Models and Semantics of Computation

Jorge A. Pérez
Bernoulli Institute
University of Groningen

October 19, 2023

The π-calculus: A Calculus of Mobile Processes (II)

- operational semantics (LTS, reduction semantics)
- bisimilarity and congruence
- subcalculi and extensions

An LTS for the π-calculus

- We would like a semantics for the π-calculus in the SOS style, just as we did for CCS.
- In CCS, actions were based on fairly simple prefixes.
  In the π-calculus, prefixes have a bit more of structure.
  - What should be the actions for the π-calculus?
  - How do we represent name passing? And scope extrusion?
- Mobility (i.e., name passing) brings in a number of issues to the definition of the SOS semantics (and derived bisimilarities)
- In fact, there will be several possible LTSs for the π-calculus

An LTS for the π-calculus

We consider four kinds of actions:

1. Internal actions, τ
2. Output actions, $\overline{a}\langle x\rangle$
3. Input actions, $a(x)$
4. Bound output actions, written $\overline{a}\langle \nu x\rangle$, represent the output of a local name

Now prefixes do not directly correspond to actions (in the sense that there are two actions for outputs).

An LTS for the π-calculus

We consider four kinds of actions:

1. Internal actions, τ
2. Output actions, $\overline{a}\langle x \rangle$
3. Input actions, $a(x)$
4. Bound output actions, written $\overline{a}\langle \nu x \rangle$, represent the output of a local name

Now prefixes do not directly correspond to actions (in the sense that there are two actions for outputs).

An LTS for the π-calculus

We consider four kinds of actions:

1. Internal actions, τ
2. Output actions, $\overline{a}\langle x\rangle$
3. Input actions, $a(x)$
4. Bound output actions, written $\overline{a}\langle \nu x\rangle$, represent the output of a local name

Now prefixes do not directly correspond to actions (in the sense that there are two actions for outputs).

An LTS for the π-calculus

Notions of free and bound names now also hold for actions:

α	kind	subj(α)	obj(α)	fn(α)	bn(α)	n(α)
$\overline{a}\langle x\rangle$	free output	a	x	$\{a, x\}$	$\emptyset$	$\{a, x\}$
$a(y)$	input	a	y	$\{a, y\}$	$\emptyset$	$\{a, y\}$
$\overline{a}\langle \nu x\rangle$	bound output	a	x	$\{a\}$	$\{x\}$	$\{a, z\}$
τ	internal	—	—	$\emptyset$	$\emptyset$	$\emptyset$

Above, we have:

- subj(α) stands for the subject of α
- obj(α) stands for the object of α
- fn(α) stands for the free names of α
- bn(α) stands for the bound names of α
- n(α) stands for the all names in α

An LTS for the π-calculus

At least two important subtleties in defining an LTS for the π-calculus.

The first concerns the meaning of input transitions.
Given $P = a(x).Q$, we expect P to have an input transition to be of the form

$$P \xrightarrow{a(x)} Q$$

meaning that P can receive a certain (unknown) name u on a and then evolve to $Q\{^u/_x\}$.

The subtlety is in when we actually record the reception of u at a.
That is, when we record the associated name substitution.

An LTS for the π-calculus

At least two important subtleties in defining an LTS for the π-calculus.

The first concerns the meaning of input transitions.
Given $P = a(x).Q$, we expect P to have an input transition to be of the form

$$P \xrightarrow{a(x)} Q$$

meaning that P can receive a certain (unknown) name u on a and then evolve to $Q\{^{u}/_{x}\}$.

The subtlety is in when we actually record the reception of u at a.
That is, when we record the associated name substitution.

An LTS for the π-calculus: Design Decisions

When do we record the substitution? Two possibilities:

1. Early style:
   The substitution is recorded as soon as the input is exercised
2. Late style:
   The substitution is recorded when the received name u needs to be substituted

[This difference is relevant from a technical point of view. We won't go into those technicalities.]

An LTS for the π-calculus: Design Decisions

When do we record the substitution? Two possibilities:

1. Early style:
   The substitution is recorded as soon as the input is exercised
2. Late style:
   The substitution is recorded when the received name u needs to be substituted

[This difference is relevant from a technical point of view. We won't go into those technicalities.]

An LTS for the π-calculus: Design Decisions

The second subtlety is how to represent scope extrusion.

This depends on the desired definition of structural congruence.
Here again, there are at least two possibilities:

1. Incorporate structural congruence as a rule of the LTS, and model scope extrusion with the laws for restriction.
2. Use dedicated rules modeling the "opening" and "closure" of the scope of a private name.

An LTS for the π-calculus

We present a possible LTS for the π-calculus

- The early LTS with scope extrusion handled by dedicated rules
- There are tradeoffs concerning what should be treated by $\equiv$ and what should be handled by the LTS rules.
  Examples: recursive definitions, replication, commutativity laws.
- Typically LTSs do not feature $\equiv$ and require only α-conversion
- Those differences are usually a matter of convenience for theoretical and practical developments.

An LTS for the π-calculus

$$\text{OUT}\ \frac{}{\overline{a}\langle v\rangle.P \xrightarrow{\overline{a}\langle v\rangle} P} \qquad \text{INP}\ \frac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{SUM}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

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

$$\text{PAR}\ \frac{P \xrightarrow{\alpha} P' \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$$

$$\text{COM}\ \frac{P \xrightarrow{a(u)} P' \quad Q \xrightarrow{\overline{a}\langle u\rangle} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad \text{RES}\ \frac{P \xrightarrow{\alpha} P' \quad x \notin \mathrm{n}(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$$

$$\text{OPEN}\ \frac{P \xrightarrow{\overline{a}\langle x\rangle} P' \quad a \neq x}{(\nu x)P \xrightarrow{\overline{a}\langle \nu x\rangle} P'} \qquad \text{CLOSE}\ \frac{P \xrightarrow{\overline{a}\langle \nu x\rangle} P' \quad Q \xrightarrow{a(x)} Q' \quad x \notin \mathrm{fn}(Q)}{P \parallel Q \xrightarrow{\tau} (\nu x)(P' \parallel Q')}$$

An LTS for the π-calculus

$$\text{OUT}\ \frac{}{\overline{a}\langle v\rangle.P \xrightarrow{\overline{a}\langle v\rangle} P} \qquad \text{INP}\ \frac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{SUM}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

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

$$\text{PAR}\ \frac{P \xrightarrow{\alpha} P' \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$$

$$\text{COM}\ \frac{P \xrightarrow{a(u)} P' \quad Q \xrightarrow{\overline{a}\langle u\rangle} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad \text{RES}\ \frac{P \xrightarrow{\alpha} P' \quad x \notin \mathrm{n}(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$$

$$\text{OPEN}\ \frac{P \xrightarrow{\overline{a}\langle x\rangle} P' \quad a \neq x}{(\nu x)P \xrightarrow{\overline{a}\langle \nu x\rangle} P'} \qquad \text{CLOSE}\ \frac{P \xrightarrow{\overline{a}\langle \nu x\rangle} P' \quad Q \xrightarrow{a(x)} Q' \quad x \notin \mathrm{fn}(Q)}{P \parallel Q \xrightarrow{\tau} (\nu x)(P' \parallel Q')}$$

An LTS for the π-calculus

$$\text{OUT}\ \frac{}{\overline{a}\langle v\rangle.P \xrightarrow{\overline{a}\langle v\rangle} P} \qquad \text{INP}\ \frac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{SUM}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

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

$$\text{PAR}\ \frac{P \xrightarrow{\alpha} P' \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$$

$$\text{COM}\ \frac{P \xrightarrow{a(u)} P' \quad Q \xrightarrow{\overline{a}\langle u\rangle} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad \text{RES}\ \frac{P \xrightarrow{\alpha} P' \quad x \notin \mathrm{n}(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$$

$$\text{OPEN}\ \frac{P \xrightarrow{\overline{a}\langle x\rangle} P' \quad a \neq x}{(\nu x)P \xrightarrow{\overline{a}\langle \nu x\rangle} P'} \qquad \text{CLOSE}\ \frac{P \xrightarrow{\overline{a}\langle \nu x\rangle} P' \quad Q \xrightarrow{a(x)} Q' \quad x \notin \mathrm{fn}(Q)}{P \parallel Q \xrightarrow{\tau} (\nu x)(P' \parallel Q')}$$

An LTS for the π-calculus

$$\text{OUT}\ \frac{}{\overline{a}\langle v\rangle.P \xrightarrow{\overline{a}\langle v\rangle} P} \qquad \text{INP}\ \frac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{SUM}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

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

$$\text{PAR}\ \frac{P \xrightarrow{\alpha} P' \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$$

$$\text{COM}\ \frac{P \xrightarrow{a(u)} P' \quad Q \xrightarrow{\overline{a}\langle u\rangle} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad \text{RES}\ \frac{P \xrightarrow{\alpha} P' \quad x \notin \mathrm{n}(\alpha)}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$$

$$\text{OPEN}\ \frac{P \xrightarrow{\overline{a}\langle x\rangle} P' \quad a \neq x}{(\nu x)P \xrightarrow{\overline{a}\langle \nu x\rangle} P'} \qquad \text{CLOSE}\ \frac{P \xrightarrow{\overline{a}\langle \nu x\rangle} P' \quad Q \xrightarrow{a(x)} Q' \quad x \notin \mathrm{fn}(Q)}{P \parallel Q \xrightarrow{\tau} (\nu x)(P' \parallel Q')}$$

An LTS for the π-calculus

- Rule STR avoids the need for dedicated rules for recursive definitions, and the commutativity of sum and parallel. (Clearly, reduced variants of $\equiv$ are possible.)
- How do we handle name communication?

$$\text{COM}\ \frac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{a(x).P \parallel \overline{a}\langle u\rangle.Q \xrightarrow{\tau} P\{u/x\} \parallel Q}$$

- How do we handle scope extrusion?

$$\text{CLOSE}\ \frac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{OPEN}\ \dfrac{\text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{(\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle \nu u\rangle} Q}}{a(x).P \parallel (\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\tau} (\nu u)(P\{u/x\} \parallel Q)}$$

- Question: What would be the situation in the early style?

An LTS for the π-calculus

- Rule STR avoids the need for dedicated rules for recursive definitions, and the commutativity of sum and parallel. (Clearly, reduced variants of $\equiv$ are possible.)
- How do we handle name communication?

$$\text{COM}\ \dfrac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{a(x).P \parallel \overline{a}\langle u\rangle.Q \xrightarrow{\tau} P\{u/x\} \parallel Q}$$

- How do we handle scope extrusion?

$$\text{CLOSE}\ \dfrac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{OPEN}\ \dfrac{\text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{(\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle \nu u\rangle} Q}}{a(x).P \parallel (\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\tau} (\nu u)(P\{u/x\} \parallel Q)}$$

- Question: What would be the situation in the early style?

An LTS for the π-calculus

- Rule STR avoids the need for dedicated rules for recursive definitions, and the commutativity of sum and parallel. (Clearly, reduced variants of $\equiv$ are possible.)
- How do we handle name communication?

$$\text{COM}\ \dfrac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{^u/_x\}} \qquad \text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{a(x).P \parallel \overline{a}\langle u\rangle.Q \xrightarrow{\tau} P\{^u/_x\} \parallel Q}$$

- How do we handle scope extrusion?

$$\text{CLOSE}\ \dfrac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{^u/_x\}} \qquad \text{OPEN}\ \dfrac{\text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{(\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle \nu u\rangle} Q}}{a(x).P \parallel (\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\tau} (\nu u)(P\{^u/_x\} \parallel Q)}$$

- Question: What would be the situation in the early style?

An LTS for the π-calculus

- Rule STR avoids the need for dedicated rules for recursive definitions, and the commutativity of sum and parallel. (Clearly, reduced variants of $\equiv$ are possible.)
- How do we handle name communication?

$$\text{COM}\ \dfrac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{a(x).P \parallel \overline{a}\langle u\rangle.Q \xrightarrow{\tau} P\{u/x\} \parallel Q}$$

- How do we handle scope extrusion?

$$\text{CLOSE}\ \dfrac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{u/x\}} \qquad \text{OPEN}\ \dfrac{\text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{(\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle \nu u\rangle} Q}}{a(x).P \parallel (\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\tau} (\nu u)(P\{u/x\} \parallel Q)}$$

- Question: What would be the situation in the early style?

An LTS for the π-calculus

- Rule STR avoids the need for dedicated rules for recursive definitions, and the commutativity of sum and parallel. (Clearly, reduced variants of $\equiv$ are possible.)
- How do we handle name communication?

$$\text{COM}\ \frac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{{}^{u}\!/\!_{x}\}} \qquad \text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{a(x).P \parallel \overline{a}\langle u\rangle.Q \xrightarrow{\tau} P\{{}^{u}\!/\!_{x}\} \parallel Q}$$

- How do we handle scope extrusion?

$$\text{CLOSE}\ \frac{\text{INP}\ \dfrac{}{a(x).P \xrightarrow{a(u)} P\{{}^{u}\!/\!_{x}\}} \qquad \text{OPEN}\ \dfrac{\text{OUT}\ \dfrac{}{\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle u\rangle} Q}}{(\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\overline{a}\langle \nu u\rangle} Q}}{a(x).P \parallel (\nu u)\overline{a}\langle u\rangle.Q \xrightarrow{\tau} (\nu u)(P\{{}^{u}\!/\!_{x}\} \parallel Q)}$$

- Question: What would be the situation in the early style?

Reduction Semantics

- A different approach to the semantics of the π-calculus is to define a reduction semantics.
- The main merit of a reduction semantics is that it offers a succinct presentation of a process' behavior

Reduction Semantics

- Intuitively, the reduction semantics focuses on the internal behavior of a process, rather than on the interaction between the process and its environment.
- Hence, relation $P \longmapsto Q$ ("P reduces to Q") should be the same as $P \xrightarrow{\tau} Q$.
  (Often $\longmapsto$ is defined simply as $\xrightarrow{\tau}$.)
- Reductions are inferred in a slightly different way from transitions.

Reduction Semantics

The reduction relation, denoted $\longmapsto$, is defined by the following rules:

$$(a(x).P + P') \parallel (\overline{a}\langle v\rangle.Q + Q') \longmapsto P\{^v/_x\} \parallel Q$$

$$\frac{P \longmapsto P'}{P \parallel Q \longmapsto P' \parallel Q} \qquad \frac{P \longmapsto P'}{(\nu x)\, P \longmapsto (\nu x)\, P'} \qquad \frac{P \equiv P' \longmapsto Q' \equiv Q}{P \longmapsto Q}$$

Observe:

- Hence, $\equiv$ can occur at any point in the inference.
  It promotes behavior, by bringing together processes.
- This works for guarded choices i.e., sums of the form $\alpha_1.P_1 + \cdots + \alpha_n.P_n$.
- It should be possible to show that $\longmapsto$ and $\xrightarrow{\tau} \circ \equiv$ coincide.

Bisimilarities and Congruences in the π-calculus

- Probably the most striking differences between CCS and the π-calculus arise in the behavioral theory
- The early vs. late issue (in general, the issue of handling substitutions) also carries over to the definition of bisimilarities
- Also, natural notions of bisimilarity are not a congruence (i.e., they do not respect the constructs of the language)
- Next, we overview some of these difficulties

Strong Bisimilarity

Actions with bound names need to be handled carefully:

- Consider the processes $P = a(u)$ and $Q = a(x).(\nu v)\overline{v}\langle u \rangle$
- P and Q represent the same behavior:
  they receive a name on a and then they do nothing
- Name u is free in Q. Now, if P moves, then Q cannot match this action, as x cannot be α-converted to u in Q
- Of course, u in P can be α-converted and Q can match this different action

If we wish to relate two processes P and Q, and $P \xrightarrow{a(x)} P'$ then we need $Q \xrightarrow{a(x)} Q'$, keeping in mind that P' and Q' should be related for every possible received value (i.e., any possible substitution)

Strong Bisimilarity

Actions with bound names need to be handled carefully:

- Consider the processes $P = a(u)$ and $Q = a(x).(\nu v)\overline{v}\langle u \rangle$
- P and Q represent the same behavior:
  they receive a name on a and then they do nothing
- Name u is free in Q. Now, if P moves, then Q cannot match this action, as x cannot be α-converted to u in Q
- Of course, u in P can be α-converted and Q can match this different action

If we wish to relate two processes P and Q, and $P \xrightarrow{a(x)} P'$ then we need $Q \xrightarrow{a(x)} Q'$, keeping in mind that P' and Q' should be related for every possible received value (i.e., any possible substitution)

Strong Bisimilarity

Actions with bound names need to be handled carefully:

- Consider the processes $P = a(u)$ and $Q = a(x).(\nu v)\overline{v}\langle u\rangle$
- P and Q represent the same behavior:
  they receive a name on a and then they do nothing
- Name u is free in Q. Now, if P moves, then Q cannot match this action, as x cannot be α-converted to u in Q
- Of course, u in P can be α-converted and Q can match this different action

If we wish to relate two processes P and Q, and $P \xrightarrow{a(x)} P'$ then we need $Q \xrightarrow{a(x)} Q'$, keeping in mind that P' and Q' should be related for every possible received value (i.e., any possible substitution)

Strong Bisimilarity

We write "bn(α) is fresh" to mean that any name in bn(α) is different from any other free name.

Strong bisimulation

A strong bisimulation is a symmetric binary relation $\mathcal{R}$ satisfying the following: $P \mathcal{R} Q$ and $P \xrightarrow{\alpha} P'$, where bn($\alpha$) is fresh implies:

1. If α is not an input, then $\exists Q'.\ Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{R} Q'$
2. If $\alpha = a(x)$ then $\exists Q'.\ Q \xrightarrow{a(x)} Q'$ and $\forall u.\ P'\{^u/_x\} \mathcal{R} Q'\{^u/_x\}$

Strong bisimilarity

Two processes P and Q are strongly bisimilar, written $P \sim Q$, if there exists a strong bisimilarity $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

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

Strong Bisimilarity

We write "bn(α) is fresh" to mean that any name in bn(α) is different from any other free name.

Strong bisimulation

A strong bisimulation is a symmetric binary relation $\mathcal{R}$ satisfying the following: $P \mathcal{R} Q$ and $P \xrightarrow{\alpha} P'$, where bn($\alpha$) is fresh implies:

1. If α is not an input, then $\exists Q'.\ Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{R} Q'$
2. If $\alpha = a(x)$ then $\exists Q'.\ Q \xrightarrow{a(x)} Q'$ and $\forall u.\ P'\{u/x\} \mathcal{R} Q'\{u/x\}$

Strong bisimilarity

Two processes P and Q are strongly bisimilar, written $P \sim Q$, if there exists a strong bisimilarity $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

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

Strong Bisimilarity: Example

A small example, using the matching construct

$$\texttt{if}\ x = u\ \texttt{then}\ P$$

which executes P if x is u and does nothing otherwise.

Consider the processes

$$P_1 = a(x).P + a(x).\mathbf{0} \qquad P_2 = a(x).P + a(x).\texttt{if}\ x = u\ \texttt{then}\ P$$

with $P \not\sim \mathbf{0}$. We have that $P_1 \not\sim P_2$ because the transition

$$P_1 \xrightarrow{a(x)} \mathbf{0}$$

cannot be simulated by P_2.

In fact, even if $P_2 \xrightarrow{a(x)} \texttt{if}\ x = u\ \texttt{then}\ P$, in case the substitution $\{^u/_x\}$ is used, the derivatives ($\mathbf{0}$ and P) are not bisimilar.

Strong Bisimilarity: Example

A small example, using the matching construct

$$\mathtt{if}\ x = u\ \mathtt{then}\ P$$

which executes P if x is u and does nothing otherwise.

Consider the processes

$$P_1 = a(x).P + a(x).\mathbf{0} \qquad P_2 = a(x).P + a(x).\mathtt{if}\ x = u\ \mathtt{then}\ P$$

with $P \not\sim \mathbf{0}$. We have that $P_1 \not\sim P_2$ because the transition

$$P_1 \xrightarrow{a(x)} \mathbf{0}$$

cannot be simulated by P_2.

In fact, even if $P_2 \xrightarrow{a(x)} \mathtt{if}\ x = u\ \mathtt{then}\ P$, in case the substitution $\{{}^{u}\!/\!_{x}\}$ is used, the derivatives ($\mathbf{0}$ and P) are not bisimilar.

Strong Bisimilarity: Example

A small example, using the matching construct

$$\texttt{if}\ x = u\ \texttt{then}\ P$$

which executes P if x is u and does nothing otherwise.

Consider the processes

$$P_1 = a(x).P + a(x).\mathbf{0} \qquad P_2 = a(x).P + a(x).\texttt{if}\ x = u\ \texttt{then}\ P$$

with $P \not\sim \mathbf{0}$. We have that $P_1 \not\sim P_2$ because the transition

$$P_1 \xrightarrow{a(x)} \mathbf{0}$$

cannot be simulated by P_2.

In fact, even if $P_2 \xrightarrow{a(x)} \texttt{if}\ x = u\ \texttt{then}\ P$, in case the substitution $\{u/x\}$ is used, the derivatives ($\mathbf{0}$ and P) are not bisimilar.

Strong Bisimilarity: Properties

Unsurprisingly, we have

$$a \parallel \overline{b} \sim a.\overline{b} \parallel \overline{b}.a$$

However, strong bisimilarity is not closed under arbitrary substitutions σ, i.e., $P \sim Q$ does not imply $P\sigma \sim Q\sigma$.

For instance, we have

$$c(a).a \parallel \overline{b} \sim c(a).a.\overline{b} \parallel \overline{b}.a$$

which means that $\sim$ is not preserved by input prefix. That is, if $P \sim Q$ it is not necessarily the case that $a(x).P \sim a(x).Q$.

Properties of $\sim$

- If $P \sim Q$ and σ is an injective substitution then $P\sigma \sim Q\sigma$
- $\sim$ is an equivalence
- $\sim$ is preserved by all operators but input prefix

Strong Bisimilarity: Properties

Unsurprisingly, we have

$$a \parallel \overline{b} \sim a.\overline{b} \parallel \overline{b}.a$$

However, strong bisimilarity is not closed under arbitrary substitutions σ, i.e., $P \sim Q$ does not imply $P\sigma \sim Q\sigma$.

For instance, we have

$$c(a).a \parallel \overline{b} \sim c(a).a.\overline{b} \parallel \overline{b}.a$$

which means that $\sim$ is not preserved by input prefix. That is, if $P \sim Q$ it is not necessarily the case that $a(x).P \sim a(x).Q$.

Properties of $\sim$

- If $P \sim Q$ and σ is an injective substitution then $P\sigma \sim Q\sigma$
- $\sim$ is an equivalence
- $\sim$ is preserved by all operators but input prefix

Strong Bisimilarity: Properties

Unsurprisingly, we have

$$a \parallel \overline{b} \sim a.\overline{b} \parallel \overline{b}.a$$

However, strong bisimilarity is not closed under arbitrary substitutions σ, i.e., $P \sim Q$ does not imply $P\sigma \sim Q\sigma$.

For instance, we have

$$c(a).a \parallel \overline{b} \sim c(a).a.\overline{b} \parallel \overline{b}.a$$

which means that $\sim$ is not preserved by input prefix. That is, if $P \sim Q$ it is not necessarily the case that $a(x).P \sim a(x).Q$.

Properties of $\sim$

- If $P \sim Q$ and σ is an injective substitution then $P\sigma \sim Q\sigma$
- $\sim$ is an equivalence
- $\sim$ is preserved by all operators but input prefix

Strong Bisimilarity: Properties

Unsurprisingly, we have

$$a \parallel \overline{b} \sim a.\overline{b} \parallel \overline{b}.a$$

However, strong bisimilarity is not closed under arbitrary substitutions σ, i.e., $P \sim Q$ does not imply $P\sigma \sim Q\sigma$.

For instance, we have

$$c(a).a \parallel \overline{b} \sim c(a).a.\overline{b} \parallel \overline{b}.a$$

which means that $\sim$ is not preserved by input prefix. That is, if $P \sim Q$ it is not necessarily the case that $a(x).P \sim a(x).Q$.

Properties of $\sim$

- If $P \sim Q$ and σ is an injective substitution then $P\sigma \sim Q\sigma$
- $\sim$ is an equivalence
- $\sim$ is preserved by all operators but input prefix

Congruence

- Recall: At the end of the day, we want behavioral equivalences that induce congruences: equivalences preserved by all the operators of the language
- Such congruences allow to "replace" components in a larger context, i.e., if $P \cong Q$ then $C(P) \cong C(Q)$, for any context C
- As $\sim$ is not preserved by input prefixes, the best we can get is the following:

Congruence

Processes P and Q are strongly congruent, written $P \cong Q$, if $P\sigma \sim Q\sigma$, for all substitutions σ.

- This is the largest congruence in bisimilarity

Congruence

- Recall: At the end of the day, we want behavioral equivalences that induce congruences: equivalences preserved by all the operators of the language
- Such congruences allow to "replace" components in a larger context, i.e., if $P \cong Q$ then $C(P) \cong C(Q)$, for any context C
- As $\sim$ is not preserved by input prefixes, the best we can get is the following:

Congruence

Processes P and Q are strongly congruent, written $P \cong Q$, if $P\sigma \sim Q\sigma$, for all substitutions σ.

- This is the largest congruence in bisimilarity

Congruence

- Recall: At the end of the day, we want behavioral equivalences that induce congruences: equivalences preserved by all the operators of the language
- Such congruences allow to "replace" components in a larger context, i.e., if $P \cong Q$ then $C(P) \cong C(Q)$, for any context C
- As $\sim$ is not preserved by input prefixes, the best we can get is the following:

Congruence

Processes P and Q are strongly congruent, written $P \cong Q$, if $P\sigma \sim Q\sigma$, for all substitutions σ.

- This is the largest congruence in bisimilarity

Barbed Bisimilarity and Barbed Congruence

- Bisimilarity is unrealistically powerful: it distinguishes $\overline{a}\langle u \rangle$ from $\overline{a}\langle \nu u \rangle$, thus detecting if a name is local
- In barbed bisimulation we reduce this discriminatory power, and restrict to observing actions on a given channel.
- An observability predicate: We say that P has a barb on a, written $P \downarrow a$, if P has some action on name a

Barbed Bisimilarity and Barbed Congruence

Barbed Bisimulation

A barbed bisimulation is a symmetric binary relation $\mathcal{R}$ satisfying the following: $P \mathcal{R} Q$ implies:

1. If $P \xrightarrow{\tau} P'$ then $\exists Q'.\ Q \xrightarrow{\tau} Q'$ and $P' \mathcal{R} Q'$
2. If $P \downarrow a$ then $Q \downarrow a$

Barbed Bisimilarity

Two processes P and Q are barbed bisimilar, written $P \sim_B Q$, if there exists a barbed bisimilarity $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

Barbed Bisimilarity and Barbed Congruence

- Barbed bisimilarity by itself is uninteresting:

$$\overline{a}\langle u\rangle \sim_B \overline{a}\langle v\rangle \sim_B (\nu u)\overline{a}\langle u\rangle$$

- Again, our objective is the induced congruence. We thus have:

Barbed Congruence

Processes P and Q are barbed congruent, written $P \cong_B Q$, if for all process contexts C it holds that $C(P) \sim_B C(Q)$.

- Hence, $\cong_B$ is the largest congruence in $\sim_B$. It distinguishes the three processes above. For instance, consider the context

$$C(P) = P \parallel a(x).\overline{x}$$

More on Barbed Congruence

Barbed congruence provides another path (somewhat more direct) to characterizing process congruences

- Rather than relying on an LTS, it relies on reduction and an observability predicate
- This is very appealing, as it allows comparisons among different calculi (provided they have compatible observability predicates, which is often the case)
- Also useful when an LTS is difficult to define or inconvenient to have

In fact, barbed congruence is the concurrent version of what is called contextual equivalence in (typed) functional languages.

More on Barbed Congruence

Barbed congruence provides another path (somewhat more direct) to characterizing process congruences

- Rather than relying on an LTS, it relies on reduction and an observability predicate
- This is very appealing, as it allows comparisons among different calculi (provided they have compatible observability predicates, which is often the case)
- Also useful when an LTS is difficult to define or inconvenient to have

In fact, barbed congruence is the concurrent version of what is called contextual equivalence in (typed) functional languages.

More on Barbed Congruence

Usually, a behavioral theory for a given calculus is composed of a barbed congruence (call it $\cong$) which is characterized by a labelled bisimilarity (call it $\sim$). One wishes to show:

- Soundness: if $P \sim Q$ then $P \cong Q$
- Completeness if $P \cong Q$ then $P \sim Q$

Question: what is the purpose of this kind of characterization?

More on Barbed Congruence

Usually, a behavioral theory for a given calculus is composed of a barbed congruence (call it $\cong$) which is characterized by a labelled bisimilarity (call it $\sim$). One wishes to show:

- Soundness: if $P \sim Q$ then $P \cong Q$
- Completeness if $P \cong Q$ then $P \sim Q$

Question: what is the purpose of this kind of characterization?

Subcalculi and Extensions of the π-calculus

- In the mid 90s, the π-calculus gave a tremendous momentum to research in process calculi, and concurrency theory
- Many fragments, extensions, and variations of the basic language were proposed.
- There were at least two major motivations:
  1. Understanding better the fundamentals of the calculus
     $\rightarrow$ Subcalculi with interesting properties
  2. Exploring novel applications (security, systems biology, distributed and mobile computing)
     $\rightarrow$ Minimal extensions tailored to some specific domain
- Only a few such proposals passed the test of time...

Subcalculi and Extensions of the π-calculus

- In the mid 90s, the π-calculus gave a tremendous momentum to research in process calculi, and concurrency theory
- Many fragments, extensions, and variations of the basic language were proposed.
- There were at least two major motivations:
  1. Understanding better the fundamentals of the calculus
     $\rightarrow$ Subcalculi with interesting properties
  2. Exploring novel applications (security, systems biology, distributed and mobile computing)
     $\rightarrow$ Minimal extensions tailored to some specific domain
- Only a few such proposals passed the test of time...

Subcalculi and Extensions of the π-calculus

I will tell you briefly about two of such proposals:

1. The asynchronous π-calculus:
   the fragment of π in which output actions are non blocking.
2. The private π-calculus:
   the fragment of π in which only local names can be exchanged.

Asynchronous π-calculus

Both CCS and the π-calculus express synchronous communication.

- Prefixes enforce temporal precedence.
- Therefore, in interactions, the act of sending must occur together with some reception elsewhere.

In practice, however, communication is asynchronous.

- The act of sending a message is separate from the act of receiving it.
- Communication is an interaction mediated by some medium, such as a buffer or a queue.
- Mediums have different characteristics and forms, and may induce delays

Asynchronous π-calculus

Both CCS and the π-calculus express synchronous communication.

- Prefixes enforce temporal precedence.
- Therefore, in interactions, the act of sending must occur together with some reception elsewhere.

In practice, however, communication is asynchronous.

- The act of sending a message is separate from the act of receiving it.
- Communication is an interaction mediated by some medium, such as a buffer or a queue.
- Mediums have different characteristics and forms, and may induce delays

Asynchronous π-calculus

- The asynchronous π-calculus results from disallowing continuations after an output prefix
- The only possible continuation for an output prefix is thus $\mathbf{0}$. We have "output particles" of the form $\overline{a}\langle v\rangle.\mathbf{0}$, noted $\overline{a}\langle v\rangle$.
- Sending a message means leaving such particles unguarded. It represents a message which has been put in some implicit medium.
- Reductions are of the form

$$\overline{a}\langle v\rangle \parallel (a(x).P + Q) \longmapsto P\{^{v}/_{y}\}$$

Now meaning that particle $\overline{a}\langle v\rangle$ has been removed from the medium and used in P, possibly liberating new particles

Asynchronous π-calculus

- The asynchronous π-calculus results from disallowing continuations after an output prefix
- The only possible continuation for an output prefix is thus $\mathbf{0}$. We have "output particles" of the form $\overline{a}\langle v\rangle.\mathbf{0}$, noted $\overline{a}\langle v\rangle$.
- Sending a message means leaving such particles unguarded. It represents a message which has been put in some implicit medium.
- Reductions are of the form

$$\overline{a}\langle v\rangle \parallel (a(x).P + Q) \longmapsto P\{^{v}/_{y}\}$$

Now meaning that particle $\overline{a}\langle v\rangle$ has been removed from the medium and used in P, possibly liberating new particles

Asynchronous π-calculus

- The asynchronous π-calculus results from disallowing continuations after an output prefix
- The only possible continuation for an output prefix is thus $\mathbf{0}$. We have "output particles" of the form $\overline{a}\langle v\rangle.\mathbf{0}$, noted $\overline{a}\langle v\rangle$.
- Sending a message means leaving such particles unguarded. It represents a message which has been put in some implicit medium.
- Reductions are of the form

$$\overline{a}\langle v\rangle \parallel (a(x).P + Q) \longmapsto P\{{}^{v}\!/\!{}_{y}\}$$

  Now meaning that particle $\overline{a}\langle v\rangle$ has been removed from the medium and used in P, possibly liberating new particles

Asynchronous π-calculus

- The asynchronous π-calculus is interesting because is simple and conveys a practical sense
- It also leads to fresh perspectives from the point of the behavioral theory.
- The notion of observation changes: it only makes sense to observe outputs.
- It is also interesting from the point of view of expressiveness:
  - Is the synchronous π-calculus equally expressive as its asynchronous fragment?
  - Is the causality induced by output actions relevant?
  - What happens when one assumes explicit communication mediums?

Asynchronous π-calculus

- The asynchronous π-calculus is interesting because is simple and conveys a practical sense
- It also leads to fresh perspectives from the point of the behavioral theory.
- The notion of observation changes: it only makes sense to observe outputs.
- It is also interesting from the point of view of expressiveness:
  - Is the synchronous π-calculus equally expressive as its asynchronous fragment?
  - Is the causality induced by output actions relevant?
  - What happens when one assumes explicit communication mediums?

Private π-calculus

In the π-calculus we can distinguish two kinds of mobility:

1. External mobility:

$$\overline{x}\langle z\rangle.P \parallel x(y).Q \xrightarrow{\tau} P \parallel Q\{^{z}/_{y}\}$$

   Asymmetric interaction: a free name takes the place of a bound name. Substitution on the received side is necessary.

2. Internal mobility:

$$(\nu z)(\overline{x}\langle z\rangle.P) \parallel x(z).Q \xrightarrow{\tau} (\nu z)(P \parallel Q)$$

   Symmetric interaction: the exchange of a private name. Only α-conversion is required.

Private π-calculus

In the π-calculus we can distinguish two kinds of mobility:

1. External mobility:

$$\overline{x}\langle z\rangle.P \parallel x(y).Q \xrightarrow{\tau} P \parallel Q\{z/y\}$$

   Asymmetric interaction: a free name takes the place of a bound name. Substitution on the received side is necessary.

2. Internal mobility:

$$(\nu z)(\overline{x}\langle z\rangle.P) \parallel x(z).Q \xrightarrow{\tau} (\nu z)(P \parallel Q)$$

   Symmetric interaction: the exchange of a private name.
   Only α-conversion is required.

Private π-calculus

- To understand the gap from CCS to the π-calculus, Sangiorgi studied the private π-calculus, with internal mobility only
- Internal mobility was shown to be responsible for most of the expressiveness of the π-calculus, while external mobility leads to most of the difficulties
- It is indeed very expressive fragment:
  - data structures
  - the λ-calculus
  - process passing calculi

  can all be expressed using internal mobility.
- Still, the behavioral theory of the private π-calculus remains close to that of CCS

Private π-calculus

- To understand the gap from CCS to the π-calculus, Sangiorgi studied the private π-calculus, with internal mobility only
- Internal mobility was shown to be responsible for most of the expressiveness of the π-calculus, while external mobility leads to most of the difficulties
- It is indeed very expressive fragment:
  - data structures
  - the λ-calculus
  - process passing calculi

  can all be expressed using internal mobility.
- Still, the behavioral theory of the private π-calculus remains close to that of CCS