Models for Concurrent Computations

- π-calculus: dynamic reconfiguration of communication links
- Behavioural semantics
- Session types for distributed protocols

Plan of Lectures and Tutorials

- Lectures, Tutorials and Exercises
- Attendance via Quiz and Exercises
- Needs a notebook and pencils.
- Lecture notes and Tutorial Sheets are distributed during the lectures.

The π-calculus

The π-calculus was first presented in 1989 by Milner, Parrow and Walker, based on an extension of CCS by Engberg and Nielsen.

It is useful for building models of concurrent/distributed/mobile systems and study their properties, like Turing Machines and the λ-calculus are useful for studying sequential computation.

The impact of π-calculus on industry and academia:

- Message-passing programming (Google's Go, MPI, ...)
- Distributed programming (Erlang, Scala, Cloud Haskell, CML, Java RMI, ...)
- **Web service orchestration and choreography** (e.g. W3C Web Service Choreography Description Languages) and Financial protocols
- Session Types (Red Hat (Scribble), Cognizant, Thoughtswork, VMWare, Amazon)
- Systems Biology and Security protocols (applied pi-calculus)
- Active field of research, especially in US, Europe, in the UK, and at Imperial

References

- What you need for the exam is on these slides (and the tutorial sheets)
- **Introductory book:**

  Communicating and Mobile Systems: the π-calculus (Milner 1999)

- Advanced books:

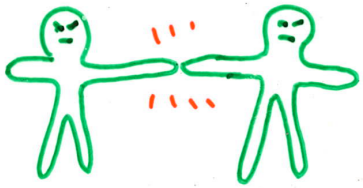
  The π-calculus: a Theory of Mobile Processes (Sangiorgi, Walker 2001)
  Distributed Pi-Calculus (Hennessy 2007)

About the π-calculus

- There is no "canonical" π-calculus. For each application domain, there are alternative notations and lots of specialised variants:
  - we fix one notation and stick to it (once you know one notation, it's easy to understand the others)
  - we start with the simplest variant (the asynchronous π-calculus) and explore various extensions
- An interaction happens by *message passing* rather than *synchronisation*.
- In the asynchronous π-calculus, the communication is *asynchronous* rather than *synchronous*.
- The π-calculus evolved from (value-passing) CCS, but it is more expressive:
  - channel mobility: send and receive *chlaannel names* as messages
  - restriction is interpreted as new (private, secret) channel generation

Synchrony and Asynchrony

- Synchrony



- Asynchrony

cf. E-mail.
News.

Overview of the π-calculus lectures

- A simple version of asynchronous π-calculus: syntax and semantics
- The asynchronous π-calculus: syntax and semantics
- Joyful Hacking in asynchronous π-calculus
- Protocols and Session-calculus

The asynchronous π-calculus

The asynchronous π-calculus is a subset of the π-calculus presented independently by Honda and Tokoro (1991), and by Boudol (1992).

Communication is asynchronous: the output process $\overline{a}\langle b\rangle$ represents a message which is in the communication layer waiting to be picked up by a receiver (it does not have a continuation P like the synchronous process $\overline{a}\langle b\rangle.P$). Several messages can be in the communication layer at the same time, and their order is not preserved. Asynchronous communication is common in distributed systems, and can be used to simulate synchronous communication (handshake) when needed.

The asynchronous π-calculus is easier and more efficient to implement than the full π-calculus. It is widely used as the basis for building more complicated calculi, by adding primitives for distributed or object-oriented programming.

As a start, we consider a subset of asynchronous π-calculus, called *asynchronous* CCS.

Syntax of CCS

	$a, b, c, ..$	name
P, Q ::=		processes
	$\mathbf{0}$	nil process
	$P \mid Q$	parallel composition of P and Q
	$(\nu\, a)P$	restriction of a (scope) in P
	$\overline{a}$	output on channel a
	$a.P$	input on channel a, with continuation P

Notation:
$$\begin{cases} \widetilde{a} \stackrel{\text{df}}{=} a_1, \ldots, a_n & \text{when we don't care about each } a_i \\ (\nu\, a_1, \ldots, a_n)P \stackrel{\text{df}}{=} (\nu\, a_1) \ldots (\nu\, a_n)P \end{cases}$$

Abbreviation:
$$\begin{cases} \overline{a} \stackrel{\text{df}}{=} \overline{a}.\mathbf{0} & \text{when we don't have a continuation} \\ a \stackrel{\text{df}}{=} a.\mathbf{0} & \text{when we don't have a continuation} \end{cases}$$

Examples of CCS

- $\mathbf{0}$ means nothing and $\mathbf{0} \mid \mathbf{0}$ is as same as one $\mathbf{0}$, hence nothing
- $\overline{a}$ is one message to a; and $\overline{a} \mid \overline{a}$ means two messages to a.
- $\overline{a} \mid \overline{b} \mid \overline{c}$

  One message to a, one message to b and One message to c
- $\overline{a} \mid \overline{b}$ is as the same as $\overline{b} \mid \overline{a}$.
- $a.b.c$ inputs from a, then inputs from b and then inputs from c
- $a.b$ does not mean $b.a$
- $a.\overline{b}$ inputs from a then outputs to b, and $a.(\overline{b} \mid \overline{c} \mid \overline{d})$ inputs from a, then outputs to b, c and d.
- $a.b.(\overline{c} \mid \overline{a})$
- $a.(b.\overline{c} \mid d.\overline{e})$

Bad Syntax of CCS

- $\overline{a}.\overline{b}$ and $\overline{a}.b$
- $a.(b \mid c).d$ and $a.(b \mid c).\overline{d}$
- $\mathbf{0.0}$

Reduction of CCS: Informally

We write $P \longrightarrow Q$ if P reduces to Q, like $8 + 2 \longrightarrow 10$.

- $\overline{a} \,|\, a.\mathbf{0} \longrightarrow \mathbf{0}$
- $\overline{a} \,|\, a.\overline{b} \longrightarrow \overline{b}$
- $\overline{a} \,|\, a.\overline{b} \,|\, \overline{c} \,|\, c.\overline{d}$ (hint: $(10 + 2) - (2 + 4)$)
- $\overline{a} \,|\, a.\overline{b} \,|\, a.\overline{c}$ reduces to either $\overline{b} \,|\, a.\overline{c}$ or $a.\overline{b} \,|\, \overline{c}$, that is

$$\overline{a} \,|\, a.\overline{b} \,|\, a.\overline{c} \longrightarrow \overline{b} \,|\, a.\overline{c} \quad \text{or} \quad \overline{a} \,|\, a.\overline{b} \,|\, a.\overline{c} \longrightarrow a.\overline{b} \,|\, \overline{c}$$

- $\overline{a} \,|\, \overline{a} \,|\, a.\overline{b} \,|\, a.\overline{c}$
- $\overline{a} \,|\, \overline{b} \,|\, a.b.\overline{e} \,|\, b.a.\overline{d}$

Name Restriction of CCS: Informally

- $(\nu\, a)(\overline{a} \,|\, a.\overline{b}) \,|\, a.\overline{c}$ means $(\nu\, d)(\overline{d} \,|\, d.\overline{b}) \,|\, a.\overline{c}$
  Similar with $f(x) = x + 2$ means $f(y) = y + 2$

- $(\nu\, a)(\overline{a} \,|\, a.\overline{b}) \,|\, a.\overline{c} \longrightarrow (\nu\, a)\overline{b} \,|\, a.\overline{c} \equiv \overline{b} \,|\, a.\overline{c}$

- but $(\nu\, a)(\overline{a} \,|\, a.\overline{b}) \,|\, a.\overline{c} \not\longrightarrow a.\overline{b} \,|\, \overline{c}$

Names and Variables

We separate channel names, which are like constants in a programming language, from variables, which are used to instantiate messages received in input. Consequently, we have two sorts:

$a, b, c \in \mathcal{N}$	Channel Names
$x, y, z \in \mathcal{V}$	Variables

This distinction is not mandatory to build the theory, but is convenient when the calculus is used to model actual systems.

Syntax of asynchronous π-calculus

u, v ::=		identifiers
	$a, b, c, ..$	name
	$x, y, z, ..$	variable
P, Q ::=		processes
	$\mathbf{0}$	nil process
	$P \mid Q$	parallel composition of P and Q
	$(\nu\, a)P$	generation of a with scope P (also called *restriction*)
	$!P$	replication of P, i.e. infinite parallel composition $P \mid P \mid P \mid \ldots$
	$\overline{u}\langle v \rangle$	output of v on channel u
	$u(x).P$	input of *distinct* variables x on u, with continuation P

Notation:
$$\begin{cases} \widetilde{u} \stackrel{\text{df}}{=} u_1, \ldots, u_n & \text{when we don't care about each } u_i \\ (\nu\, a_1, \ldots, a_n)P \stackrel{\text{df}}{=} (\nu\, a_1) \ldots (\nu\, a_n)P \end{cases}$$

Later on, we will consider other operators such as choice, output continuation, recursive definitions.

Free variables and free names

In order to understand the formal semantics of the π-calculus, it is important to know exactly what are the free variables fv and the free names fn of each term.

$$
\begin{aligned}
\mathit{fv}(x) &= \{x\} & \qquad \mathit{fn}(x) &= \emptyset \\
\mathit{fv}(a) &= \emptyset & \qquad \mathit{fn}(a) &= \{a\} \\
\mathit{fv}(\mathbf{0}) &= \emptyset & \qquad \mathit{fn}(\mathbf{0}) &= \emptyset \\
\mathit{fv}(P \mid Q) &= \mathit{fv}(P) \cup \mathit{fv}(Q) & \qquad \mathit{fn}(P \mid Q) &= \mathit{fn}(P) \cup \mathit{fn}(Q) \\
\mathit{fv}((\nu\, a)P) &= \mathit{fv}(P) & \qquad \mathit{fn}((\nu\, a)P) &= \mathit{fn}(P) \setminus \{a\} \\
\mathit{fv}(!P) &= \mathit{fv}(P) & \qquad \mathit{fn}(!P) &= \mathit{fn}(P) \\
\mathit{fv}(\overline{u}\langle v\rangle) &= \mathit{fv}(u) \cup \mathit{fv}(v) & \qquad \mathit{fn}(\overline{u}\langle v\rangle) &= \mathit{fn}(u) \cup \mathit{fn}(v) \\
\mathit{fv}(u(x).P) &= \mathit{fv}(u) \cup (\mathit{fv}(P) \setminus \{x\}) & \qquad \mathit{fn}(u(x).P) &= \mathit{fn}(u) \cup \mathit{fn}(P)
\end{aligned}
$$

Both $u(-)$ and $(\nu\, -)$ are called *binders*. A term is *closed* if it has no free variables, it is *open* otherwise.

In the process $P = (\nu\, b)a(x).(\overline{x}\langle z\rangle \mid \overline{x}\langle b\rangle)$, we have highlighted all the *free occurrences* of names and variables. In particular, $\mathit{fv}(P) = \{z\}$ and $\mathit{fn}(P) = \{a\}$. Above, the first occurrences of b and x are called *binding occurrences*, whereas the second occurrence of b and the second and third occurrences of x are called *bound*.

α-conversion

α-conversion is the meta-operation of renaming consistently (i.e. avoiding clashes) the bound names or variables of a process. If P is obtained from Q by α-conversion, we say that P and Q are α-equivalent, and we write $P =_\alpha Q$. For example:

$$(\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, c)\overline{c}\langle a\rangle) =_\alpha (\nu\, d)(\overline{d}\langle b\rangle \mid (\nu\, c)\overline{c}\langle d\rangle)$$

But we cannot replace a with b:

$$(\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, c)\overline{c}\langle a\rangle) \neq_\alpha (\nu\, b)(\overline{b}\langle b\rangle \mid (\nu\, c)\overline{c}\langle b\rangle)$$

Can we replace a with c?

$$(\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, c)\overline{c}\langle a\rangle) \neq_\alpha (\nu\, c)(\overline{c}\langle b\rangle \mid (\nu\, c)\overline{c}\langle c\rangle)$$

Not naively! We can if, for example, we first α-convert the name c to e.

$$(\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, c)\overline{c}\langle a\rangle) =_\alpha (\nu\, c)(\overline{c}\langle b\rangle \mid (\nu\, e)\overline{e}\langle c\rangle)$$

The intuition is that α-conversion preserves each difference between names. We will use α-conversion very often, without explicit mention.

Substitution

A *substitution* $\{^{a}/_{x}\}$ applied to a process P (denoted by $P\{^{a}/_{x}\}$), has the effect of replacing all the free occurrences of x in P with a.

The substitution avoids clashes with any bound names by implicitly using α-conversion (*capture-avoiding substitution*). For example:

$$((\nu\, d)(\overline{a}\langle b\rangle \mid \overline{a}\langle d\rangle \mid \overline{a}\langle x\rangle))\{^{d}/_{x}\} \;=\; (\nu\, c)(\overline{a}\langle b\rangle \mid \overline{a}\langle c\rangle \mid \overline{a}\langle d\rangle))$$

where we have avoided the capture of the external d by α-converting $(\nu\, d)$ to $(\nu\, c)$.

Given a substitution σ and an open process P, if $P\sigma$ is closed (i.e. $\mathit{fv}(P\sigma) = \emptyset$) we say that σ is a *closing substitution*. For example, $\{^{b}/_{x}\}$ is closing for $a(y).\overline{x}\langle y\rangle$:

$$a(y).\overline{x}\langle y\rangle\{^{b}/_{x}\} = a(y).\overline{b}\langle y\rangle \qquad \mathit{fv}(a(y).\overline{b}\langle y\rangle) = \emptyset$$

As a shorthand, we use the notation $\{^{v_1,\cdots,v_n}/_{x_1,\ldots,x_n}\} \stackrel{\text{df}}{=} \{^{v_1}/_{x_1}\}\ldots\{^{v_n}/_{x_n}\}$.

Quiz: α-conversion and substitution

1. Correct?
   $(\nu\, a)(\overline{a}\langle b\rangle \mid \overline{c}\langle a\rangle) =_\alpha (\nu\, d)(\overline{d}\langle b\rangle \mid \overline{c}\langle d\rangle)$

2. Correct?
   $(\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, a)\overline{c}\langle a\rangle) =_\alpha (\nu\, d)(\overline{d}\langle b\rangle \mid (\nu\, a)\overline{c}\langle a\rangle)$

3. Correct?
   $(\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, a)\overline{c}\langle a\rangle) =_\alpha (\nu\, d)(\overline{d}\langle b\rangle \mid (\nu\, a)\overline{c}\langle d\rangle)$

4. Correct?
   $a(y).\overline{x}\langle y\rangle\{^b/_y\} = a(b).\overline{x}\langle b\rangle$

5. Correct?
   $a(y).(\overline{x}\langle y\rangle \mid (\nu\, a)\overline{c}\langle a\rangle)\{^a/_x\} = a(y).(\overline{a}\langle y\rangle \mid (\nu\, a)\overline{c}\langle a\rangle)$

6. Correct?
   $a(y).(\overline{x}\langle y\rangle \mid (\nu\, a)\overline{c}\langle x\rangle)\{^a/_x\} = a(y).(\overline{a}\langle y\rangle \mid (\nu\, a)\overline{c}\langle a\rangle)$

Quiz: Answers: α-conversion and substitution

1. Yes

   $(\nu\, a)(\overline{a}\langle b\rangle \mid \overline{c}\langle a\rangle) =_\alpha (\nu\, d)(\overline{d}\langle b\rangle \mid \overline{c}\langle d\rangle)$

2. Yes

   $(\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, a)\overline{c}\langle a\rangle) =_\alpha (\nu\, d)(\overline{d}\langle b\rangle \mid (\nu\, a)\overline{c}\langle a\rangle)$

3. No.

   $(\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, a)\overline{c}\langle a\rangle) =_\alpha (\nu\, a)(\overline{a}\langle b\rangle \mid (\nu\, d)\overline{c}\langle d\rangle) \neq_\alpha (\nu\, d)(\overline{d}\langle b\rangle \mid (\nu\, a)\overline{c}\langle d\rangle)$

4. No. Because

   $a(y).\overline{x}\langle y\rangle\{^b/_y\} =_\alpha a(z).\overline{x}\langle z\rangle\{^b/_y\} = a(z).\overline{x}\langle z\rangle$

5. Yes. Because

   $a(y).(\overline{x}\langle y\rangle \mid (\nu\, a)\overline{c}\langle a\rangle)\{^a/_x\} = a(y).(\overline{a}\langle y\rangle \mid (\nu\, a)\overline{c}\langle a\rangle) =_\alpha a(y).(\overline{a}\langle y\rangle \mid (\nu\, d)\overline{c}\langle d\rangle)$

6. No. Because

   $a(y).(\overline{x}\langle y\rangle \mid (\nu\, a)\overline{c}\langle x\rangle) =_\alpha a(y).(\overline{a}\langle y\rangle \mid (\nu\, d)\overline{c}\langle x\rangle)$

   Hence

   $a(y).(\overline{x}\langle y\rangle \mid (\nu\, a)\overline{c}\langle x\rangle)\{^a/_x\} =_\alpha a(y).(\overline{a}\langle y\rangle \mid (\nu\, d)\overline{c}\langle a\rangle)$

Structural congruence (1)

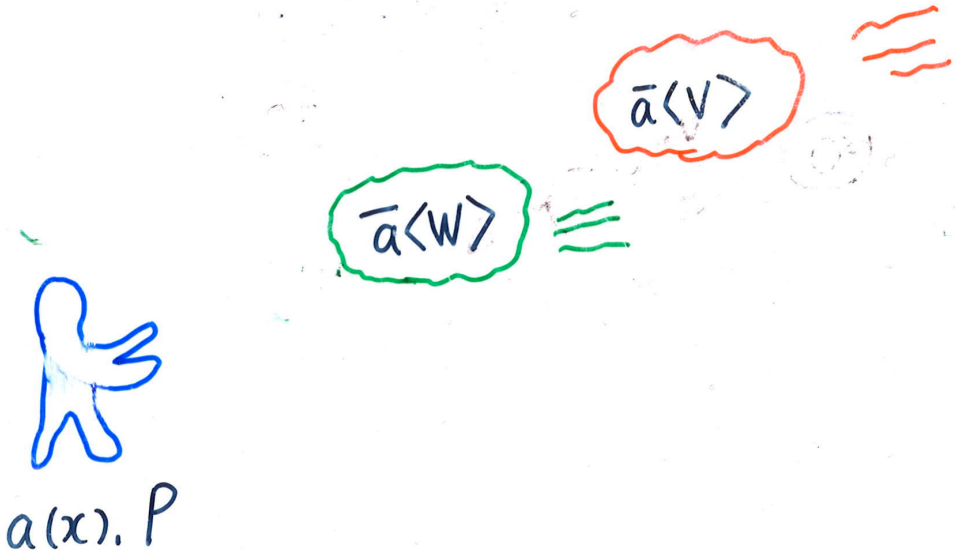
The reduction semantics of the π-calculus is inspired by the Chemical Abstract Machine (CHAM) of Berry and Boudol. Processes "float around" like molecules in a solution using structural congruence ($\equiv$) and "react" using a reduction relation ($\longrightarrow$).

The intuition is that if $P \equiv Q$ then we consider P and Q completely interchangeable.

Structural congruence is an equivalence relation, is preserved by all the syntactic operators, and contains α-equivalence.

$P \equiv P$	(Eq Reflexivity)
$P \equiv Q \Longrightarrow Q \equiv P$	(Eq Symmetry)
$P \equiv R \ and \ R \equiv Q \Longrightarrow P \equiv Q$	(Eq Transitivity)
$P \equiv Q \Longrightarrow (\nu\, a)P \equiv (\nu\, a)Q$	(Cong Res)
$P \equiv Q \Longrightarrow P \mid R \equiv Q \mid R$	(Cong Par)
$P \equiv Q \Longrightarrow u(x).P \equiv u(x).Q$	(Cong In)
$P \equiv Q \Longrightarrow !P \equiv !Q$	(Cong Rep)
$P =_\alpha Q \Longrightarrow P \equiv Q$	(α-equivalence)

Structural congruence satisfies also additional rules specific to the π-calculus. . .


ā⟨v⟩
ā⟨w⟩
a(x).P
a(x).P | ā⟨v⟩ | ā⟨w⟩
ā⟨w⟩
ā⟨v⟩
≡
a(x).P
(a(x).P | ā⟨w⟩ | ā⟨v⟩

Structural congruence (2)

The set of all processes, with the operation of parallel composition, and with the nil process as the neutral element are a *commutative monoid*.

$P \mid (Q \mid Q') \equiv (P \mid Q) \mid Q'$ (Associativity)

$P \mid Q \equiv Q \mid P$ (Commutativity)

$P \mid \mathbf{0} \equiv P$ (Zero)

That is, we can exchange freely the position of parallel processes and we can insert or delete the $\mathbf{0}$ process at will. For example:

$$\mathbf{0} \mid \mathbf{0} \mid P \mid Q \mid R \equiv Q \mid \mathbf{0} \mid R \mid P$$

Replication can be folded or unfolded as many times as needed.

$!P \equiv P \mid !P$ (Rep)

Thanks to this rule, the π-calculus can express infinite computations.

Structural congruence (3)

The last rules that complete the definition of $\equiv$ state that the scope of a restricted name can be extended (or contracted) across terms which do not contain free occurrences of that name.

$(\nu\, a)\mathbf{0} \equiv \mathbf{0}$ (Res Nil)

$(\nu\, a)(\nu\, b)P \equiv (\nu\, b)(\nu\, a)P$ (Res Res)

$a \notin \mathit{fn}(P) \Longrightarrow P \,|\, (\nu\, a)Q \equiv (\nu\, a)(P \,|\, Q)$ (Res Par)

For example:

$$(\nu\, a, b)(a(x).\overline{x}\langle c\rangle \,|\, \overline{a}\langle b\rangle) \equiv (\nu\, a)(a(x).\overline{x}\langle c\rangle \,|\, (\nu\, b)(\overline{a}\langle b\rangle))$$

$$(\nu\, a, b)(c(x).\overline{c}\langle x\rangle \,|\, \overline{c}\langle d\rangle) \equiv c(x).\overline{c}\langle x\rangle \,|\, \overline{c}\langle d\rangle$$

It is very important to gain familiarity with structural congruence, we will use it very often during the rest of the course.

Question 1 *Prove* $(\nu\, c)P \equiv P$ *if* $c \notin \mathit{fn}(P)$.

Question 2 *Prove if* $P \equiv Q$ *then* $\mathit{fn}(P) = \mathit{fn}(Q)$ *and* $\mathit{fv}(P) = \mathit{fv}(Q)$.

Reduction relation

Reduction describes how processes interact by exchanging messages. It is the smallest *partial relation* between processes satisfying the rules given below.

$$\overline{a}\langle v\rangle \,|\, a(x).P \;\longrightarrow\; P\{^{v}/_{x}\} \qquad \text{(Comm)}$$

$$\frac{P \longrightarrow P'}{P \,|\, Q \longrightarrow P' \,|\, Q} \qquad \text{(Par)}$$

$$\frac{P \longrightarrow P'}{(\nu\, a)P \longrightarrow (\nu\, a)P'} \qquad \text{(Res)}$$

$$\frac{P \equiv Q \longrightarrow Q' \equiv P'}{P \longrightarrow P'} \qquad \text{(Struct)}$$

As a shorthand notation, we write $P \stackrel{*}{\longrightarrow} Q$ if $P \equiv Q$ or $P \longrightarrow \ldots \longrightarrow Q$.

Reduction Examples

Communication:

$$a(x).\overline{x}\langle c\rangle \,|\, \overline{a}\langle b\rangle \longrightarrow \overline{b}\langle c\rangle \qquad \text{by (Struct), (Commutativity) and (Comm)}$$

$$\begin{array}{lll} & a(x).\overline{x}\langle c\rangle \,|\, \overline{a}\langle b\rangle & \\ \equiv & \overline{a}\langle b\rangle \,|\, a(x).\overline{x}\langle c\rangle & \text{by (Struct), (Commutativity)} \\ \longrightarrow & \overline{b}\langle c\rangle & \text{by (Comm)} \end{array}$$

Scope extrusion:

$$(\nu\, b)(\overline{a}\langle b\rangle) \,|\, a(x).\overline{c}\langle x\rangle \longrightarrow (\nu\, b)\overline{c}\langle b\rangle \qquad \text{by (Struct), (Res Par), (Res) and (Comm)}$$

$$\begin{array}{lll} & (\nu\, b)(\overline{a}\langle b\rangle) \,|\, a(x).\overline{c}\langle x\rangle & \text{note } b \notin \mathit{fn}(a(x).\overline{c}\langle x\rangle) \\ \equiv & (\nu\, b)(\overline{a}\langle b\rangle \,|\, a(x).\overline{c}\langle x\rangle) & \text{by (Struct), (Res Par)} \\ \longrightarrow & (\nu\, b)\overline{c}\langle b\rangle & \text{by (Res) and (Comm)} \end{array}$$

Infinite behaviour and Sorts

We use the macros for process definition $\mathbf{A}(\widetilde{x})$ and usage $\mathbf{A}\langle\widetilde{v}\rangle$ *only informally*. To represent infinite behaviour, we use the *replication* operator $!P$, stating that there are as many copies of P as needed, all running in parallel. For example:

$$a(x).P \,|\, a(x).Q \,|\, !\overline{a}\langle b\rangle \longrightarrow P\{^{b}/_{x}\} \,|\, a(x).Q \,|\, !\overline{a}\langle b\rangle \longrightarrow P\{^{b}/_{x}\} \,|\, Q\{^{b}/_{x}\} \,|\, !\overline{a}\langle b\rangle$$

Replication is very simple, yet combined with channel generation it can encode recursive definitions (later on we will see how).

Reduction Examples

Infinite behaviour:

$$\overline{a}\langle b\rangle \mid !a(x).\overline{a}\langle x\rangle \longrightarrow \overline{a}\langle b\rangle \mid !a(x).\overline{a}\langle x\rangle \qquad \text{by (Struct), (Rep), (Par) and (Comm)}$$

$$\begin{array}{lll} & \overline{a}\langle b\rangle \mid !a(x).\overline{a}\langle x\rangle & \\ \equiv & \overline{a}\langle b\rangle \mid a(x).\overline{a}\langle x\rangle \mid !a(x).\overline{a}\langle x\rangle & \text{by (Struct), (Rep)} \\ \longrightarrow & \overline{a}\langle b\rangle \mid !a(x).\overline{a}\langle x\rangle & \text{by (Par) and (Comm)} \end{array}$$

Nondeterminism:

$$\overline{a}\langle b\rangle \mid \overline{a}\langle d\rangle \mid a(x).\overline{c}\langle x\rangle \begin{array}{ll} \nearrow \overline{a}\langle b\rangle \mid \overline{c}\langle d\rangle & \text{by (Par) and (Comm)} \\ \searrow \overline{a}\langle d\rangle \mid \overline{c}\langle b\rangle & \text{using also (Struct), (Commutativity)} \end{array}$$

Atoms for Name Passing

Small Agents (1)

Forwarder $\mathbf{FW}(a,b) \stackrel{\text{df}}{=} a(z).\overline{b}\langle z\rangle$

A *forwarder* from channel a to channel b, is a process that forwards a message for a on b.

$$\mathbf{FW}\langle a,b\rangle \,|\, \overline{a}\langle d\rangle \longrightarrow \overline{b}\langle d\rangle$$

$$(\nu\, b)(\mathbf{FW}\langle a,b\rangle \,|\, \mathbf{FW}\langle b,c\rangle) \,|\, \overline{a}\langle d\rangle \longrightarrow\longrightarrow \overline{c}\langle d\rangle$$

Duplicator $\mathbf{D}(a,b,c) \stackrel{\text{df}}{=} a(z).(\overline{b}\langle z\rangle \mid \overline{c}\langle z\rangle)$

A *duplicator* from channel a to b and c, is a process that duplicates a message for a to b and c.

$$\mathbf{D}\langle a,b,c\rangle \,|\, \overline{a}\langle d\rangle \longrightarrow (\overline{b}\langle d\rangle \,|\, \overline{c}\langle d\rangle)$$

$$(\nu\, b)(\mathbf{D}\langle a,b,c_1\rangle \,|\, \mathbf{D}\langle b,c_2,c_3\rangle) \,|\, \overline{a}\langle d\rangle \longrightarrow\longrightarrow (\overline{c_1}\langle d\rangle \,|\, \overline{c_2}\langle d\rangle \,|\, \overline{c_3}\langle d\rangle)$$

Killer $\mathbf{K}(a) \stackrel{\text{df}}{=} a(z).\mathbf{0}$

A *killer* at channel a is a process that kills a message from a.

$a(z).(P \,|\, Q)$ can be decomposed as $(\nu\, c_1, c_2)(\mathbf{D}\langle a, c_1, c_2\rangle \,|\, c_1(z).P \,|\, c_2(z).Q)$.

For example, $a(z).(\overline{b}\langle z\rangle \,|\, \mathbf{0})$ is the same as $(\nu\, c_1, c_2)(\mathbf{D}\langle a, c_1, c_2\rangle \,|\, \mathbf{FW}\langle c_1, b\rangle \,|\, \mathbf{K}\langle c_2\rangle)$

Small Agents (3)

Identity Receptor $\mathbf{I}(a) \stackrel{\text{df}}{=} !\mathbf{FW}\langle a, a\rangle$

An *identity receptor* at channel a is a process that forwards messages for a on a.

$$\overline{a}\langle d\rangle \,|\, \mathbf{I}\langle a\rangle \longrightarrow \overline{a}\langle d\rangle \,|\, \mathbf{I}\langle a\rangle \longrightarrow \overline{a}\langle d\rangle \,|\, \mathbf{I}\langle a\rangle \longrightarrow \overline{a}\langle d\rangle \,|\, \mathbf{I}\langle a\rangle \ldots$$

Equator $\mathbf{EQ}(a, b) \stackrel{\text{df}}{=} !\mathbf{FW}\langle a, b\rangle \,|\, !\mathbf{FW}\langle b, a\rangle.$

An *equator* between two channels a and b, is a process that forwards all messages for a on b, and viceversa, making a and b in some sense "equivalent". For example,

$$\overline{a}\langle d\rangle \,|\, \mathbf{EQ}\langle a, b\rangle \longrightarrow \overline{b}\langle d\rangle \,|\, \mathbf{EQ}\langle a, b\rangle \longrightarrow \overline{a}\langle d\rangle \,|\, \mathbf{EQ}\langle a, b\rangle \longrightarrow \ldots$$

Omega $\Omega \stackrel{\text{df}}{=} (\nu\, a)(!\mathbf{FW}\langle a, a\rangle \,|\, \overline{a}\langle a\rangle)$

An *omega* is a process that continues infinite reductions by himself.

$$\Omega \longrightarrow \Omega \longrightarrow \Omega \longrightarrow \cdots$$

New Name Generator $\mathbf{NN}(a) \stackrel{\text{df}}{=} !a(x).(\nu\, b)\overline{x}\langle b\rangle$. A *new name generator* is a process that creates a new name infinitely when it is asked.

$$\overline{a}\langle c\rangle \,|\, \overline{a}\langle d\rangle \,|\, \mathbf{NN}\langle a\rangle \longrightarrow (\nu\, b)\overline{c}\langle b\rangle \,|\, \overline{a}\langle d\rangle \,|\, \mathbf{NN}\langle a\rangle \longrightarrow (\nu\, b)\overline{c}\langle b\rangle \,|\, (\nu\, b')\overline{d}\langle b'\rangle \,|\, \mathbf{NN}\langle a\rangle$$

Quiz: Name Generateor

1. Is $\mathbf{NN}(a) \stackrel{\text{df}}{=} !a(x).(\nu\, b)\overline{x}\langle b\rangle$ structural congruent with $\mathbf{NN}_1(a) = (\nu\, b)!a(x).\overline{x}\langle b\rangle$?

2. Is $\mathbf{NN}(a) \stackrel{\text{df}}{=} !a(x).(\nu\, b)\overline{x}\langle b\rangle$ structural congruent with $\mathbf{NN}_2(a) =!(\nu\, b)a(x).\overline{x}\langle b\rangle$?

Quiz: Answer: Name Generateor

1. Is $\mathbf{NN}(a) \stackrel{\text{df}}{=} !a(x).(\nu\, b)\overline{x}\langle b\rangle$ structural congruent with $\mathbf{NN}_1(a) = (\nu\, b)!a(x).\overline{x}\langle b\rangle$?

   No: since

$$\begin{array}{lcl} \mathbf{NN}(a) & = & !a(x).(\nu\, b)\overline{x}\langle b\rangle \\ & \equiv & a(x).(\nu\, b)\overline{x}\langle b\rangle \,|\, a(x).(\nu\, b)\overline{x}\langle b\rangle \,|\, \cdots \,|\, a(x).(\nu\, b)\overline{x}\langle b\rangle \,|\, !a(x).(\nu\, b)\overline{x}\langle b\rangle \\ \mathbf{NN}_1(a) & = & (\nu\, b)!a(x).\overline{x}\langle b\rangle \\ & \equiv & (\nu\, b)(a(x).\overline{x}\langle b\rangle \,|\, a(x).\overline{x}\langle b\rangle \,|\, \cdots \,|\, a(x).\overline{x}\langle b\rangle \,|\, !a(x).\overline{x}\langle b\rangle) \end{array}$$

2. Is $\mathbf{NN}(a) \stackrel{\text{df}}{=} !a(x).(\nu\, b)\overline{x}\langle b\rangle$ structural congruent with $\mathbf{NN}_2(a) = !(\nu\, b)a(x).\overline{x}\langle b\rangle$?

   No: since we do **not** have the rule $(\nu\, b)a(x).P \equiv a(x).(\nu\, b)P$ but they are observationally equivalent as we shall see soon.

Quiz: Small Agents

1. Is $\mathbf{I}\langle a\rangle$ like $\mathbf{0}$?

2. Is Ω like $\mathbf{0}$?

3. Is $\mathbf{EQ}\langle a,b\rangle \,|\, \overline{a}\langle d\rangle$ like $\mathbf{EQ}\langle a,b\rangle \,|\, \overline{b}\langle d\rangle$?

4. Is $\mathbf{EQ}\langle a,b\rangle \,|\, \overline{d}\langle a\rangle$ like $\mathbf{EQ}\langle a,b\rangle \,|\, \overline{d}\langle b\rangle$?

5. Is $\mathbf{EQ}\langle a,b\rangle \,|\, \overline{d}\langle a\rangle$ like $\mathbf{EQ}\langle a,b\rangle \,|\, \overline{d}\langle b\rangle$?

Joyful Hacking in π-calculus

Channel mobility example

Internet connection: a $\mathbf{Client}$ and a $\mathbf{Server}$ talk on dedicated communication ports, set up using the channel a:

$$\mathbf{Client}(a, c) \stackrel{\text{df}}{=} (\overline{a}\langle c\rangle \mid c(x).\mathbf{Client}_1\langle c, x\rangle)$$
$$\mathbf{Server}(a, s) \stackrel{\text{df}}{=} a(y).(\overline{y}\langle s\rangle \mid \mathbf{Server}_1\langle y, s\rangle)$$

Difference with CCS: the variable y is used as the name of a channel.

$$\begin{array}{ll} & \mathbf{Client}\langle a, c\rangle \,|\, \mathbf{Server}\langle a, s\rangle \\ \longrightarrow & c(x).\mathbf{Client}_1\langle c, x\rangle \,|\, \overline{c}\langle s\rangle \mid \mathbf{Server}_1\langle c, s\rangle \\ \longrightarrow & \mathbf{Client}_1\langle c, s\rangle \,|\, \mathbf{Server}_1\langle c, s\rangle \end{array}$$

After the two communication steps, client and server know each other's port.

Channel generation example

We can make the Internet connection example more flexible. To avoid external interferences, the client and server exchange newly generated port names:

$$\mathbf{Client}(a) \stackrel{\text{df}}{=} (\nu\, c)(\overline{a}\langle c\rangle \mid c(x).\mathbf{Client}_1\langle c, x\rangle)$$

$$\mathbf{Server}(a) \stackrel{\text{df}}{=} a(y).(\nu\, s)(\overline{y}\langle s\rangle \mid \mathbf{Server}_1\langle y, s\rangle)$$

No other process knows about c and s because they are inside the name restriction: you can imagine that they will be created at run-time by the ν operator (pronounced "new").

$$\begin{aligned} & \mathbf{Client}\langle a\rangle \mid \mathbf{Server}\langle a\rangle \\ \longrightarrow\ & (\nu\, c)(c(x).\mathbf{Client}_1\langle c, x\rangle \mid (\nu\, s)(\overline{c}\langle s\rangle \mid \mathbf{Server}_1\langle c, s\rangle)) \\ \longrightarrow\ & (\nu\, c, s)(\mathbf{Client}_1\langle c, s\rangle \mid \mathbf{Server}_1\langle c, s\rangle) \end{aligned}$$

Note that $\mathbf{Client}_1$ and $\mathbf{Server}_1$ are now within the scope of the ν operator. This phenomenon, called *scope extrusion*, is a distinguishing feature of the π-calculus.

Secure client-server communication

We can now represent secure client-server communication. The client creates a new (secret) channel c before contacting the server on the public channel a:

$$\mathbf{SClient_i}(a) \stackrel{\text{df}}{=} (\nu\, c)(\overline{a}\langle c\rangle \mid c(x).\mathbf{PClient_i}\langle c, x\rangle)$$

The server is a replicated process, ready to spawn a *session* for each client request:

$$\mathbf{SServer}(a) \stackrel{\text{df}}{=} !a(y).(\nu\, s)(\overline{y}\langle s\rangle \mid \mathbf{PServer}\langle y, s\rangle)$$

The server can interact with multiple clients at the same time:

$$\begin{aligned}&\mathbf{SClient_1}\langle a\rangle \,|\, \mathbf{SClient_2}\langle a\rangle \,|\, \mathbf{SServer}\langle a\rangle \;\stackrel{*}{\longrightarrow}\\ &\quad (\nu\, s_1, c_1)(\mathbf{PClient_1}\langle c_1, s_1\rangle \,|\, \mathbf{PServer}\langle c_1, s_1\rangle)\\ &\mid (\nu\, s_2, c_2)(\mathbf{PClient_2}\langle c_2, s_2\rangle \,|\, \mathbf{PServer}\langle c_2, s_2\rangle)\\ &\mid \mathbf{SServer}\langle a\rangle\end{aligned}$$

Each session is protected from the external environment by the restriction on the client and server communication ports, which can be used to exchange data without external interferences.

Reduction congruence in π-calculus

Reduction congruence: Intuition

The purpose of *reduction congruence*, denoted by $\cong$, is to determine when two processes are semantically equivalent, in the sense that if $P \cong Q$ then there is no way for an observer *within* the system (i.e. another process) to tell P from Q.

Before giving the formal definition of $\cong$, we look at some intuitive examples of equalities and inequalities that we expect to hold.

$P \equiv Q \Longrightarrow P \cong Q$	structural congruence is just a syntactic reorganization
$!(P \mid P) \cong !P$	in both cases we intuitively have $P \mid P \mid P \mid \ldots$
$(\nu\, a)a.P \cong \mathbf{0}$	a process which is stuck cannot be observed
$(\nu\, a)\overline{a} \cong \mathbf{0}$	same as above
$(\nu\, a)(\overline{a} \mid a) \cong \mathbf{0}$	a process which does not interact externally cannot be observed
$P \oplus P \cong P$	do either P or P
$\overline{a} \ncong \overline{b}$	an observer can tell a difference by trying to input on a
$\overline{c}\langle a\rangle \ncong \overline{c}\langle b\rangle$	an observer can input on c and repeat the experiment above
$\overline{a} \mid \overline{a} \ncong \overline{a}$	an observer can perform two inputs on a

Barbs

In asynchronous semantics, we can only observe the output. Hence, we define some observation predicates called *barbs* which establish if a process has the possibility to send an output message on a given channel (independently from the contents of the message).

The *strong* and *weak barbs* are defined by

$$P \downarrow_a \iff P \equiv (\nu\, \widetilde{c})(\overline{a}\langle \widetilde{v} \rangle \mid Q) \ \textit{where}\ a \notin \{\widetilde{c}\}$$
$$P \Downarrow_a \iff P \stackrel{*}{\longrightarrow} P' \ \textit{and}\ P' \downarrow_a$$

We write $P \not\downarrow_a$ (or $P \not\Downarrow_a$) if it is not the case that $P \downarrow_a$ (respectively $P \Downarrow_a$).

For example:

$$P \stackrel{\text{df}}{=} (\nu\, a)(\overline{c}\langle a \rangle \mid c(x).(\overline{b} \mid \overline{a})) \quad P \downarrow_c \quad P \not\downarrow_b \quad P \Downarrow_b \quad P \not\Downarrow_a$$

Reduction congruence requires that processes have the same weak barbs:

$$\text{if } P \cong Q \text{ then } P \Downarrow_a \Longrightarrow Q \Downarrow_a$$

Hence, it is not trivial, in the sense that there are at least two processes different from one another. For example, $\overline{a} \ncong \overline{b}$ because $\overline{a} \downarrow_a$ but $\overline{b} \not\Downarrow_a$.

Quiz: Barbs

Give the strong barbs and weak barbs of each term.

1. $\overline{a}$
2. $\overline{a} \mid \overline{a}$
3. $\overline{a} \mid \overline{b}$
4. $a.\mathbf{0} \mid b.\mathbf{0}$
5. $a.\overline{b} \mid \overline{a}$
6. $(\nu\, a)(a.\overline{b} \mid \overline{a})$
7. $(\nu\, b)(a.\overline{b} \mid \overline{a})$
8. $(\nu\, a)(a(x).\overline{x} \mid \overline{a}\langle b \rangle)$
9. $(\nu\, a, b)(a(x).\overline{x} \mid \overline{a}\langle b \rangle)$
10. $(\nu\, a, b)(a(x).\overline{x} \mid \overline{a}\langle c \rangle) \mid \overline{e} \mid c.\overline{d}$

Quiz: Answer: Barbs

1. $\overline{a} \downarrow_a$ and $\overline{a} \Downarrow_a$
2. $(\overline{a} \mid \overline{a}) \downarrow_a$ and $(\overline{a} \mid \overline{a}) \Downarrow_a$
3. $(\overline{a} \mid \overline{b}) \downarrow_a$ and $(\overline{a} \mid \overline{b}) \downarrow_b$ and $(\overline{a} \mid \overline{b}) \Downarrow_a$ and $(\overline{a} \mid \overline{b}) \Downarrow_b$
4. $a.\mathbf{0} \not\downarrow_d$ for all d. $a.\mathbf{0} \not\Downarrow_d$ for all d.
5. $(a.\mathbf{0} \mid b.\mathbf{0}) \not\downarrow_d$ for all d and $(a.\mathbf{0} \mid b.\mathbf{0}) \not\Downarrow_d$ for all d
6. $(a.\overline{b} \mid \overline{a}) \downarrow_a$ and $(a.\overline{b} \mid \overline{a}) \Downarrow_a$ and $(a.\overline{b} \mid \overline{a}) \Downarrow_b$
7. $(\nu\, a)(a.\overline{b} \mid \overline{a}) \Downarrow_b$
8. $(\nu\, b)(a.\overline{b} \mid \overline{a}) \downarrow_a$ and $(\nu\, b)(a.\overline{b} \mid \overline{a}) \Downarrow_a$
9. $(\nu\, a)(a(x).\overline{x} \mid \overline{a}\langle b \rangle) \longrightarrow \overline{b}$. Hence $(\nu\, a)(a(x).\overline{x} \mid \overline{a}\langle b \rangle) \Downarrow_b$.
10. $(\nu\, a, b)(a(x).\overline{x} \mid \overline{a}\langle b \rangle) \not\Downarrow_d$ for all d
11. $((\nu\, a, b)(a(x).\overline{x} \mid \overline{a}\langle c \rangle) \mid \overline{e} \mid c.\overline{d}) \downarrow_e$ and $(((\nu\, a, b)(a(x).\overline{x} \mid \overline{a}\langle c \rangle) \mid \overline{e} \mid c.\overline{d}) \Downarrow_{c,e,d}$

Contexts and reduction

In order to test processes with respect to any possible observer, we require that equivalent processes must have the same barbs in all the *reduction contexts* of the form:

$$C ::= \; - \quad | \quad C \,|\, P \quad | \quad P \,|\, C \quad | \quad (\nu\, a)C$$

A context is a function which, given a process P, returns the process obtained by replacing "$-$" with P. For example, if C is $(\nu\, a)(- \,|\, a(x).Q)$ we have

$$C[\overline{a}\langle b\rangle \,|\, b.R] = (\nu\, a)(\overline{a}\langle b\rangle \,|\, b.R \,|\, a(x).Q)$$

Reduction congruence requires that processes are equivalent in all possible reduction contexts:

$$\text{if } P \cong Q \text{ then } \forall C.\, C[P] \cong C[Q]$$

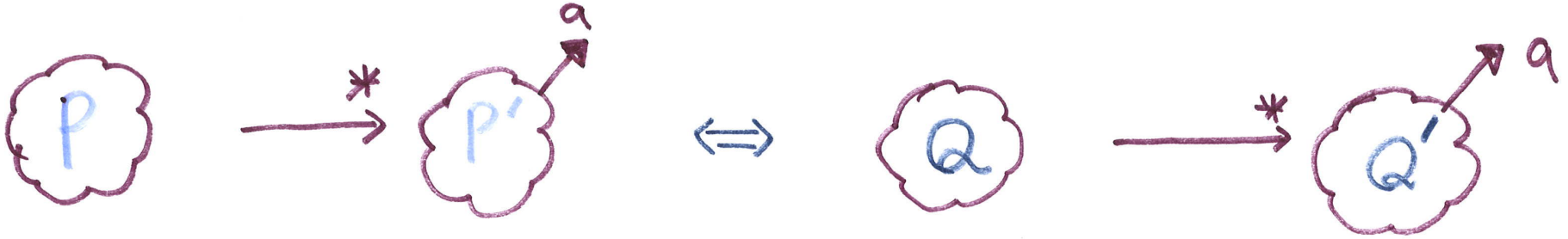
Hence, $\cong$ is a congruence by definition.

Moreover, since we are dealing with reactive systems, we are not satisfied with two processes being equivalent only in their initial state. If one process can evolve to a different state, then the other process must be able to reach a state equivalent to the new state:

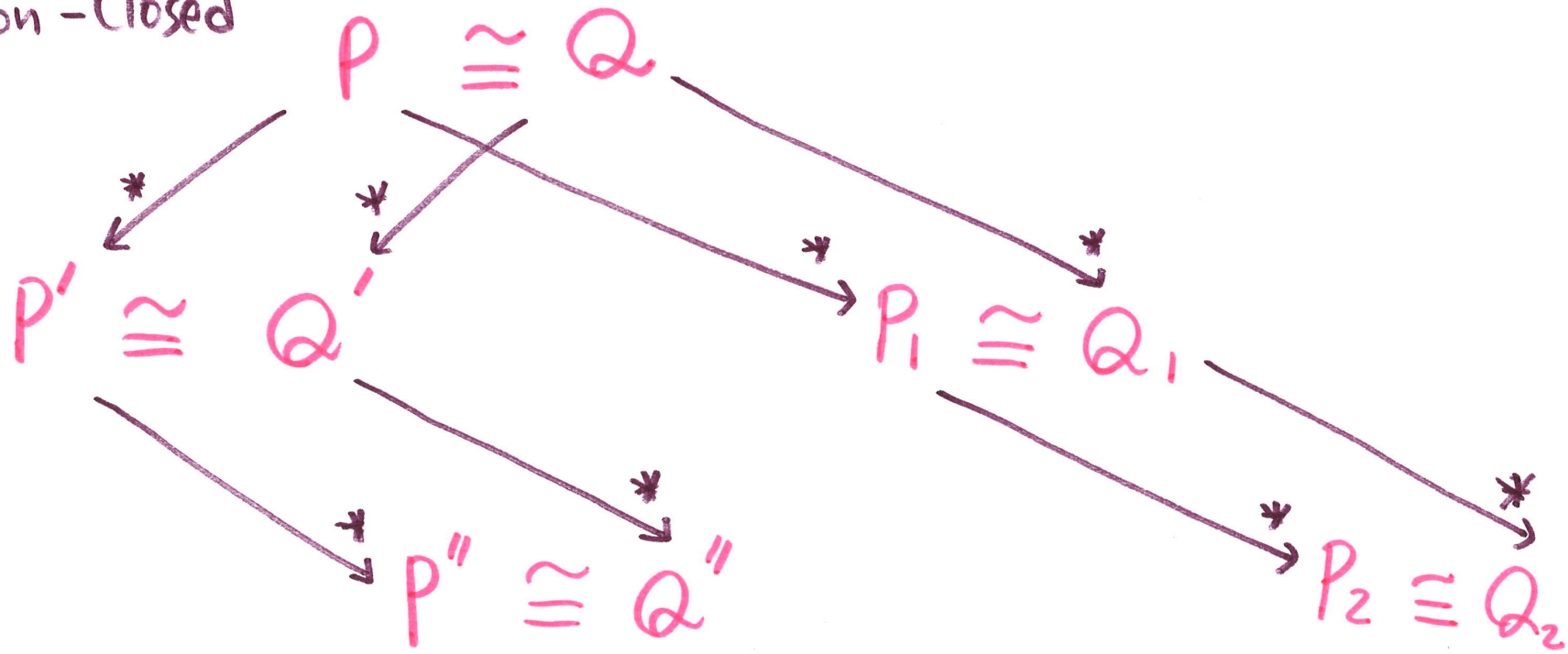
$$\text{if } P \cong Q \text{ then } P \longrightarrow P' \Longrightarrow Q \stackrel{*}{\longrightarrow} Q' \text{ and } P' \cong Q'$$

Hence, $\cong$ is a weak bisimulation on the transition system originated by $\longrightarrow$.

Barb



Reduction-Closed



Reduction congruence: Formal definition

Reduction congruence $\cong$ is the largest symmetric relation $\dot{\cong}$ on *closed* processes such that if $P \dot{\cong} Q$ then:

1. P and Q have the same (weak) barbs: $P \Downarrow_a \Longrightarrow Q \Downarrow_a$;
2. the property is preserved by all reduction contexts: $\forall C.\, C[P] \dot{\cong} C[Q]$;
3. the property is preserved by reductions: $P \longrightarrow P' \Longrightarrow Q \stackrel{*}{\longrightarrow} Q' \ and \ P' \dot{\cong} Q'$.

Two open processes P and Q are reduction congruent if and only if $P\sigma \cong Q\sigma$ for all *closing* substitutions σ.

For example, $\overline{x} \cong \overline{x}$ because for all a, $\overline{a} \cong \overline{a}$, but $\overline{x} \ncong \overline{y}$ because for $\{^a/_x, ^b/_y\}$, $\overline{a} \ncong \overline{b}$.

Reduction congruence is in fact a set of pairs of processes. The notation $P \cong Q$ is equivalent to $(P, Q) \in \cong$, and $P \ncong Q$ is equivalent to $(P, Q) \notin \cong$.

Properties of reduction congruence

- $\cong$ is less restrictive than $\equiv$ (also written $P \equiv Q \Longrightarrow P \cong Q$ or $\equiv \subseteq \cong$).

  Proof. We have to show that the relation $\equiv$ respects the definition of reduction congruence. Since $\cong$ is the largest such relation, then $\equiv \subseteq \cong$.

  By (Eq Symmetry), $\equiv$ is symmetric. Point 1 follows by the fact that the definition of barbs includes $\equiv$. Point 2 follows from the fact that $\equiv$ is a congruence. Point 3 follows from rule (Struct) in the definition of $\longrightarrow$.

- $\cong$ is an equivalence relation.

  Proof. We have to show that $\cong$ is reflexive, symmetric and transitive.

  By definition, $\cong$ is symmetric. We show by contradiction that it is also reflexive and transitive. Suppose that $(P, P) \notin \cong$ and let $\dot{\cong}$ be defined as

  $$\cong \cup \{(C[P], C[P]) \ : \ P \text{ is a process, } C \text{ is a reduction context}\}$$

  It is easy to see that $\dot{\cong}$ respects the definition of reduction congruence, and since it is larger than $\cong$ we have reached a contradiction. The case for transitivity is analogous.

How to prove and disprove reduction congruence

Since reduction congruence requires closure under reduction contexts, in the general case it is hard to prove directly that an equation holds. Still, in some cases we are able to do it.

For example, to prove $(\nu\, a)a.P \cong \mathbf{0}$, we can say that both processes have no barbs and no reductions, but then we have to prove that for all C, $C[(\nu\, a)a.P] \cong C[\mathbf{0}]$. In this case, it can be done by a simple induction on the structure of C.

A more practical approach to prove reduction congruence is by using a more restrictive bisimulation relation which implies $\cong$, and does not quantify over contexts. Another approach is to prove equivalences starting from basic laws (we will see a list of those) and using the congruence and transitivity properties of $\cong$.

For example, we can show that $\overline{b} \cong (\nu\, a)(\overline{a} \,|\, a.(\overline{b} \,|\, (\nu\, c)\overline{c}))$ as follows:

$$\overline{b} \equiv \overline{b} \,|\, \mathbf{0} \cong \overline{b} \,|\, (\nu\, c)\overline{c} \equiv (\nu\, a)(\overline{b} \,|\, (\nu\, c)\overline{c}) \cong (\nu\, a)(\overline{a} \,|\, a.(\overline{b} \,|\, (\nu\, c)\overline{c}))$$

where we have used the law $(\nu\, a)(\overline{a} \,|\, a.P) \cong (\nu\, a)P$.

Showing that two processes P and Q are not reduction congruent is easier: we just need to exhibit a context C such that for some a, $C[P] \Downarrow_a$ and $C[Q] \not\Downarrow_a$, or vice-versa.

For example, we have $\overline{a} \,|\, \overline{a} \ncong \overline{a}$ because the context $C = a.a.\overline{b}$ distinguishes the two processes: $C[\overline{a} \,|\, \overline{a}] \Downarrow_b$ but $C[\overline{a}] \not\Downarrow_b$.

How to disprove reduction congruence

Showing that two processes P and Q are not reduction congruent is easier: we just need to exhibit a context C such that for some a, $C[P] \Downarrow_a$ and $C[Q] \not\Downarrow_a$, or vice-versa.

For example, we have $\overline{a} \,|\, \overline{a} \not\cong \overline{a}$ because the context $C = a.a.\overline{b}$ distinguishes the two processes: $C[\overline{a} \,|\, \overline{a}] \Downarrow_b$ but $C[\overline{a}] \not\Downarrow_b$.

1. $\overline{a} \not\cong c.\overline{a}$ since $\overline{a} \Downarrow_a$ but $c.\overline{a} \not\Downarrow_a$ with $C = \_$.

2. $c.\overline{a} \not\cong c.\overline{b}$ since $C[c.\overline{a}] \Downarrow_a$ but $C[c.\overline{b}] \not\Downarrow_a$ with $C = \_ \,|\, \overline{c}$

3. $\overline{a}\langle c\rangle \not\cong \overline{a}\langle b\rangle$ since $C[\overline{a}\langle c\rangle] \Downarrow_c$ but $C[\overline{a}\langle b\rangle] \not\Downarrow_c$ with $C = \_ \,|\, a(x).\overline{x}$

4. $\overline{a}\langle c\rangle \,|\, \overline{a}\langle d\rangle \not\cong \overline{a}\langle c\rangle$ since $C[\overline{a}\langle c\rangle \,|\, \overline{a}\langle d\rangle] \Downarrow_d$ but $C[\overline{a}\langle c\rangle] \not\Downarrow_d$ with $C =$???

5. $\overline{a}\langle c\rangle \,|\, \overline{a}\langle c\rangle \not\cong \overline{a}\langle c\rangle$ since $C[\overline{a}\langle c\rangle \,|\, \overline{a}\langle c\rangle] \Downarrow_d$ but $C[\overline{a}\langle c\rangle] \not\Downarrow_d$ with $C =$???

How to prove reduction congruence: Part 1

Since reduction congruence requires closure under reduction contexts, in the general case it is hard to prove directly that an equation holds. Still, in some cases we are able to do it.

For example, to prove $(\nu\, a)a.P \cong \mathbf{0}$, we can say that both processes have no barbs and no reductions, but then we have to prove that for all C, $C[(\nu\, a)a.P] \cong C[\mathbf{0}]$. In this case, it can be done by a simple induction on the structure of C.

1. **(Base Case)** P and Q have the same barbs: $(\nu\, a)a.P \not\Downarrow_d$ and $\mathbf{0} \not\Downarrow_d$ for all d

2. **(Base Case)** the property is preserved by reductions: $(\nu\, a)a.P \not\longrightarrow$ and $\mathbf{0} \not\longrightarrow$.

3. **(Inductive Case)** $C[P]$ and $C[Q]$ have the same barbs: $C[(\nu\, a)a.P] \Downarrow_d$ iff $C[\mathbf{0}] \Downarrow_d$ for all d.

   Let $C = (\nu b_1, \ldots, b_n)(R \mid \_)$. Then $C[(\nu\, a)a.P] \Downarrow_d$ means $(\nu b_1, \ldots, b_n)R \Downarrow_d$, which means that $C[\mathbf{0}] \Downarrow_d$.

4. **(Inductive Case)** If $C[(\nu\, a)a.P] \longrightarrow Q$, then $C[\mathbf{0}] \longrightarrow^* Q'$ and $Q \cong Q'$.

   Let $C = (\nu b_1, \ldots, b_n)(R \mid \_)$. Then $C[(\nu\, a)a.P] \longrightarrow R'$ means

   $R' = (\nu b_1, \ldots, b_n)(R'' \mid (\nu\, a)a.P)$ with $R \longrightarrow R''$.

   Hence $C[\mathbf{0}] \longrightarrow (\nu b_1, \ldots, b_n)(R'' \mid \mathbf{0})$ with $R \longrightarrow R''$.

   Thus $(\nu b_1, \ldots, b_n)(R'' \mid (\nu\, a)a.P) \cong (\nu b_1, \ldots, b_n)(R'' \mid \mathbf{0})$.

Asynchrony

We now look at the more characteristic law of the asynchronous π-calculus: the *asynchrony law*. It captures the essence of asynchronous communication, stating that it is not possible to observe the presence of a communication buffer:

$$a(x).\overline{a}\langle x\rangle \cong \mathbf{0}$$

Since $\cong$ is a congruence, we immediately have, for any P:

$$a(x).\overline{a}\langle x\rangle \mid P \cong P$$

The intuition is that the only way to probe $a(x).\overline{a}\langle x\rangle$ is by sending it a message $\overline{a}\langle b\rangle$ (for some b), but since the effect of receiving the message is exactly to produce it again, nothing changes from the observer viewpoint:

$$a(x).\overline{a}\langle x\rangle \mid P \mid \overline{a}\langle b\rangle \longrightarrow \mathbf{0} \mid P \mid \overline{a}\langle b\rangle$$

This law is useful from a practical point of view. For example, it guarantees that if an optimisation introduces buffering to implement channel communication, the overall behaviour is not affected.

Equators

An *equator* between two channels a and b, is a process that forwards all messages for a on b, and viceversa, making a and b in some sense "equivalent". For example,

$$\mathbf{EQ}(x,y) \stackrel{\text{df}}{=} !x(z).\overline{y}\langle z\rangle \,|\, !y(z).\overline{x}\langle z\rangle$$

$$\overline{a}\langle d\rangle \,|\, \mathbf{EQ}\langle a,b\rangle \longrightarrow \overline{b}\langle d\rangle \,|\, \mathbf{EQ}\langle a,b\rangle \longrightarrow \overline{a}\langle d\rangle \,|\, \mathbf{EQ}\langle a,b\rangle \longrightarrow \ldots$$

In fact, we have that

$$\overline{a}\langle d\rangle \,|\, \mathbf{EQ}\langle a,b\rangle \cong \overline{b}\langle d\rangle \,|\, \mathbf{EQ}\langle a,b\rangle$$

In the monadic asynchronous π-calculus we also have the law

$$\overline{c}\langle a\rangle \,|\, \mathbf{EQ}\langle a,b\rangle \cong \overline{c}\langle b\rangle \,|\, \mathbf{EQ}\langle a,b\rangle$$

which does not hold in the polyadic case, because for example:

$$\overline{a}\langle d,c\rangle \,|\, \mathbf{EQ}\langle a,b\rangle \ncong \overline{b}\langle d,c\rangle \,|\, \mathbf{EQ}\langle a,b\rangle$$

To recover this law, we should introduce an equator with two parameters, and similarly for all the (infinite) possible arities. . .

Equational laws for the (polyadic) asynchronous π-calculus

Here are some equational laws that can be used to prove reduction congruence in both the polyadic and monadic asynchronous π-calculus:

$$P \equiv Q \Longrightarrow P \cong Q \qquad (\equiv)$$

$$P \stackrel{*}{\longrightarrow} Q \ \mathit{and}\ Q \stackrel{*}{\longrightarrow} P \Longrightarrow P \cong Q \qquad \text{(Cyclic)}$$

$$P \cong Q \Longrightarrow !P \cong !Q \qquad (!)$$

$$!(P \mid \ldots \mid P) \cong !P \qquad \text{(Multi!)}$$

$$!(P \mid Q) \cong !P \mid !Q \qquad \text{(Par!)}$$

$$!\mathbf{0} \cong \mathbf{0} \qquad (!\ 0)$$

$$(\nu\, a)a(\widetilde{x}).P \cong \mathbf{0} \qquad \text{(In 0)}$$

$$(\nu\, a)\overline{a}\langle\widetilde{c}\rangle \cong \mathbf{0} \qquad \text{(Out 0)}$$

$$P \oplus P \cong P \qquad \text{(Plus)}$$

$$(\nu\, a)(\overline{a}\langle c_1, \ldots, c_n\rangle \mid a(x_1, \ldots, x_n).P) \cong (\nu\, a)(P\{\widetilde{c}/\widetilde{x}\}) \qquad \text{(Tau)}$$

$$a(x_1, \ldots, x_n).\overline{a}\langle x_1, \ldots, x_n\rangle \cong \mathbf{0} \qquad \text{(Asynchrony)}$$

Remember that the following (EQ) law holds only for the monadic asynchronous π-calculus.

$$\overline{c}\langle a\rangle \mid \mathbf{EQ}\langle a, b\rangle \cong \overline{c}\langle b\rangle \mid \mathbf{EQ}\langle a, b\rangle \qquad \text{(EQ)}$$

Question 3 *Show* $\overline{a}\langle d\rangle \mid \mathbf{EQ}\langle a, b\rangle \cong \overline{b}\langle d\rangle \mid \mathbf{EQ}\langle a, b\rangle$.

Question 4 *(very advanced) Prove all of the above equational laws.*

Reduction congruence in the full π-calculus

The full π-calculus adds output continuation, polyadicity and matching to the asynchronous calculus. Matching can be read as "if $u = v$ then P else 0".

$$P, Q ::= \mathbf{0} \quad | \quad P \,|\, Q \quad | \quad (\nu\, a)P \quad | \quad !P \quad | \quad \overline{u}\langle\widetilde{v}\rangle.P \quad | \quad u(\widetilde{x}).P \quad | \quad [u = v]P$$

Functions fn and fv are extended in the obvious way (there are no new binders). We have new structural congruence rules for matching and choice

$$\text{(Match)} \quad [a = a]P \equiv P \qquad \text{(Mismatch)} \quad [a = b]P \equiv \mathbf{0}$$

Our definition of reduction congruence can be used also for any π-calculus. All of the laws we have seen are still valid, except for the asynchrony and the equator laws, which are invalidated by the new operators.

- With output continuation:

  $$a(x).\overline{a}\langle x\rangle \ncong \mathbf{0}$$

  The context $C = \overline{a}\langle b\rangle.\overline{b}$ is such that $C[a(x).\overline{a}\langle x\rangle] \Downarrow_b$ but $C[\mathbf{0}] \not\Downarrow_b$.

- With matching, in the monadic case:

  $$\overline{c}\langle a\rangle \,|\, \mathbf{EQ}\langle a, b\rangle \ncong \overline{c}\langle b\rangle \,|\, \mathbf{EQ}\langle a, b\rangle$$

  The context $C = c(x).[x = a].\overline{b}$ is such that $C[\overline{c}\langle a\rangle \,|\, \mathbf{EQ}\langle a, b\rangle] \Downarrow_b$ but $C[\overline{c}\langle b\rangle \,|\, \mathbf{EQ}\langle a, b\rangle] \not\Downarrow_b$.

Quiz

Show that the following equivalences hold, using the laws seen during the lectures:

1. $(\nu\, b, c)(\overline{a}\langle b\rangle \mid c(x).P) \;\cong\; (\nu\, d)\overline{a}\langle d\rangle$
2. $a(x).\overline{a}\langle x\rangle \mid !(Q \mid Q) \;\cong\; !Q$
3. $(\nu\, c, b)(\mathbf{FW}\langle c, b\rangle \mid \mathbf{FW}\langle b, e\rangle \mid \overline{c}\langle d\rangle) \;\cong\; \overline{e}\langle d\rangle$ with $\mathbf{FW}\langle c, b\rangle = c(x)\overline{b}\langle x\rangle$.

Quiz

Show that the following equivalences hold, using the laws seen during the lectures:

1. $(\nu\, b, c)(\overline{a}\langle b\rangle \,|\, c(x).P) \;\cong\; (\nu\, d)\overline{a}\langle d\rangle$
2. $a(x).\overline{a}\langle x\rangle \,|\, !(Q \,|\, Q) \;\cong\; !Q$
3. $(\nu\, c, b)(\mathbf{FW}\langle c, b\rangle \,|\, \mathbf{FW}\langle b, e\rangle \,|\, \overline{c}\langle d\rangle) \;\cong\; \overline{e}\langle d\rangle$ with $\mathbf{FW}\langle c, b\rangle = c(x)\overline{b}\langle x\rangle$.

1.

$$\begin{aligned}
(\nu\, b, c)(\overline{a}\langle b\rangle \,|\, c(x).P) &\cong (\nu\, d)(\overline{a}\langle d\rangle) \,|\, (\nu\, c)c(x).P && (\equiv)\\
&\cong (\nu\, d)\overline{a}\langle d\rangle \,|\, \mathbf{0} && (\text{In } 0)\\
&\cong (\nu\, d)\overline{a}\langle d\rangle && (\equiv)
\end{aligned}$$

2.

$$\begin{aligned}
!Q &\cong !(Q \,|\, Q) && (\text{Multi!})\\
&\cong \mathbf{0} \,|\, !(Q \,|\, Q) && (\equiv)\\
&\cong a(x).\overline{a}\langle x\rangle \,|\, !(Q \,|\, Q) && (\text{Asynchrony})
\end{aligned}$$

3.

$$\begin{aligned}
(\nu\, c, b)(\mathbf{FW}\langle c, b\rangle \,|\, \mathbf{FW}\langle b, e\rangle \,|\, \overline{c}\langle d\rangle) &\cong\; (\nu\, b)(\mathbf{FW}\langle b, e\rangle \,|\, \overline{b}\langle d\rangle) && (\text{Tau})\\
&\cong\; \overline{e}\langle d\rangle && (\text{Tau})
\end{aligned}$$

Quiz

1. Find all the weak barbs of the processes below:

   (a) $(\nu\, b)(a(x).\overline{x}\langle b\rangle) \mid (\nu\, b)(\overline{a}\langle b\rangle \mid b(x))$

   (b) $!(\nu\, a)(\overline{a} \mid !(!b.\overline{c} \mid !a.\overline{b}))$

   (c) $(\nu\, b)(\overline{b}\langle a, b\rangle \mid !b(x, y).(\nu\, a)(\overline{y}\langle a, x\rangle)) \mid \overline{c}\langle a\rangle$

2. Show the inequality $P \not\cong Q$ where

$$P \stackrel{\text{df}}{=} (\nu\, a, b)(\overline{b}\langle a\rangle \mid b(x).(\overline{a}\langle y\rangle \mid \overline{y}\langle c\rangle)) \mid \overline{x}\langle y, z\rangle \mid y(y, z).z(x).\overline{x}$$

$$Q \stackrel{\text{df}}{=} \overline{y}\langle c\rangle \mid \overline{x}\langle y, z\rangle \mid y(y, z).y(x).\overline{x}$$

   (Hint: you have to find a closing substitution σ such that $P\sigma \not\cong Q\sigma$.)

Quiz: Answer

Find all the weak barbs of the processes below:

1. $(\nu\, b)(a(x).\overline{x}\langle b\rangle) \,|\, (\nu\, b)(\overline{a}\langle b\rangle \,|\, b(x))$

2. $!(\nu\, a)(\overline{a} \,|\, !(!b.\overline{c} \,|\, !a.\overline{b}))$

3. $(\nu\, b)(\overline{b}\langle a, b\rangle \,|\, !b(x, y).(\nu\, a)(\overline{y}\langle a, x\rangle)) \,|\, \overline{c}\langle a\rangle$

1. The barbs of $P = (\nu\, b)(a(x).\overline{x}\langle b\rangle) \,|\, (\nu\, b)(\overline{a}\langle b\rangle \,|\, b(x))$ are $\{a\}$, since $P \downarrow_a$ and $P \longrightarrow (\nu\, b, c)(\overline{c}\langle b\rangle \,|\, c(x))$, which has no barbs.

2. The barbs of $P = !(\nu\, a)(\overline{a} \,|\, !(!b.\overline{c} \,|\, !a.\overline{b}))$ are $\{b, c\}$ since $P \longrightarrow P' = P \,|\, (\nu\, a)(!(!b.\overline{c} \,|\, !a.\overline{b})) \,|\, \overline{b}$ and $P' \downarrow_b$, and
   also $P' \longrightarrow P'' = P \,|\, (\nu\, a)(!(!b.\overline{c} \,|\, !a.\overline{b})) \,|\, \overline{c}$ and $P'' \downarrow_c$, but $P'' \not\longrightarrow$.

3. The barbs of $P = (\nu\, b)(\overline{b}\langle a, b\rangle \,|\, !b(x, y).(\nu\, a)(\overline{y}\langle a, x\rangle)) \,|\, \overline{c}\langle a\rangle$ are $\{c, a\}$, since $P \downarrow_c$ and $P \stackrel{*}{\longrightarrow} P' = (\nu\, b)(!b(x, y).(\nu\, a)(\overline{y}\langle a, x\rangle)) \,|\, \overline{c}\langle a\rangle \,|\, (\nu\, d, e)\overline{a}\langle d, e\rangle$, so $P' \downarrow_a$.

Quiz: Answer 2

Show the inequality $P \not\cong Q$ where

$$P \stackrel{\text{df}}{=} (\nu\, a, b)(\overline{b}\langle a\rangle \,|\, b(x).(\overline{a}\langle y\rangle \,|\, \overline{y}\langle c\rangle)) \,|\, \overline{x}\langle y, z\rangle \,|\, y(y, z).z(x).\overline{x}$$

$$Q \stackrel{\text{df}}{=} \overline{y}\langle c\rangle \,|\, \overline{x}\langle y, z\rangle \,|\, y(y, z).y(x).\overline{x}$$

(Hint: you have to find a closing substitution σ such that $P\sigma \not\cong Q\sigma$.)

The substitution $\sigma = \{{}^{d}/_{x}, {}^{d}/_{y}, {}^{e}/_{z}\}$ proves that $P \not\cong Q$. In fact,

$$P\sigma \stackrel{*}{\longrightarrow} (\nu\, a)(\overline{a}\langle d\rangle \,|\, \overline{d}\langle c\rangle) \,|\, e(x).\overline{x} \not\Downarrow_{c}$$

$$Q\sigma \stackrel{*}{\longrightarrow} \overline{c} \downarrow_{c}$$