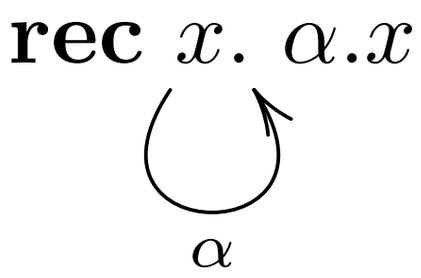
Iso is too strict

if two processes have isomorphic LTS,
then they must be considered as equivalent

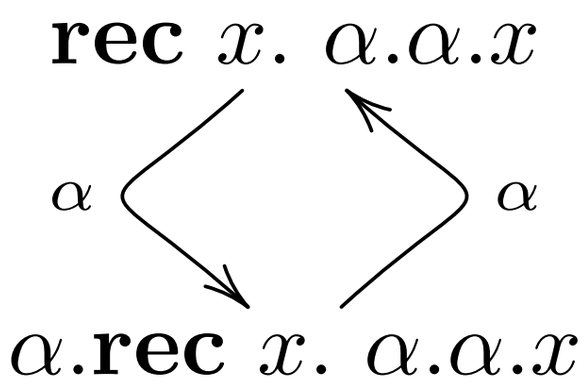
graph isomorphism captures some interesting equivalences

but is it enough?

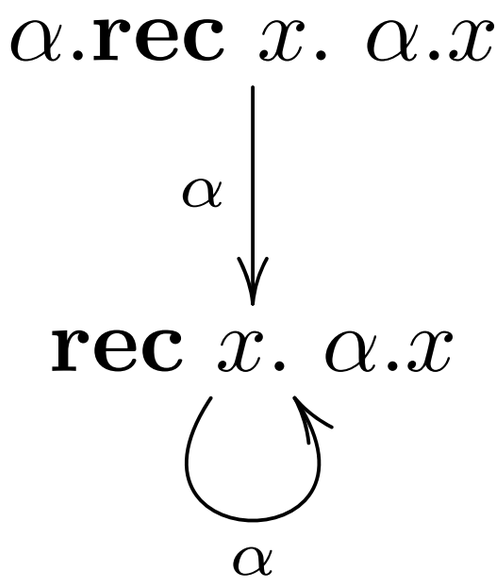
$P \triangleq \alpha.P$



$Q \triangleq \alpha.\alpha.Q$



$\alpha.P$



intuitively equivalent, but not isomorphic!

CCS

Trace equivalence

Trace equivalence

in automata theory: language equivalence

notion of (finite) trace $p \xrightarrow{\mu_1 \mu_2 \cdots \mu_k} q$

$$p = p_0 \xrightarrow{\mu_1} p_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_k} p_k = q$$

(finite) trace semantics of a process

$$\mathcal{T}(p) = \{\mu_1 \mu_2 \cdots \mu_k \mid \exists q.\ p \xrightarrow{\mu_1 \mu_2 \cdots \mu_k} q\}$$

two processes are *trace equivalent*
if they have the same trace semantics

$$p \equiv_{\mathsf{tr}} q \text{ iff } \mathcal{T}(p) = \mathcal{T}(q)$$

is trace equivalent a good notion for concurrent systems?

Trace equivalence

graph isomorphism implies trace equivalence

we preserves all the useful equivalences seen before

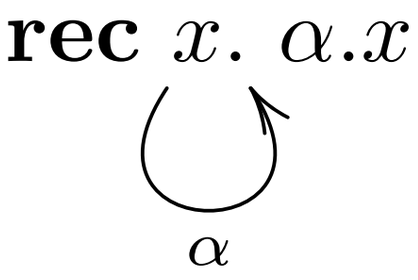
$$p \equiv_{\mathsf{tr}} p + \mathbf{nil} \equiv_{\mathsf{tr}} p + p \equiv_{\mathsf{tr}} p|\mathbf{nil}$$

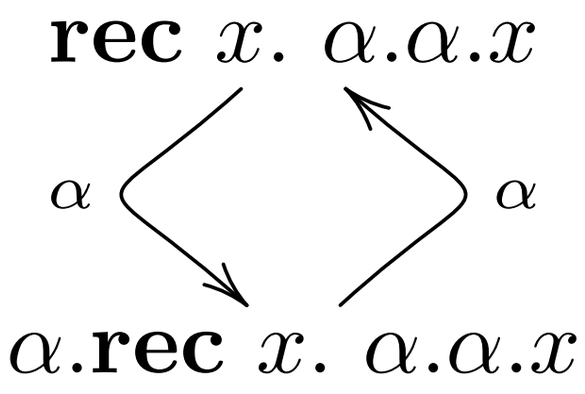
$$p + q \equiv_{\mathsf{tr}} q + p$$

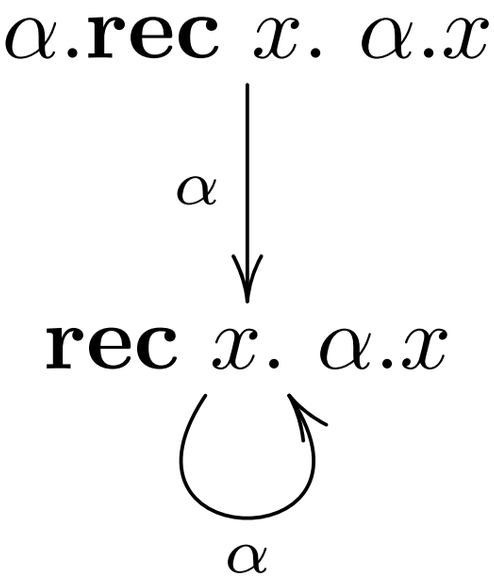
$$p|q \equiv_{\mathsf{tr}} q|p$$

trace semantics is prefix closed
(if a trace is present, all its prefixes are also present)

Trace equivalence





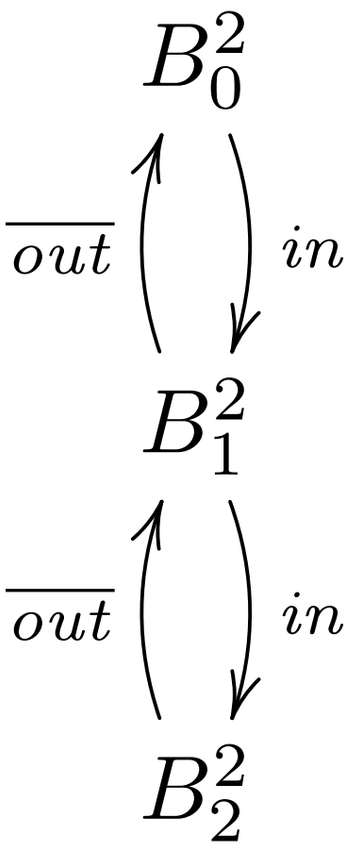


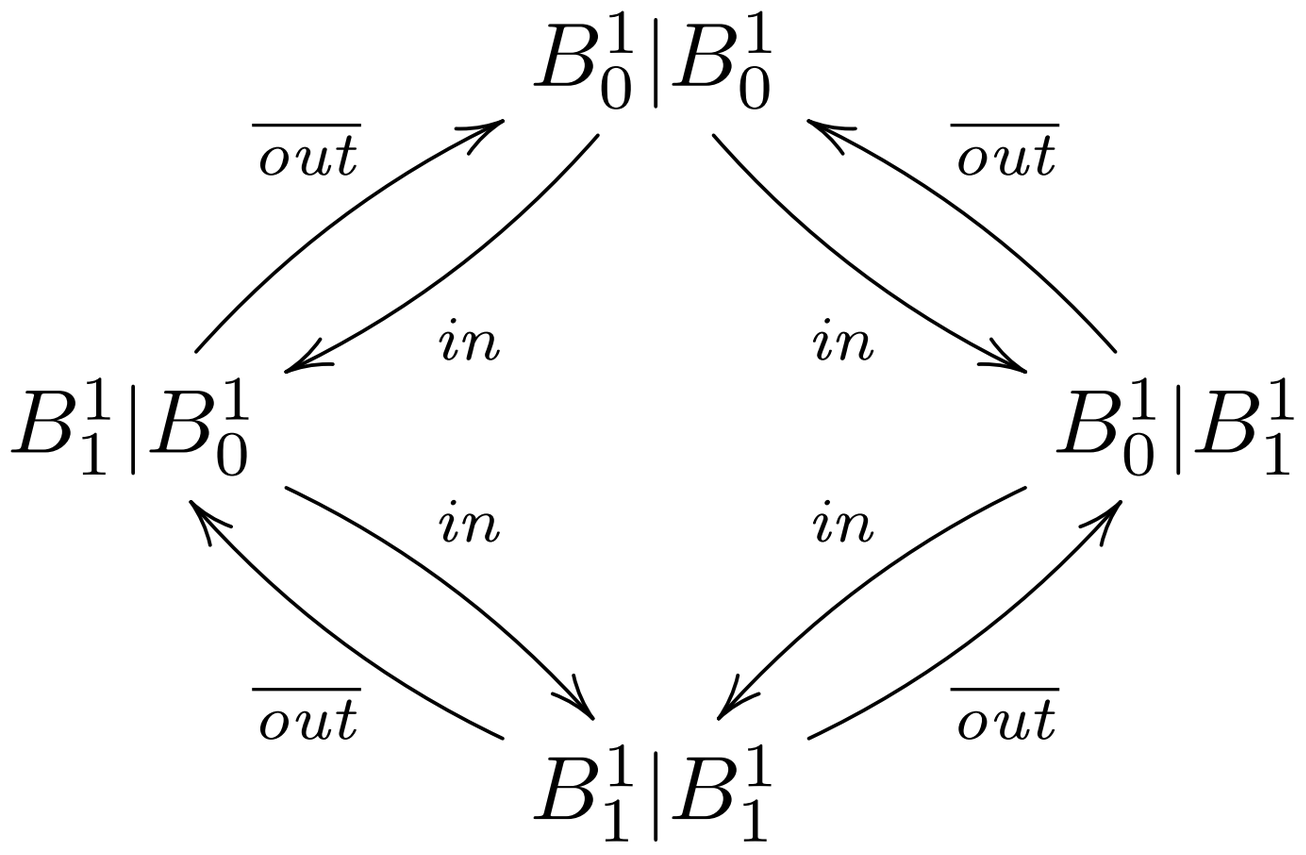
$$\mathcal{T}(\mathbf{rec}\ x.\ \alpha.x) = \{\alpha^n \mid n \in \mathbb{N}\}$$

$$\mathcal{T}(\mathbf{rec}\ x.\ \alpha.\alpha.x) = \{\alpha^n \mid n \in \mathbb{N}\}$$

$$\mathcal{T}(\alpha.\mathbf{rec}\ x.\ \alpha.x) = \{\alpha^n \mid n \in \mathbb{N}\}$$

Trace equivalence



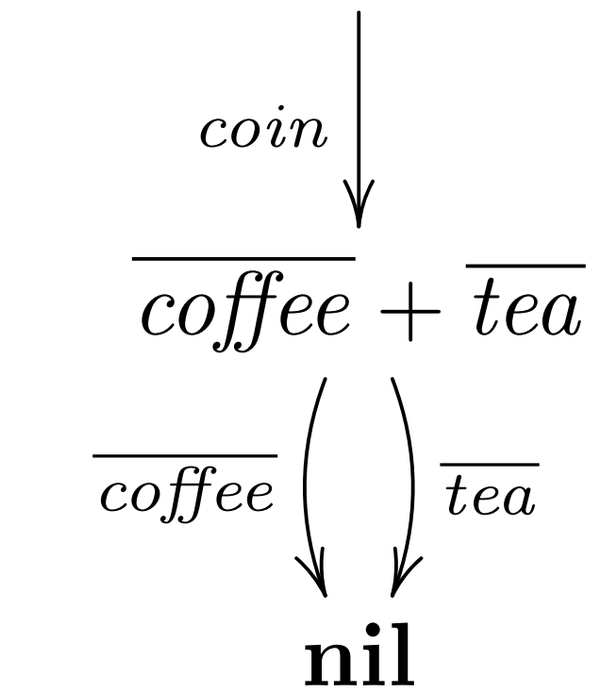


$$\mathcal{T}(B_0^2) = \mathcal{T}(B_0^1|B_0^1)$$

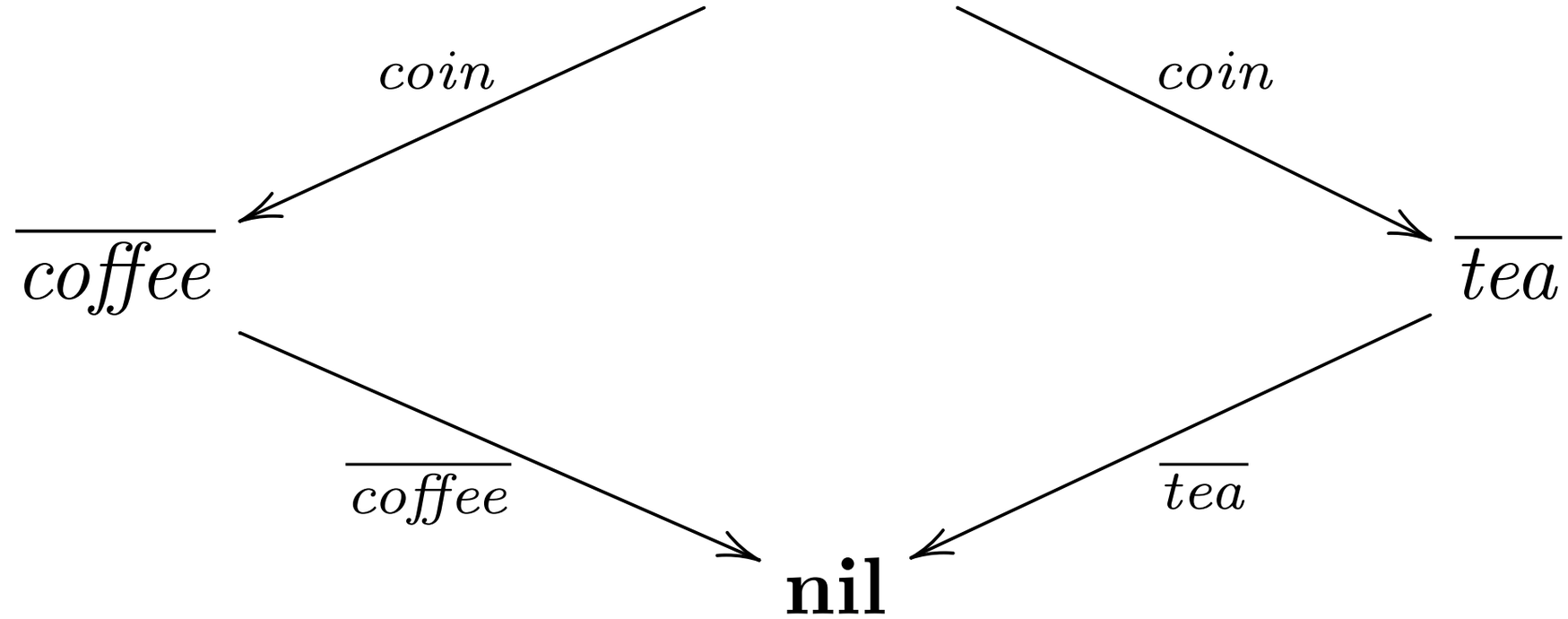
tentative description:
$0 \leq \#in - \#out \leq 2$ (for any prefix)

Two Vending Machines

$coin.(\overline{coffee} + \overline{tea})$



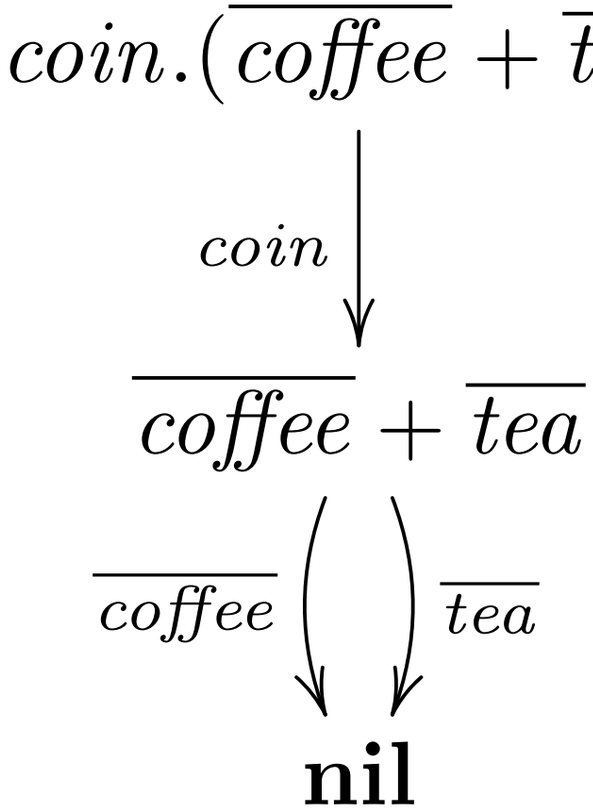
$coin.\overline{coffee} + coin.\overline{tea}$



not isomorphic, but trace equivalent

Customer's view

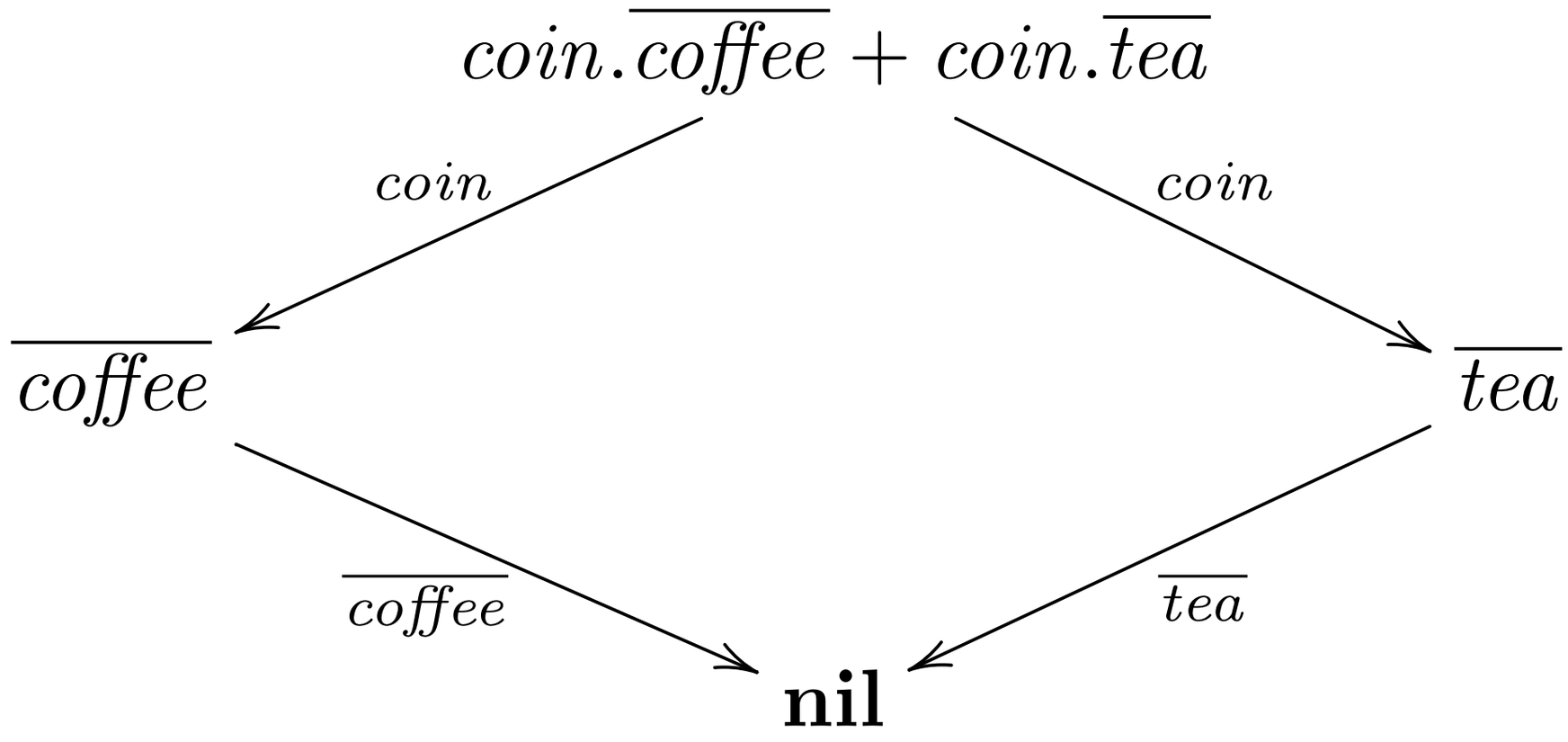
which vending machine would you prefer?



insert a coin, then choose the drink

Customer's view

which vending machine would you prefer?

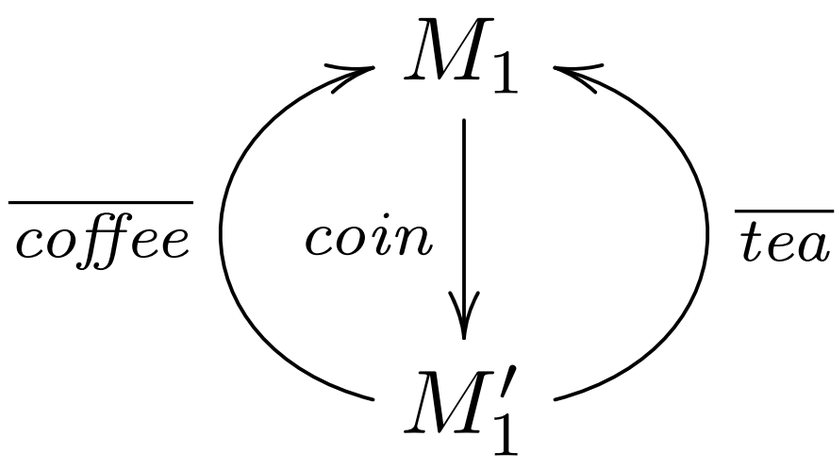


insert a coin, then get the drink chosen by the machine

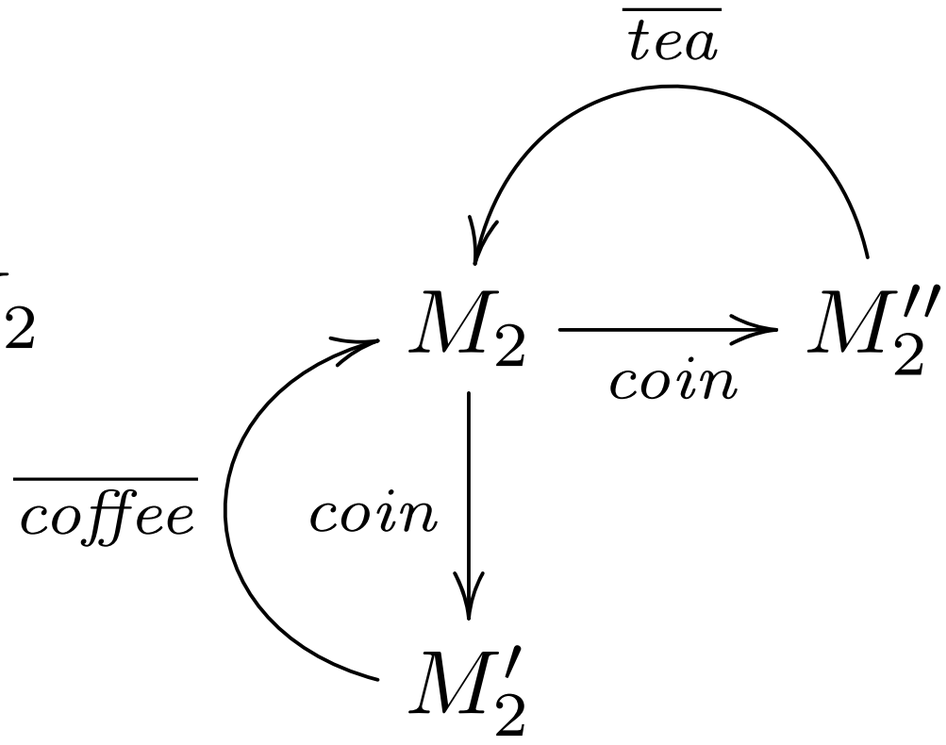
Recursive Machines

$$M_2 \triangleq coin.\overbrace{\overline{coffee}.M_2}^{M_2'} + coin.\overbrace{\overline{tea}.M_2}^{M_2''}$$

$$M_1 \triangleq coin.\overbrace{(\overline{coffee}.M_1 + \overline{tea}.M_1)}^{M_1'}$$



$$M_1 \equiv_{\mathsf{tr}} M_2$$



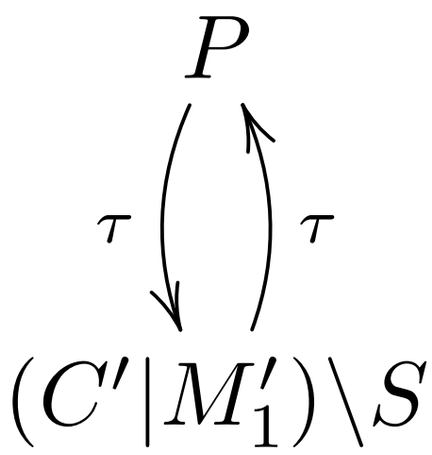
System View

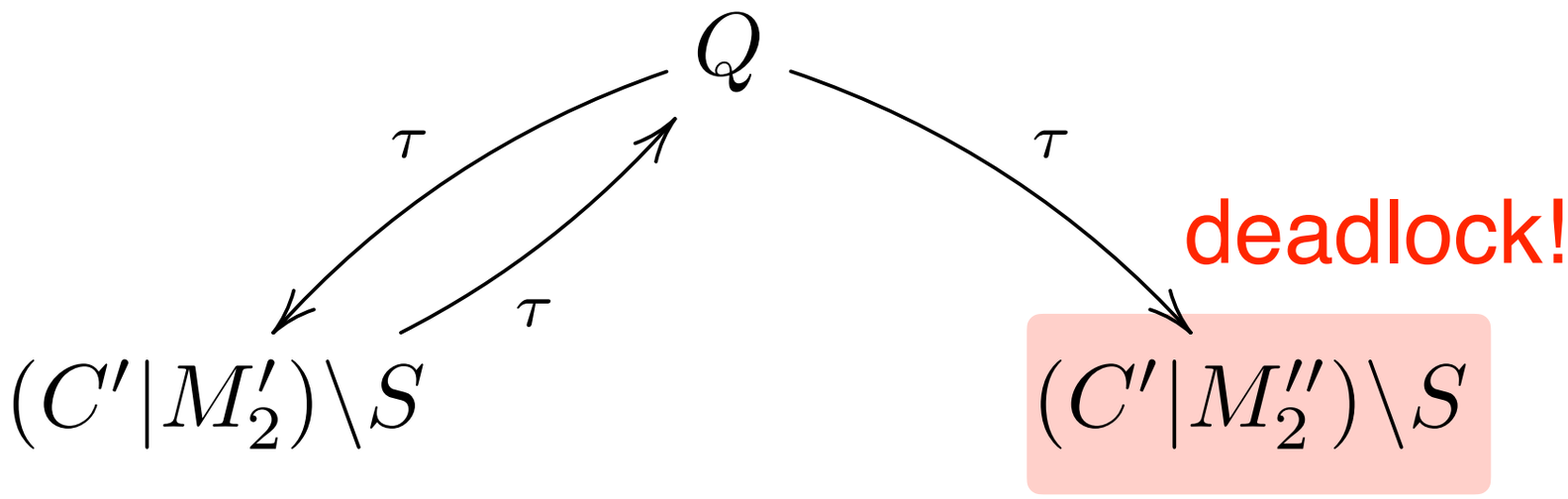
$$M_1 \triangleq coin.\overbrace{(\overline{coffee}.M_1 + \overline{tea}.M_1)}^{M_1'}$$

$$M_1 \equiv_{\mathsf{tr}} M_2 \qquad M_2 \triangleq coin.\overbrace{\overline{coffee}.M_2}^{M_2'} + coin.\overbrace{\overline{tea}.M_2}^{M_2''}$$

$$C \triangleq \overline{coin}.\overbrace{coffee.C}^{C'} \qquad S \triangleq \{coin, coffee, tea\}$$

$$P \triangleq (C|M_1)\backslash S \qquad P \equiv_{\mathsf{tr}} Q \qquad Q \triangleq (C|M_2)\backslash S$$





Coming next: bisimilarity

graph isomorphism distinguishes too many processes

trace equivalence identifies too many processes

we need some notion of equivalence in between the two

we introduce the notion of *strong bisimilarity*

as a game

as a fixpoint

as a logical equivalence

to keep in mind: two processes are equivalent unless we have some good reasons to distinguish them