FIT Summer School
Novi Sad, June 2009

Concurrent and Global Computing

Part I: The theory of concurrent and distributed computation

Marina Lenisa
Università di Udine, Italy

Overview of Part I: The theory of concurrent and distributed computation

- Introduction
- Classical computational systems vs modern computational systems
- Reactive Systems
- The need for a theory of reactive systems
- Models of reactive systems:
  - CCS
  - π-calculus
  - ambient calculus

Classical View of Computational Systems

- A computational system is a black box which takes an input and produces an output.
- It is a program implementing an algorithm.
- It can be viewed as a (partial) function from memory states into memory states:

$$[\![SYS]\!] : \text{States} \longrightarrow \text{States}\ , \quad \text{States} = \text{Var} \rightarrow \text{Val}$$

- Non-termination is undesired.
- In case of termination the result is unique.

Modern View of Computational Systems

Many modern computational systems are not describable via the classical model:

- operating systems
- communications protocols
- control programs
- mobile phones
- vending machines
- ...

Reactive Systems

A reactive system computes by reacting to stimuli from its environment.

Key Issues:

- communication and interaction
- parallelism

Non-termination is good.

The result (if any) does not have to be unique.

Analysis of Reactive Systems

Questions:

- How can we develop (design) a system that “works”?
- How do we analyze (verify) such a system?

Facts:

Even short parallel programs may be hard to analyze.

Phenomena:

- Deadlock.
- Starvation.
- Safety properties.
- Liveness properties.
- Security.

The Need for a Theory

We need abstract models of reactive systems and formal methods for rigorously reasoning on them, otherwise ...

- Intel's Pentium-II bug in floating point division unit
- Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer
- Mars Pathfinder
- ...

CCS as a Model of Reactive Systems

CCS, [Robin Milner 80's]
Process algebra called "Calculus of Communicating Systems".

Process Algebras
Concurrent (parallel) processes have an algebraic structure.

1. a few atomic processes;
2. composition operations are introduced for building more complex processes:

$$\boxed{P_1} \text{ op } \boxed{P_2} \Longrightarrow \boxed{P_1 \text{ op } P_2}$$

A CCS Process: Black Box View

What is a CCS Process to its Environment?

A CCS process is a computing agent that may communicate with its environment via its interface.

Interface: collection of communication ports/channels, together with an indication of whether they are used for input or output.

Example: A Computer Scientist

Process interface: coffee (input port)
$\overline{\text{coin}}$, $\overline{\text{pub}}$ (output ports)

Question: How do we describe the behavior of the “black box”?

CCS Basics (Sequential Fragment)

- Nil (or 0) process (the only atomic process)
- action prefixing ($a.P$)
- names and recursive definitions ($\overset{\text{def}}{=}$)
- non-deterministic choice ($+$)

CCS Basics (Parallelism and Renaming)

- parallel composition ($|$)
  (synchronous communication between two components = handshake synchronization)
- restriction ($P \backslash L$)
- relabelling ($P[f]$)

Definition of CCS (channels, actions, process names)

Let

- A be a set of channel names (e.g. *tea*, *coffee* are channel names)
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\mathcal{A} = \{\overline{a} \mid a \in A\}$ (elements of A are called names and those of $\overline{A}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$
- $Act = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - τ is the internal or silent action

  (e.g. τ, *tea*, $\overline{coffee}$ are actions)
- K is a set of process names (constants) (e.g. CM).

Definition of CCS (expressions)

$P :=$	K	$\mid$	process constants ($K \in \mathcal{K}$)
	$\alpha.P$	$\mid$	prefixing ($\alpha \in Act$)
	$\sum_{i \in I} P_i$	$\mid$	summation (I is an arbitrary index set)
	$P_1 \mid P_2$	$\mid$	parallel composition
	$P \setminus L$	$\mid$	restriction ($L \subseteq \mathcal{A}$)
	$P[f]$	$\mid$	relabelling ($f : Act \to Act$) such that

- $f(\tau) = \tau$
- $f(\overline{a}) = \overline{f(a)}$

The set of all terms generated by the abstract syntax is the set of CCS process expressions (and is denoted by $\mathcal{P}$).

Definition of CCS (defining equations)

CCS program.
A collection of defining equations of the form

$$K \stackrel{\text{def}}{=} P$$

where $K \in \mathcal{K}$ is a process constant and $P \in \mathcal{P}$ is a CCS process expression.

- Only one defining equation per process constant.
- Recursion is allowed: e.g. $A \stackrel{\text{def}}{=} a.A|A$.

Operational Semantics for CCS: Labelled Transition System (LTS)

Given a collection of CCS defining equations, we define the LTS $(\mathit{Proc}, \mathit{Act}, \{\xrightarrow{\alpha} \mid \alpha \in \mathit{Act}\})$:

- $\mathit{Proc} = \mathcal{P}$ (the set of all CCS process expressions)
- $\mathit{Act} = \mathcal{L} \cup \{\tau\}$ (the set of all CCS actions including τ)
- $\xrightarrow{\alpha} \subseteq \mathit{Proc} \times \mathit{Proc}$ is a binary relation on states called the transition relation, for each $\alpha \in \mathit{Act}$.
  The transition relation is given by Structural Operational Semantics (SOS) rules.

We will use the infix notation $P \xrightarrow{\alpha} P'$ meaning that $(P, P') \in \xrightarrow{\alpha}$.

Structural Operational Semantics for CCS

Structural Operational Semantics (SOS), [G. Plotkin 1981]: small-step operational semantics where the behaviour of a system is inferred using syntax driven rules.

The CCS transition relation is given by SOS rules of the form:

$$\text{RULE} \quad \frac{\textit{premises}}{\textit{conclusions}} \quad \textit{conditions}$$

SOS rules for CCS ($\alpha \in Act$, $a \in \mathcal{L}$)

$$\text{ACT}\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \text{SUM}_j\ \frac{P_j \xrightarrow{\alpha} P'_j}{\Sigma_{i\in I} P_i \xrightarrow{\alpha} P'_j}\ j \in I$$

$$\text{COM}_1\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \text{COM}_2\ \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\text{COM}_3\ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\text{RES}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L}\ \alpha, \overline{\alpha} \notin L \qquad \text{REL}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

$$\text{CON}\ \frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'}\ K \stackrel{\text{def}}{=} P$$

Behavioural Equivalence

Implementation

$$CM \stackrel{\text{def}}{=} coin.\overline{coffee}.CM$$

$$CS \stackrel{\text{def}}{=} \overline{pub}.\overline{coin}.coffee.CS$$

$$Uni \stackrel{\text{def}}{=} (CM \mid CS) \backslash \{ coin, coffee \}$$

Specification

$$Spec \stackrel{\text{def}}{=} \overline{pub}.Spec$$

Question: Are the processes *Uni* and *Spec* behaviourally equivalent?

$$Uni \simeq Spec$$

Goals

What should a reasonable behavioural equivalence satisfy?

- Abstract from states (consider only the behaviour – actions)
- Abstract from nondeterminism
- Abstract from internal behaviour

What else should a reasonable behavioural equivalence satisfy?

- Reflexivity: $P \simeq P$ for each process P
- Transitivity: $Spec_0 \simeq Spec_1 \simeq Spec_2 \simeq \ldots \simeq Impl$ gives that $Spec_0 \simeq Impl$
- Symmetry: $P \simeq Q$ iff $Q \simeq P$.

Congruence

$C(P)$

$C(Q)$

Congruence Property

$P \simeq Q$ implies that $C[P] \simeq C[Q]$

Trace Equivalence

Let $(Proc, Act, \{\xrightarrow{\alpha} \mid \alpha \in Act\})$ be an LTS.

Trace Set for $s \in Proc$

$$Traces(s) = \{w \in Act^* \mid \exists s' \in Proc.\ s \xrightarrow{w} s'\}$$

Let $s, t \in Proc$.

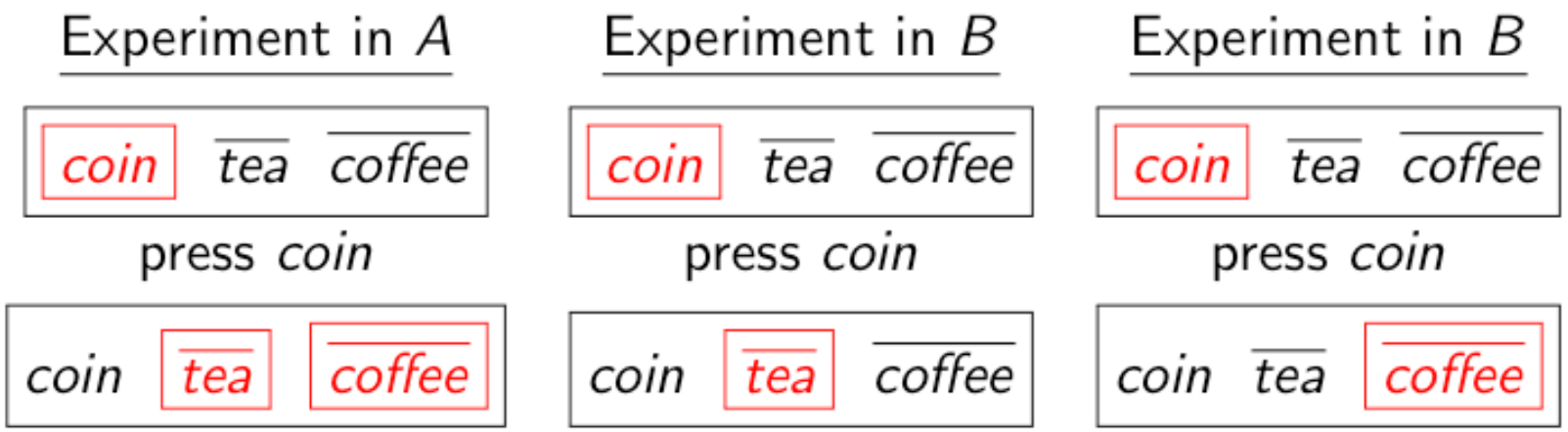
Trace Equivalence

We say that s and t are trace equivalent ($s \simeq_t t$) if and only if

$$Traces(s) = Traces(t)$$

Is this a "good" behavioural equivalence?

Black Box Experiments



Main Idea

Two processes are behaviorally equivalent if and only if an external observer cannot tell them apart.

Strong Bisimilarity

Let $(Proc, Act, \{\xrightarrow{\alpha} \mid \alpha \in Act\})$ be an LTS.

Strong Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a strong bisimulation iff whenever $(s, t) \in R$ then for each $\alpha \in Act$:

- if $s \xrightarrow{\alpha} s'$ then $t \xrightarrow{\alpha} t'$ for some t' such that $(s', t') \in R$
- if $t \xrightarrow{\alpha} t'$ then $s \xrightarrow{\alpha} s'$ for some s' such that $(s', t') \in R$.

Strong Bisimilarity

Two processes $p_1, p_2 \in Proc$ are strongly bisimilar ($p_1 \sim p_2$) if and only if there exists a strong bisimulation R such that $p_1 R p_2$.

$$\sim = \bigcup \{R \mid R \text{ is a strong bisimulation}\}$$

Coinduction Principle:

$$\frac{R \text{ bisimulation} \quad p_1 R p_2}{p_1 \sim p_2}$$

Basic Properties of Strong Bisimilarity

Theorem

$\sim$ is an equivalence relation (reflexive, symmetric and transitive).

Theorem

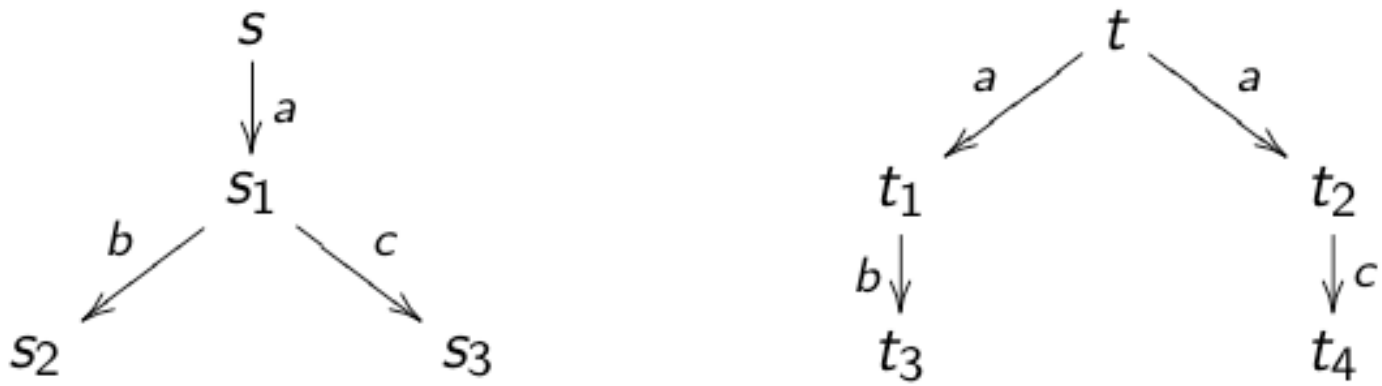
$\sim$ is the largest strong bisimulation.

Theorem

$s \sim t$ if and only if for each $\alpha \in Act$:

- if $s \xrightarrow{\alpha} s'$ then $t \xrightarrow{\alpha} t'$ for some t' such that $s' \sim t'$
- if $t \xrightarrow{\alpha} t'$ then $s \xrightarrow{\alpha} s'$ for some s' such that $s' \sim t'$.

How to show Non-bisimilarity?



To prove that $s \not\sim t$:

- Enumerate all binary relations and show that none of them at the same time contains (s, t) and is a strong bisimulation. (Expensive: $2^{|Proc|^2}$ relations.)
- Make certain observations which enable us to disqualify many bisimulation candidates in one step.
- Use the game characterization of strong bisimilarity.

Strong Bisimulation Game

Let $(Proc, Act, \{\xrightarrow{\alpha} | \alpha \in Act\})$ be an LTS and $s, t \in Proc$.

We define a two-player game of an attacker and a defender starting from s and t .

- The game is played in rounds, and configurations of the game are pairs of states from $Proc \times Proc$.
- In every round exactly one configuration is called current. Initially the configuration (s, t) is the current one.

Intuition:
The defender wants to show that s and t are strongly bisimilar while the attacker aims at proving the opposite.

Rules of the Bisimulation Games

Game Rules:

In each round the players change the current configuration as follows:

1. the attacker chooses one of the processes in the current configuration and makes an $\stackrel{\alpha}{\rightarrow}$-move for some $\alpha \in \mathit{Act}$, and
2. the defender must respond by making an $\stackrel{\alpha}{\rightarrow}$-move in the other process under the same action α.

The newly reached pair of processes becomes the current configuration. The game then continues by another round.

Result of the Game:

- If one player cannot move, the other player wins.
- If the game is infinite, the defender wins.

Game Characterization of Strong Bisimilarity

Theorem.

- States s and t are strongly bisimilar if and only if the defender has a (universal) winning strategy starting from the configuration (s, t).
- States s and t are not strongly bisimilar if and only if the attacker has a (universal) winning strategy starting from the configuration (s, t).

Remark.

The bisimulation game can be used to prove both bisimilarity and nonbisimilarity of two processes. It very often provides elegant arguments for the negative case.

Strong Bisimilarity is a Congruence for CCS Operations

Theorem.

Let P and Q be CCS processes such that $P \sim Q$. Then

- $\alpha.P \sim \alpha.Q$ for each action $\alpha \in Act$
- $P + R \sim Q + R$ and $R + P \sim R + Q$ for each CCS process R
- $P|R \sim Q|R$ and $R|P \sim R|Q$ for each CCS process R
- $P[f] \sim Q[f]$ for each relabelling function f
- $P\backslash L \sim Q\backslash L$ for each set of labels L

Other Properties of Strong Bisimilarity

The Following Properties hold for all CCS Processes P, Q, R:

- $P + Q \sim Q + P$
- $P|Q \sim Q|P$
- $P + 0 \sim P$
- $P|0 \sim P$
- $(P + Q) + R \sim P + (Q + R)$
- $(P|Q)|R \sim P|(Q|R)$

Problems with Internal Actions

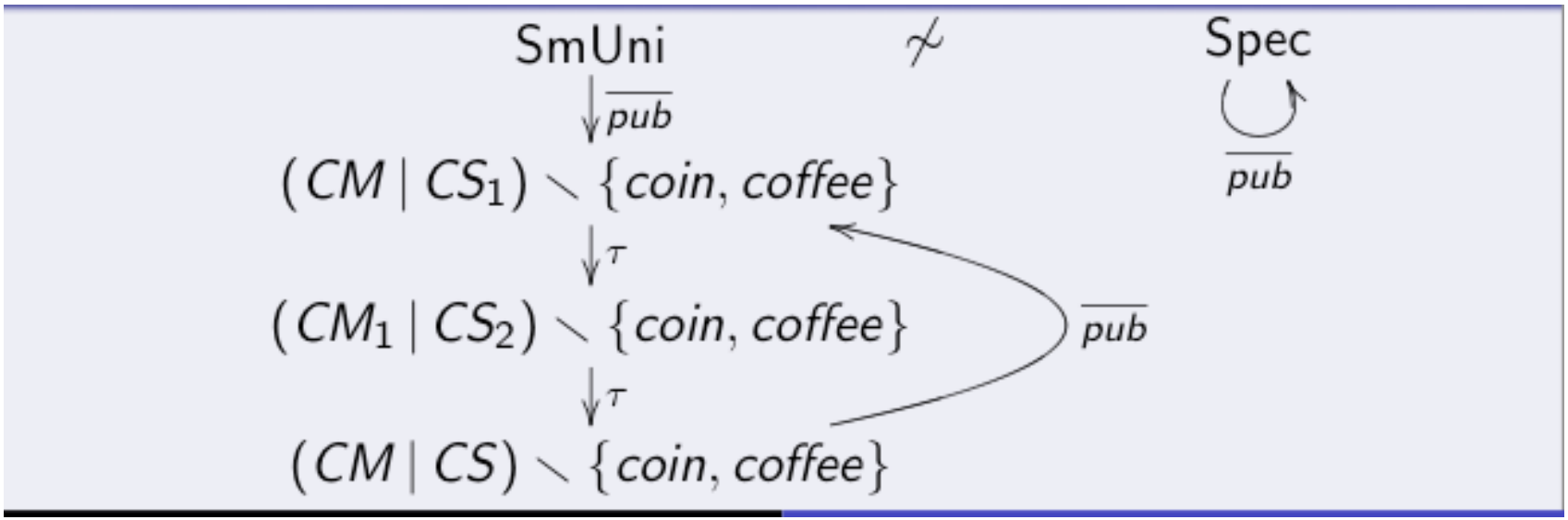
Question:

Does $a.\tau.0 \sim a.0$ hold? NO!

Problem:

Strong bisimilarity does not abstract away from τ actions.

Example: $SmUni \not\sim Spec$



Weak Transition Relation

Let $(Proc, Act, \{\xrightarrow{\alpha} | \alpha \in Act\})$ be an LTS such that $\tau \in Act$.

Definition of Weak Transition Relation.

$$\stackrel{\alpha}{\Longrightarrow} = \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{\alpha} \circ (\xrightarrow{\tau})^* & \text{if } \alpha \neq \tau \\ (\xrightarrow{\tau})^* & \text{if } \alpha = \tau \end{cases}$$

What does $s \stackrel{\alpha}{\Longrightarrow} t$ informally mean?

- If $\alpha \neq \tau$ then $s \stackrel{\alpha}{\Longrightarrow} t$ means that
  from s we can get to t by doing zero or more τ actions, followed by the action α, followed by zero or more τ actions.
- If $\alpha = \tau$ then $s \stackrel{\alpha}{\Longrightarrow} t$ means that
  from s we can get to t by doing zero or more τ actions.

Weak Bisimilarity

Let $(Proc, Act, \{\xrightarrow{\alpha} | \alpha \in Act\})$ be an LTS such that $\tau \in Act$.

Weak Bisimulation.

A binary relation $R \subseteq Proc \times Proc$ is a weak bisimulation if whenever $(s, t) \in R$ then for each $\alpha \in Act$ (including τ):

- if $s \xrightarrow{\alpha} s'$ then $t \stackrel{\alpha}{\Longrightarrow} t'$ for some t' such that $(s', t') \in R$
- if $t \xrightarrow{\alpha} t'$ then $s \stackrel{\alpha}{\Longrightarrow} s'$ for some s' such that $(s', t') \in R$.

Weak Bisimilarity.

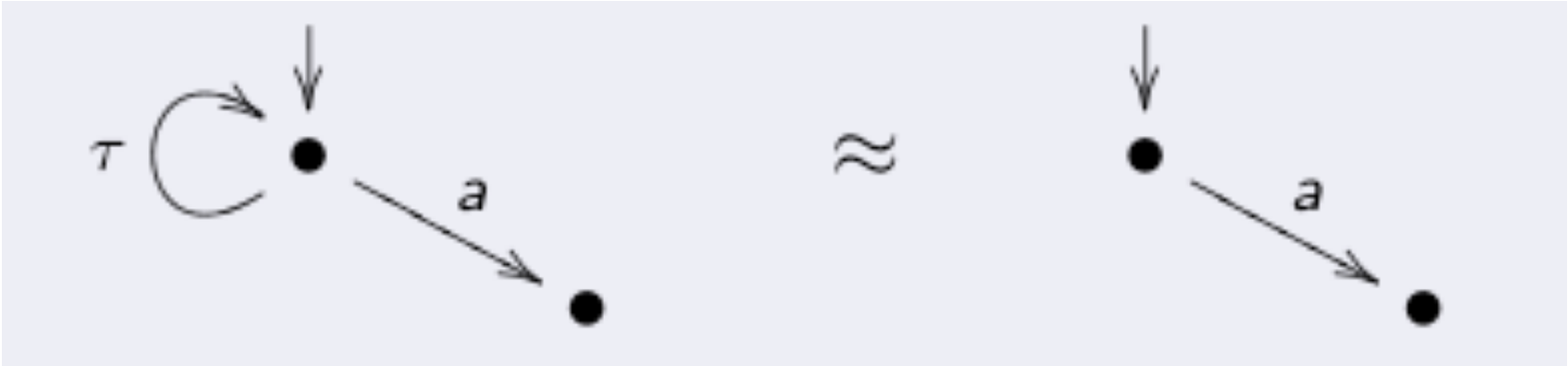
Two processes $p_1, p_2 \in Proc$ are weakly bisimilar ($p1 \approx p2$) if and only if there exists a weak bisimulation R such that $(p_1, p_2) \in R$.

$$\approx = \bigcup \{R \mid R \text{ is a weak bisimulation } \}$$

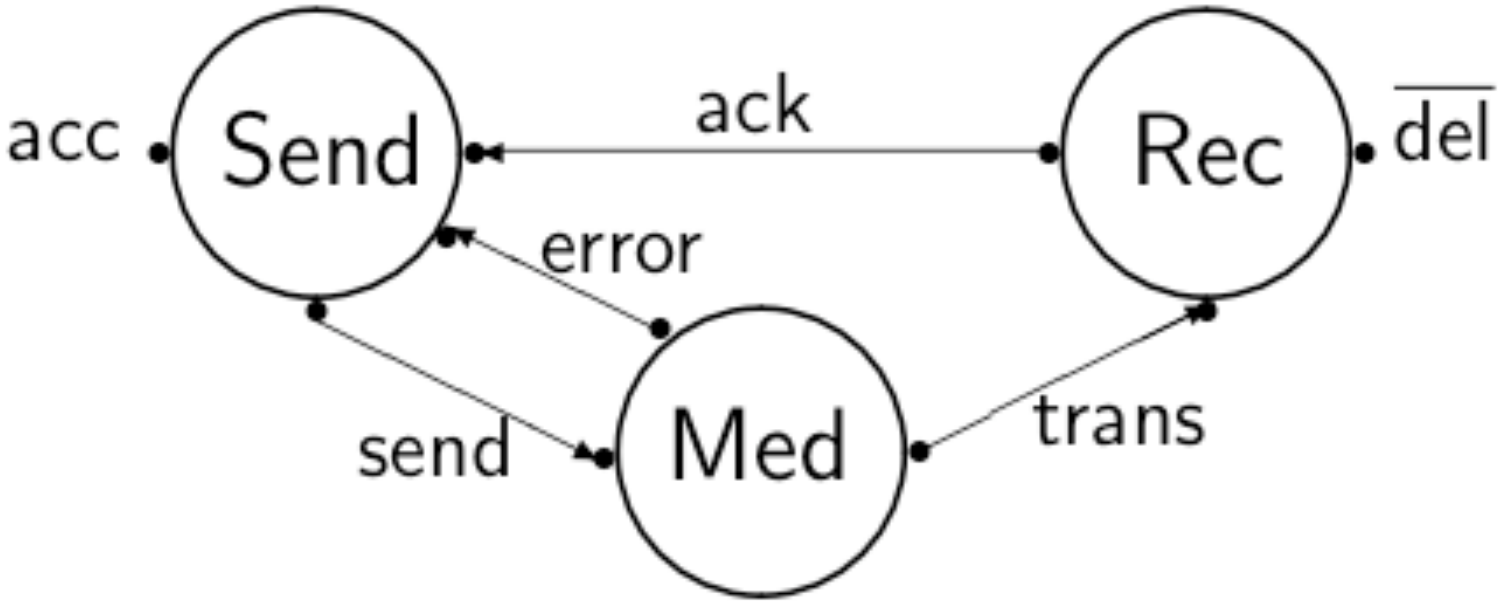
Weak Bisimilarity – Properties

Properties of $\approx$:

- an equivalence relation
- the largest weak bisimulation
- validates lots of natural laws, e.g.
  - $a.\tau.P \approx a.P$
  - $P + \tau.P \approx \tau.P$
  - $a.(P + \tau.Q) \approx a.(P + \tau.Q) + a.Q$
  - $P + Q \approx Q + P \quad P|Q \approx Q|P \quad P + 0 \approx P \quad \ldots$
- strong bisimilarity is included in weak bisimilarity ($\sim \subseteq \approx$)
- abstracts from τ loops



Case Study: Communication Protocol



$$
\begin{array}{lcl}
\text{Send} & \stackrel{\text{def}}{=} & \text{acc.Sending} \\
\text{Sending} & \stackrel{\text{def}}{=} & \overline{\text{send}}\text{.Wait} \\
\text{Wait} & \stackrel{\text{def}}{=} & \text{ack.Send} + \text{error.Sending}
\end{array}
\qquad
\begin{array}{lcl}
\text{Rec} & \stackrel{\text{def}}{=} & \text{trans.Del} \\
\text{Del} & \stackrel{\text{def}}{=} & \overline{\text{del}}\text{.Ack} \\
\text{Ack} & \stackrel{\text{def}}{=} & \overline{\text{ack}}\text{.Rec}
\end{array}
$$

$$
\begin{array}{lcl}
\text{Med} & \stackrel{\text{def}}{=} & \text{send.Med'} \\
\text{Med'} & \stackrel{\text{def}}{=} & \tau\text{.Err} + \overline{\text{trans}}\text{.Med} \\
\text{Err} & \stackrel{\text{def}}{=} & \overline{\text{error}}\text{.Med}
\end{array}
$$

Verification Question

$$Impl \stackrel{def}{=} (Send \mid Med \mid Rec) \backslash \{send, trans, ack, error\}$$

$$Spec \stackrel{def}{=} acc.del.Spec$$

Question: $Impl \stackrel{?}{\approx} Spec$

Draw the LTS of Impl and Spec and prove the equivalence.

Is Weak Bisimilarity a Congruence for CCS?

Theorem.
Let P and Q be CCS processes such that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$ for each action $\alpha \in Act$
- $P|R \approx Q|R$ and $R|P \approx R|Q$ for each CCS process R
- $P[f] \approx Q[f]$ for each relabelling function f
- $P\backslash L \approx Q\backslash L$ for each set of labels L.

What about choice?

$\tau.a.0 \approx a.0$ but $\tau.a.0 + b.0 \not\approx a.0 + b.0$

Conclusion.
Weak bisimilarity is not a congruence for CCS.

Weak Congruence

Definition.

$P \approx_c Q$ iff

- if $P \xrightarrow{\alpha} P'$ then there exists Q' s.t. $Q \xrightarrow{\alpha} Q'$ and $P' \approx Q'$
- if $Q \xrightarrow{\alpha} Q'$ then there exists P' s.t. $P \xrightarrow{\alpha} P'$ and $P' \approx Q'$

Then $\tau.a.0 \not\approx_c a.0$.

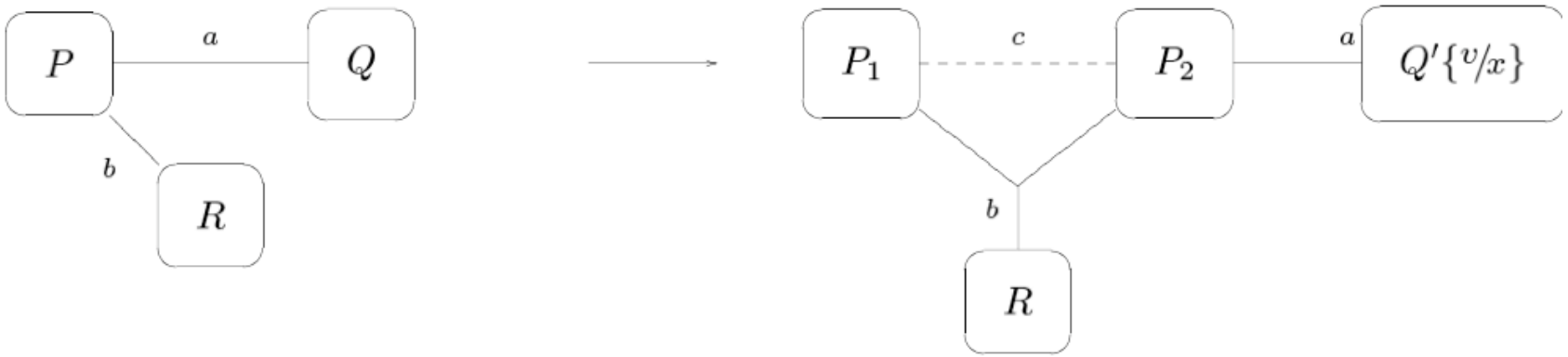
π-calculus: a calculus of mobile processes

Name mobility: Processes communicate via names (channels/links) and names may move (cf: hyper text links, mobile phones, object references, ...):

$$a(x).P|\overline{a}b.Q \xrightarrow{\tau} P\{b/x\}|Q$$

Mobile processes: the process linkage structure may change.

Mobility I: process creation

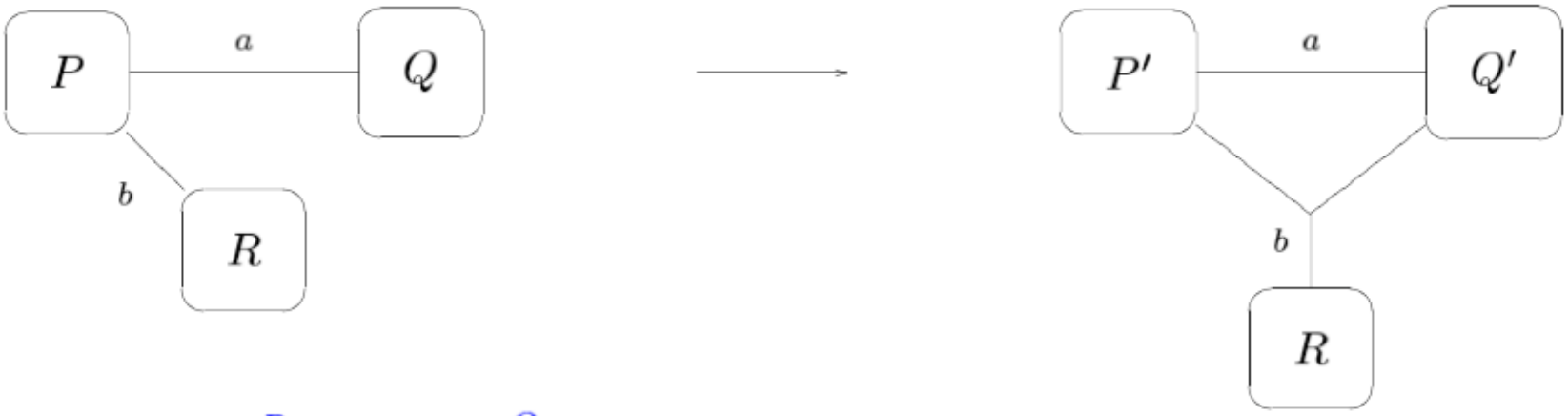


$$\overbrace{\overline{a}\,v\,.\,\boldsymbol{\nu} c\,(P_1 \mid P_2)}^{P} \mid \overbrace{a(x).\,Q'}^{Q} \mid R \longrightarrow \boldsymbol{\nu} c\,(P_1 \mid P_2) \mid Q'\{v/x\} \mid R$$

Mobility II: process termination

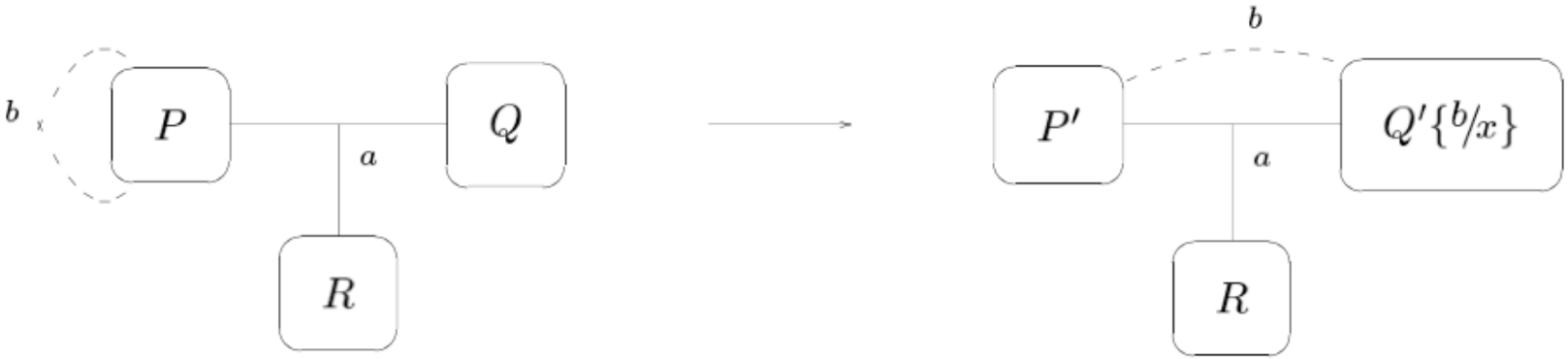
$$\overbrace{b(x).P'}^{P} \mid Q \mid \overbrace{\overline{b}\,v\,.0}^{R} \longrightarrow P'\{v/x\} \mid Q \mid 0$$

Mobility III: movement of communication links



$$\overbrace{\overline{a}\,b\,.\,P'}^{P} \mid R \mid \overbrace{a(x).\,Q'}^{Q} \longrightarrow P' \mid R \mid Q'\{^{b}/_{x}\}$$

Mobility IV: movement of private links



$$\overbrace{(\nu b)\,(\overline{a}\,b\,.\,P')}^{P} \mid \overbrace{a(x).\,Q'}^{Q} \mid R \longrightarrow (\nu b)\,(P' \mid Q'\{^{b}/_{x}\}) \mid R$$

scope extrusion

π-calculus Syntax

$a, b, \ldots, x, y, z, \ldots$ *Names*

Processes

P	::=	0	nil process
	$\mid$	$\Sigma_{i \in I} P_i$	sum
	$\mid$	$P \mid P$	parallel
	$\mid$	$(\nu a)P$	restriction
	$\mid$	$a(x).P$	input
	$\mid$	$\bar{a}b.P$	output
	$\mid$	$\tau.P$	silent action
	$\mid$	$!P$	replication $(= P \mid P \ldots)$

Prefixes

π	::=	$\bar{a}x$	output
	$\mid$	$a(x)$	input
	$\mid$	τ	silent

Replication replaces recursive definitions.

Structural Congruence $\equiv$

- if P, Q are α-convertible, then $P \equiv Q$
- Abelian monoidal laws for $|$: commutativity $P|Q \equiv Q|P$, associativity $(P|Q)|R \equiv P|(Q|R)$, 0 as unit $P|0 \equiv P$
- Abelian monoidal laws for $+$
- $!P \equiv P|!P$
- scope extension laws:

$$
\begin{array}{lcll}
(\nu x)0 & \equiv & 0 & \\
(\nu x)(P|Q) & \equiv & P|(\nu x)Q & \text{if } x \notin \mathrm{fn}(P) \\
(\nu x)(P+Q) & \equiv & P+(\nu x)Q & \text{if } x \notin \mathrm{fn}(P) \\
(\nu x)(\nu y)P & \equiv & (\nu y)(\nu x)P &
\end{array}
$$

LTS Operational Semantics

Labels $\quad \alpha ::= \tau \mid a(x) \mid \overline{a}x \mid \overline{a}(x)$

Rules

$$\text{STRUCT}\ \frac{P' \equiv P \quad P \xrightarrow{\alpha} Q \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$$

$$\text{PREFIX}\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \text{SUM}_j\ \frac{P_j \xrightarrow{\alpha} P'_j}{\Sigma_{i \in I} P_i \xrightarrow{\alpha} P'_j}\ j \in I$$

$$\text{PAR}\ \frac{P \xrightarrow{\alpha} P' \quad \text{bn}(\alpha) \cap \text{fn}(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \text{COM}\ \frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\overline{a}u} Q'}{P|Q \xrightarrow{\tau} P\{u/x\}|Q'}$$

$$\text{RES}\ \frac{P \xrightarrow{\alpha} P' \quad x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \qquad \text{OPEN}\ \frac{P \xrightarrow{\overline{a}(x)} P' \quad a \neq x}{(\nu x)P \xrightarrow{\overline{a}(x)} P'}$$

Example: names as access rights to resources

- process R: resource, e.g. printer
- process S: server, distributing access rights to the resource
- the access right is just to execute R; to model this, we use a name e as a trigger (guarding R by e):

$$(\nu e)(S|e.R)$$

  where $e.R$ is an abbreviation for $e(z).R$ (when the received name is irrelevant)
- R executes only it receives a signal on e
- the server S can invoke R by performing the action $\overline{e}$
- moreover S can send e to a client Q wishing to use R: Q asks S along channel c for the access key e

$$c(x).\overline{x}.Q|(\nu e)(\overline{c}e.S|e.R) \xrightarrow{\tau} (\nu e)(\overline{e}.Q|S|e.R) \xrightarrow{\tau} (\nu e)(Q|S|R)$$

Generalizing the Example

- a server S can send e to many clients, sharing R
- R can have several keys for making different things, e.g. $e_1.R_1 | e_2.R_2 \ldots$ and the server can send some of the keys to each client
- S can send two names d, e to a single client (avoiding other clients from receiving them). S first establish a private channel with a client, then S sends d, e along that channel:

$$(\nu p)\overline{c}p.\overline{p}d.\overline{p}e.S$$

The client must be prepared to receive a name, and then along that name to receive d and e:

$$c(p).p(x).p(y).Q$$

The Polyadic π-calculus

We introduce the abbreviations

$$\begin{array}{lll} \overline{c} < e_1 \ldots e_n > .P & \text{for} & (\nu p)\overline{c}p.\overline{p}e_1 \ldots . \overline{p}e_n.P \\ c(x_1 \ldots x_n).Q & \text{for} & cp.p(x_1) \ldots . p(x_n).Q \end{array}$$

where $p \notin \text{fn}(P, Q)$.

Then we have:

$$< e_1 \ldots e_n > .P | c(x_1 \ldots x_n).Q \xrightarrow{\tau} \ldots \xrightarrow{\tau} P | Q\{e_1 \ldots e_n / x_1 \ldots x_n\}$$

Late, Early Bisimilarities and Congruences

Late Bisimulation: symmetric relation $\mathcal{R}$ s.t.: $P\mathcal{R}Q$ and $P \xrightarrow{\alpha} P'$ implies

- if α is not an input then $\exists Q'.\ Q \xrightarrow{\alpha} Q' \ \wedge \ P'\mathcal{R}Q'$
- if $\alpha = a(x)$ then $\exists Q'.\ Q \xrightarrow{a(x)} Q' \ \wedge \ \forall u.\ P'\{u/x\}\mathcal{R}Q'\{u/x\}$

Early Bisimulation: symmetric relation $\mathcal{R}$ s.t.: $P\mathcal{R}Q$ and $P \xrightarrow{\alpha} P'$ implies

- if α is not an input then $\exists Q'.\ Q \xrightarrow{\alpha} Q' \ \wedge \ P'\mathcal{R}Q'$
- if $\alpha = a(x)$ then $\forall u.\ \exists Q'.\ Q \xrightarrow{a(x)} Q' \ \wedge \ P'\{u/x\}\mathcal{R}Q'\{u/x\}$

Early, Late Bisimilarities are not a congruences w.r.t. input (substitution).

Late, Early Congruence: P, Q are late (early) congruent if for all substitution σ, $P\sigma$, $Q\sigma$ are late (early) bisimilar.

Open Bisimilarity

Open Bisimulation (on the fragment without restriction): symmetric relation $\mathcal{R}$ s.t., for all substitutions σ, $P\mathcal{R}Q$ and $P\sigma \xrightarrow{\alpha} P'$ implies

$$\exists Q'.\ Q\sigma \xrightarrow{\alpha} Q' \quad \text{and } P'\mathcal{R}Q'$$

Open Bisimilarity is a congruence.

Reduction Operational Semantics

- unlabelled transition system $\rightarrow$, capturing internal (τ) transitions
- assume all sums to be guarded, $\alpha_1.P_1 + \ldots + \alpha_n.P_n$

$$\text{R-PREFIX}\ \frac{}{\tau.P \rightarrow P}$$

$$\text{R-COM}\ \frac{}{(\ldots + a(x).P + \ldots) | (\ldots + \bar{a}u.Q + \ldots) \rightarrow P\{u/x\} | Q}$$

$$\text{R-PAR}\ \frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad \text{R-RES}\ \frac{P \rightarrow P'}{(\nu x)P \rightarrow (\nu x)P'}$$

$$\text{R-STRUCT}\ \frac{P' \equiv P \quad P \xrightarrow{\alpha} Q \quad Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$$

From π-calculus to Ambient Calculus

In the π-calculus:

- processes exist in a single contiguous location
- interaction is by shared names, used as I/O channels
- there is no direct account of access control

In the ambient calculus [Cardelli-Gordon98]:

- processes exist in multiple disjoint locations (ambients)
- interaction is by shared position, with no action at a distance
- capabilities, derived from ambient names, regulate access

Two overlapping views of mobility

- Mobile Computing.
  - I.e. mobile hardware (networks with a dynamic topology), physical mobility.
- Mobile Computation.
  - I.e. mobile software (executable code able to move around the network), virtual mobility.
- But the borders are fuzzy:
  - Agents may move traversing a network (virtually), or by being carried on a laptop (physically).
  - Computers may move by lugging them around (physically), or by telecontrol software (virtually).
  - Boundaries may be physical (buildings) or virtual (firewalls).

The ambient calculus provides a unified framework for modeling both kinds of mobility.
It is used to model interactions in concurrent systems as the Internet.

Ambients

- The fundamental primitive is the Ambient.
- An ambient is a place, delimited by a boundary, where computation happens.
- Ambients have a name, a collection of local processes, and a collection of subambients.
- Ambients can move in and out of other ambients, subject to capabilities that are associated with ambient names.
- Ambient names are unforgeable (as in π).

Examples of Ambients

- a web page (bounded by a file)
- a virtual address space (bounded by an addressing range)
- a Unix file system (bounded within a physical volume)
- a single data object (bounded by “self”)
- a laptop (bounded by its case and data ports)

Syntax of the Ambient Calculus

$P ::=$	$(\nu n)\ P$	new name n in a scope	scoping
	0	inactivity	standard in process calculi
	$P \mid P$	parallel	
	$!P$	replication	data structures
	$M[P]$	ambient	ambient-specific
	$M.P$	exercise a capability	
	$(n).P$	input locally, bind to n	actions
	$\langle M \rangle$	output locally (async)	ambient I/O
$M ::=$	n	name	basic capabilities
	in M	entry capability	
	out M	exit capability	
	open M	open capability	
	ε	empty path	useful with I/O
	$M.M'$	composite path	

Actions and Capabilities

- Operations that change the hierarchical structure of ambients are sensitive. They can be interpreted as the crossing of firewalls or the decoding of ciphertexts.
- Hence these operations are restricted by capabilities.

$$M.P$$

This executes an action regulated by the capability M, and then continues as the process P.

- The reduction rules for $M.P$ depend on M.

Ambient Names

- An ambient is written as follows, where n is the name of the ambient, and P is the process running inside it.

$$n[P]$$

- In $n[P]$, it is understood that P is actively running:

$$P \longrightarrow Q \implies n[P] \longrightarrow n[Q]$$

- Multiple ambients may have the same name, (e.g. replicated servers).

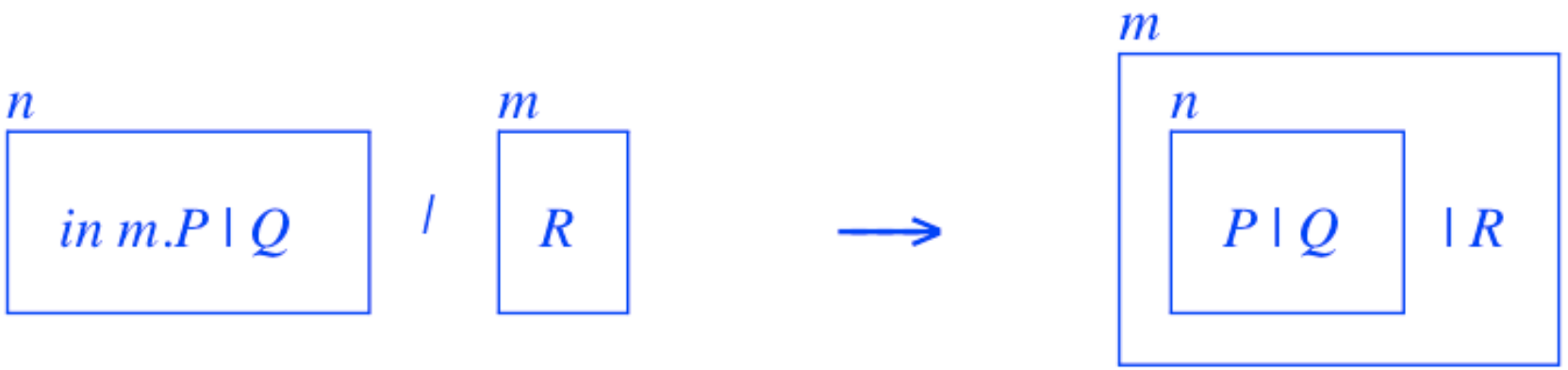
Entry Capability

- An entry capability, in m, can be used in the action:

$$\text{in } m.P$$

- The reduction rule (non-deterministic and blocking) is:

$$n[\text{in } m.P|Q]|m[R] \longrightarrow m[n[P|Q]|R]$$



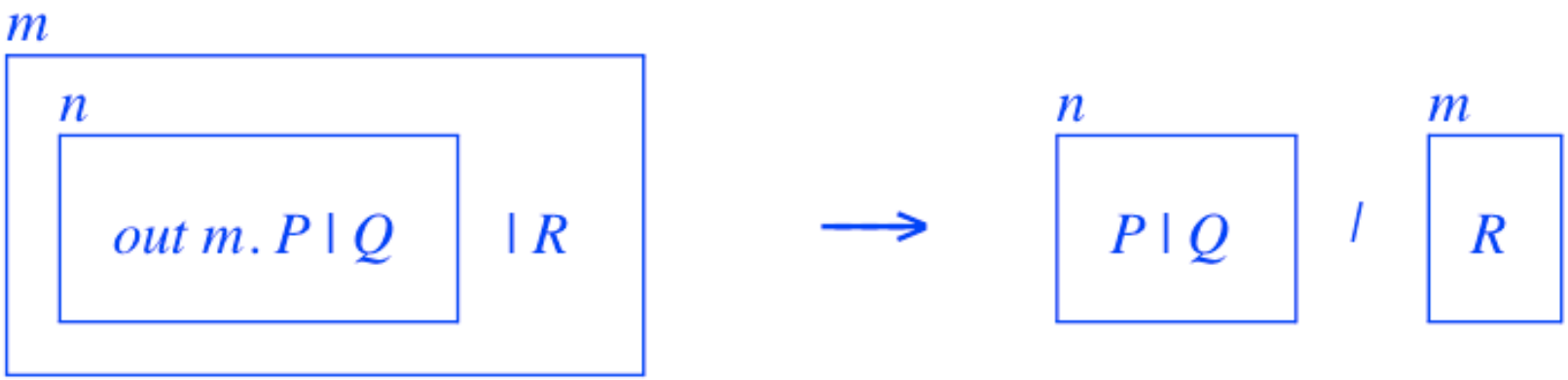
Exit Capability

- An exit capability, out m, can be used in the action:

$$\text{out } m.P$$

- The reduction rule (non-deterministic and blocking) is:

$$m[n[\text{out } m.P|Q]|R] \longrightarrow n[P|Q]|m[R]$$



Open Capability

- An opening capability, open m, can be used in the action:

$$\text{open } n.P$$

- The reduction rule (non-deterministic and blocking) is:

$$\text{open } n.P | n[Q] \longrightarrow P|Q$$

- An open operation may be upsetting both P and Q above.
  - From the point of view of P, there is no telling in general what Q might do when unleashed.
  - From the point of view of Q, its environment is being ripped open.
- Still, this operation is relatively well-behaved because:
  - The dissolution is initiated by the agent open $n.P$, so that the appearance of Q at the same level as P is not totally unexpected;
  - open n is a capability that is given out by n, so $n[Q]$ cannot be dissolved if it does not wish to be.

Ambient I/O

- Local anonymous communication within an ambient:

$(x).P$	input action
<M>	async output action

- We have the reduction

$$(x).P | < M > \longrightarrow P\{M/x\}$$

- This mechanism fits well with the ambient intuitions.
  - Long-range communication, like long-range movement, should not happen automatically because messages may have to cross firewalls and other obstacles.
  - Still, this is sufficient to emulate communication over named channels, etc.

Structural Congruence

$$P \equiv P \qquad \text{(Struct Refl)}$$

$$P \equiv Q \Rightarrow Q \equiv P \qquad \text{(Struct Symm)}$$

$$P \equiv Q, Q \equiv R \Rightarrow P \equiv R \qquad \text{(Struct Trans)}$$

$$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q \qquad \text{(Struct Res)}$$

$$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R \qquad \text{(Struct Par)}$$

$$P \equiv Q \Rightarrow !P \equiv !Q \qquad \text{(Struct Repl)}$$

$$P \equiv Q \Rightarrow M[P] \equiv M[Q] \qquad \text{(Struct Amb)}$$

$$P \equiv Q \Rightarrow M.P \equiv M.Q \qquad \text{(Struct Action)}$$

$$P \equiv Q \Rightarrow (n).P \equiv (n).Q \qquad \text{(Struct Input)}$$

$$P \mid Q \equiv Q \mid P \qquad \text{(Struct Par Comm)}$$

$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad \text{(Struct Par Assoc)}$$

$$!P \equiv P \mid !P \qquad \text{(Struct Repl Par)}$$

$$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \qquad \text{(Struct Res Res)}$$

$$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad \text{if } n \notin fn(P) \qquad \text{(Struct Res Par)}$$

$$(\nu n)(m[P]) \equiv m[(\nu n)P] \quad \text{if } n \neq m \qquad \text{(Struct Res Amb)}$$

$$P \mid \mathbf{0} \equiv P \qquad \text{(Struct Zero Par)}$$

$$(\nu n)\mathbf{0} \equiv \mathbf{0} \qquad \text{(Struct Zero Res)}$$

$$!\mathbf{0} \equiv \mathbf{0} \qquad \text{(Struct Zero Repl)}$$

$$\varepsilon.P \equiv P \qquad \text{(Struct } \varepsilon\text{)}$$

$$(M.M').P \equiv M.(M'.P) \qquad \text{(Struct .)}$$

Reduction Operational Semantics

$$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R] \qquad \text{(Red In)}$$

$$m[n[out\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R] \qquad \text{(Red Out)}$$

$$open\ n.\ P \mid n[Q] \longrightarrow P \mid Q \qquad \text{(Red Open)}$$

$$(n).\ P \mid \langle M \rangle \longrightarrow P\{n \leftarrow M\} \qquad \text{(Red Comm)}$$

$$P \longrightarrow Q \ \Rightarrow\ (\nu n)P \longrightarrow (\nu n)Q \qquad \text{(Red Res)}$$

$$P \longrightarrow Q \ \Rightarrow\ n[P] \longrightarrow n[Q] \qquad \text{(Red Amb)}$$

$$P \longrightarrow Q \ \Rightarrow\ P \mid R \longrightarrow Q \mid R \qquad \text{(Red Par)}$$

$$P' \equiv P, P \longrightarrow Q, Q \equiv Q' \ \Rightarrow\ P' \longrightarrow Q' \qquad \text{(Red } \equiv\text{)}$$

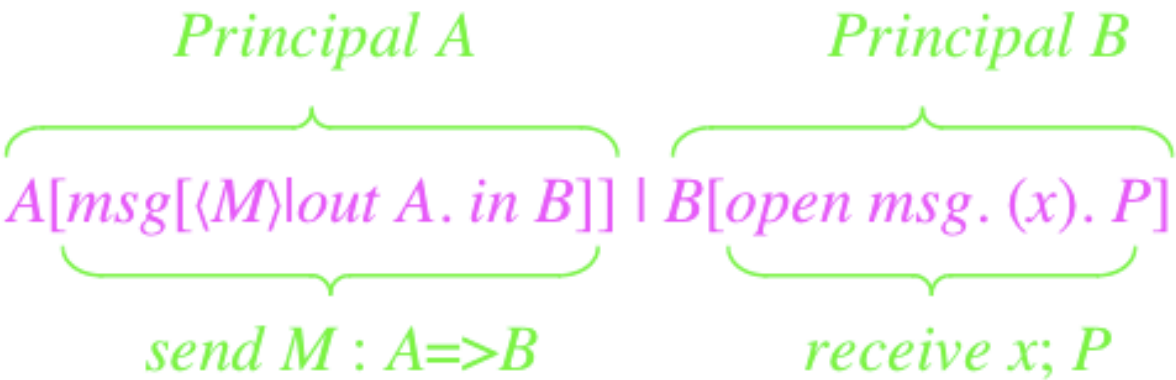
$\longrightarrow^*$ reflexive and transitive closure of $\longrightarrow$

In addition, we identify terms up to renaming of bound names:

$$(\nu n)P = (\nu m)P\{n \leftarrow m\} \quad \text{if } m \notin fn(P)$$

$$(n).P = (m).P\{n \leftarrow m\} \quad \text{if } m \notin fn(P)$$

Example



$$
\begin{aligned}
&A[msg[\langle M\rangle|out\ A.\ in\ B]] \mid B[open\ msg.\ (x).\ P] \\
&\longrightarrow A[] \mid msg[\langle M\rangle|in\ B] \mid B[open\ msg.\ (x).\ P] \\
&\longrightarrow A[] \mid B[msg[\langle M\rangle]|open\ msg.\ (x).\ P] \\
&\longrightarrow A[] \mid B[\langle M\rangle|(x).\ P] \\
&\longrightarrow A[] \mid B[P\{x \leftarrow M\}]
\end{aligned}
$$

Contextual Equivalence

- A standard semantic equivalence: $P \simeq Q$ iff the observable behavior of a system with component P is unaffected by replacing P with Q.
- In the ambient calculus:
  - $P\downarrow n$ means that process P exhibits n (observable by *open* n).

$$P\downarrow n \;\Leftrightarrow\; P \equiv (\nu n_1 ... n_p)(n[Q] \mid R) \;\wedge\; n \notin \{n_1 ... n_p\}$$

  - $P\Downarrow n$ means that process P converges n, i.e., it evolves into a process that exhibits n.

$$P\Downarrow n \;\Leftrightarrow\; P \longrightarrow^* Q \;\wedge\; Q\downarrow n$$

  - A *context* $C(\bullet)$ is a process with a hole. Ex.: $\bullet$, $n[\bullet]$, $in\ n.\bullet$, $!\bullet$.
  - Contextual Equivalence:

$$P \simeq Q \;\Leftrightarrow\; \forall n.\, \forall C(\bullet).\, C(P)\Downarrow n \Leftrightarrow C(Q)\Downarrow n$$

Basic References

- CCS:
  - R. Milner. *Communication and Concurrency*, Prentice Hall, 1989.
  - L. Aceto, A. Ingolfsdottir, K. Larsen, J. Srba. *Reactive Systems*, Cambridge University Press, 2007.
- π-calculus:
  - R. Milner. *Communicating and Mobile Systems: the π-calculus*, Cambridge University Press, 1999.
  - J. Parrow. An Introduction to the π-calculus, Chapter in *Handbook of Process Algebra*, Bergstra et al eds., Elsevier, 2001.
  - D. Sangiorgi, D. Walker. *The π-calculus: a Theory of Mobile Processes*, Cambridge University Press, 2001.
- Ambient Calculus:
  - L. Cardelli, A. Gordon. Mobile Ambients, *Proc. of FoSSaCS'98*, M. Nivat ed., LNCS 1378, 1998.
  - See also the web page of L. Cardelli, http://lucacardelli.name/

Further Reading

- P. Di Gianantonio, F. Honsell, M. Lenisa, *RPO, second-order contexts, and the lambda-calculus*, Logical Methods in Computer Science, to apper.
- P. Di Gianantonio, F. Honsell, M. Lenisa, *Finitely Branching Labelled Transition Systems from Reaction Semantics for Process Calculi*, WADT'08, to appear.
- R. Milner, J. Leifer, *Deriving Bisimulation Congruences for Reactive Systems*, CONCUR'00, LNCS 1877, 2000.
- M. Merro, F. Zappa Nardelli, *Behavioral theory for mobile ambients*, Journal of the ACM 52(6), 2005.
- J. Rathke, P. Sobocinski, *Deriving structural labelled transitions for mobile ambients*, LNCS 5201, 2008.
- D. Sangiorgi, *On the bisimulation proof method*, Journal of MSCS, 8, 1998.