Concurrency theory

name passing, contextual equivalences

Francesco Zappa Nardelli

INRIA Rocquencourt, MOSCOVA research team

francesco.zappa_nardelli@inria.fr

together with

Frank Valencia (INRIA Futurs) Catuscia Palamidessi (INRIA Futurs) Roberto Amadio (PPS)

One remark

When we write the CCS term

$$(\boldsymbol{\nu} b)(a.(b \; \Big|\Big| \; c) + \tau.(b \; \Big|\Big| \; \overline{b}.c))$$

the names a, b, and c are distinct. No side conditions are needed to prove

$$(\boldsymbol{\nu} b)(a.(b \; \Big|\Big| \; c) + \tau.(b \; \Big|\Big| \; \overline{b}.c)) \; \sim \; \tau.\tau.c + a.c \; .$$

What we have seen

- A syntax for visible actions, synchronisation, parallel composition.
- Executing programs: LTS, reduction semantics.
- Equivalences: from linear time to branching time. Bisimulation as the reference equivalence. Ignoring τ transitions: weak equivalences.
- Proof techniques, axiomatisations, Hennessy-Milner logic (more examples to come).

Memento: CCS, reduction semantics

We define reduction, denoted $\rightarrowtriangle$, by

$$a.P \parallel \overline{a}.Q \ \rightarrowtriangle \ P \parallel Q$$

$$\frac{P \ \rightarrowtriangle \ P'}{P \parallel Q \ \rightarrowtriangle \ P' \parallel Q} \qquad \frac{P \ \rightarrowtriangle \ P'}{(\boldsymbol{\nu} x)P \ \rightarrowtriangle \ (\boldsymbol{\nu} x)P'} \qquad \frac{P \equiv P' \rightarrowtriangle Q' \equiv Q}{P \rightarrowtriangle Q}$$

where, the structural congruence relation, denoted $\equiv$, is defined as:

$$P \parallel Q \equiv Q \parallel P \qquad (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$$

$$P \parallel \mathbf{0} \equiv P \qquad !P \equiv P \parallel !P \qquad (\boldsymbol{\nu} a)P \parallel Q \equiv (\boldsymbol{\nu} a)(P \parallel Q) \text{ if } a \notin \mathrm{fn}(Q)$$

Theorem $P \ \rightarrowtriangle \ Q$ iff $P \xrightarrow{\tau} \equiv Q$.

Value passing

Names can be interpreted as *channel names*: allow channels to carry values, so instead of pure outputs $\overline{a}.P$ and inputs $a.P$ allow e.g.: $\overline{a}\langle 15,3\rangle.P$ and $a(x,y).Q$.

Value 6 being sent along channel x:

$$\overline{x}\langle 6\rangle \;||\; x(u).\overline{y}\langle u\rangle \;\rightharpoonup\; (\overline{y}\langle u\rangle)\{^6/_u\} \;=\; \overline{y}\langle 6\rangle$$

Restricted names are different from all others:

$$\begin{array}{ccc}
\overline{x}\langle 5\rangle \;||\; (\boldsymbol{\nu} x)(\overline{x}\langle 6\rangle \;||\; x(u).\overline{y}\langle u\rangle) & \rightharpoonup & \overline{x}\langle 5\rangle \;||\; (\boldsymbol{\nu} x)(\overline{y}\langle 6\rangle) \\
\equiv & & \equiv \\
\overline{x}\langle 5\rangle \;||\; (\boldsymbol{\nu} x')(\overline{x'}\langle 6\rangle \;||\; x'(u).\overline{y}\langle u\rangle) & \rightharpoonup & \overline{x}\langle 5\rangle \;||\; (\boldsymbol{\nu} x'')(\overline{y}\langle 6\rangle)
\end{array}$$

(note that we are working with alpha equivalence classes).

Exercise

Program a server that increments the value it receives.

$$!x(u).\overline{x}\langle u+1\rangle$$

Argh!!! This server exhibits exactly the problems we want to avoid when programming concurrent systems:

$$\overline{x}\langle 3\rangle.x(u).P \;\|\; \overline{x}\langle 7\rangle.x(v).Q \;\|\; !x(u).\overline{x}\langle u+1\rangle \;\rightarrowtriangle \ldots$$

$$\ldots \rightarrowtriangle P\{^{8}/_{u}\} \;\|\; Q\{^{4}/_{u}\} \;\|\; !x(u).\overline{x}\langle u+1\rangle$$

Ideas...

Allow those values to include channel names.

A new implementation for the server:

$$!x(u, r).\overline{r}\langle u + 1\rangle$$

This server prevents confusion provided that the return channels are distinct.

How can we guarantee that the return channels are distinct?

Idea: use restriction, and communicate restricted names...

The π-calculus

1. A name received on a channel can then be used itself as a channel name for output or input — here y is received on x and the used to output 7:

$$\overline{x}\langle y\rangle \;\Big|\Big|\; x(u).\overline{u}\langle 7\rangle \;\rightharpoondown\; \overline{y}\langle 7\rangle$$

2. A restricted name can be sent outside its original scope. Here y is sent on channel x outside the scope of the $(\boldsymbol{\nu}y)$ binder, which must therefore be moved (with care, to avoid capture of free instances of y). This is *scope extrusion*:

$$\begin{aligned}(\boldsymbol{\nu}y)(\overline{x}\langle y\rangle \;||\; y(v).P) \;||\; x(u).\overline{u}\langle 7\rangle \quad &\rightharpoondown \quad (\boldsymbol{\nu}y)(y(v).P \;||\; \overline{y}\langle 7\rangle)\\ &\rightharpoondown \quad (\boldsymbol{\nu}y)(P\{{}^{7}/_{v}\})\end{aligned}$$

The (simplest) π-calculus

Syntax:

$$
\begin{array}{rcll}
P, Q & ::= & \mathbf{0} & \text{nil} \\
& | & P \parallel Q & \text{parallel composition of } P \text{ and } Q \\
& | & \overline{c}\langle v\rangle.P & \text{output } v \text{ on channel } c \text{ and resume as } P \\
& | & c(x).P & \text{input from channel } c \\
& | & (\boldsymbol{\nu} x)P & \text{new channel name creation} \\
& | & !P & \text{replication}
\end{array}
$$

Free names (alpha-conversion follows accordingly):

$$
\begin{array}{rclrcl}
\mathrm{fn}(\mathbf{0}) & = & \emptyset & \mathrm{fn}(P \parallel Q) & = & \mathrm{fn}(P) \cup \mathrm{fn}(Q) \\
\mathrm{fn}(\overline{c}\langle v\rangle.P) & = & \{c, v\} \cup \mathrm{fn}(P) & \mathrm{fn}(c(x).P) & = & (\mathrm{fn}(P) \setminus \{x\}) \cup \{c\} \\
\mathrm{fn}((\boldsymbol{\nu} x)P) & = & \mathrm{fn}(P) \setminus \{x\} & \mathrm{fn}(!P) & = & \mathrm{fn}(P)
\end{array}
$$

π-calculus, reduction semantics

Structural congruence:

$$P \parallel 0 \equiv P \qquad\qquad P \parallel Q \equiv Q \parallel P$$

$$(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) \qquad\qquad !P \equiv P \parallel !P$$

$$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P$$

$$P \Big\| (\boldsymbol{\nu} x)Q \equiv (\boldsymbol{\nu} x)(P \Big\| Q) \text{ if } x \notin \mathrm{fn}(P)$$

Reduction rules:

$$\overline{c}\langle v\rangle.P \parallel c(x).Q \rightarrow P \parallel Q\{^{v}/_{x}\}$$

$$\frac{P \rightarrow P'}{P \parallel Q \rightarrow P' \parallel Q} \qquad \frac{P \rightarrow P'}{(\boldsymbol{\nu} x)P \rightarrow (\boldsymbol{\nu} x)P'} \qquad \frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$$

Expressiveness

A small calculus (and the semantics only involves name-for-name substitution, not term-for-variable substitution), but very expressive:

- encoding data structures
- encoding functions as processes (Milner, Sangiorgi)
- encoding higher-order π (Sangiorgi)
- encoding synchronous communication with asynchronous (Honda/Tokoro, Boudol)
- encoding polyadic communication with monadic (Quaglia, Walker)
- encoding choice (or not) (Nestmann, Palamidessi)
- ...

Example: polyadic with monadic

Let us extend our notion of monadic channels, which carry exactly one name, to polyadic channels, which carry a vector of names, i.e.

$$\begin{array}{lcll} P & ::= & \overline{x}\langle y_1, ..., y_n \rangle.P & \text{output} \\ & \mid & x(y_1, ..., y_n).P & \text{input} \end{array}$$

with the main reduction rule being:

$$\overline{x}\langle y_1, ..., y_n \rangle P \;||\; x(z_1, ..., z_n).Q \;\rightarrow\; P \;||\; Q\{^{y_1, \cdots y_n}/_{z_1, \ldots, z_n}\}$$

Is there an encoding from polyadic to monadic channels?

Polyadic with monadic, ctd.

We might try:

$$\begin{array}{rcl} [[\overline{x}\langle y_1, ..., y_n\rangle.P]] & = & \overline{x}\langle y_1\rangle.\ldots.\overline{x}\langle y_n\rangle.[[P]] \\ [[x(y_1, ..., y_n).P]] & = & x(y_1).\ldots.x(y_n).[[P]] \end{array}$$

but this is broken! Why?

The right approach is use new binding:

$$\begin{array}{rcl} [[\overline{x}\langle y_1, ..., y_n\rangle.P]] & = & (\boldsymbol{\nu} z)(\overline{x}\langle z\rangle.\overline{z}\langle y_1\rangle.\ldots.\overline{z}\langle y_n\rangle.[[P]]) \\ [[x(y_1, ..., y_n).P]] & = & x(z).z(y_1).\ldots.z(y_n).[[P]] \end{array}$$

where $z \notin \mathrm{fn}(P)$ (why?). (We also need some well-sorted assumptions.)

Recursion

Alternative to replication: recursive definition of processes.

Recursive definition (in CCS we used to write $K(\tilde{x}) = P$):

$$K \;=\; (\tilde{x}).P$$

Constant application:

$$K\lfloor a \rfloor$$

Reduction rule:

$$\frac{K = (\tilde{x}).P}{K\lfloor \tilde{a} \rfloor \;\longrightarrow\; P\{\tilde{a}/\tilde{x}\}}$$

Recursion vs. Replication

Theorem Any process involving recursive definitions is representable using replication, and conversely replication is redundant in presence of recursion.

The proof requires some techniques we have not seen, but...

Intuition: given

$$F = (\tilde{x}).P$$

where P may contain recursive calls to F of the form $F\lfloor\tilde{z}\rfloor$, we may replace the RHS with the following process abstraction containing no mention of F:

$$(\tilde{x}).(\boldsymbol{\nu} f)(\overline{f}\langle\tilde{x}\rangle \;\big|\big|\; !f(\tilde{x}).P')$$

where P' is obtained by replacing every occurrence of $F\lfloor\tilde{z}\rfloor$ by $\overline{f}\langle\tilde{z}\rangle$ in P, and f is fresh for P.

Data as processes: booleans

Consider the truth-values $\{\mathrm{True}, \mathrm{False}\}$. Consider the abstractions:

$$T \;=\; (x).x(t,f).\bar{t}\langle\rangle \quad \text{and} \quad F \;=\; (x).x(t,f).\overline{f}\langle\rangle$$

These represent a *located copy* of a truth-value at x. The process

$$R \;=\; (\boldsymbol{\nu} t)(\boldsymbol{\nu} f)\bar{b}\langle t,f\rangle.(t().P \;\|\; f().Q)$$

where $t, f \notin \mathrm{fn}(P, Q)$ can test for a truth-value at x and behave accordingly as P or Q:

$$R \;\|\; T\lfloor b\rfloor \twoheadrightarrow P \;\|\; (\boldsymbol{\nu} t, f)f().Q$$

The term obtained behaves as P because the thread $(\boldsymbol{\nu} t, f)f().Q$ is deadlocked.

Data as processes: integers

Using a unary representation.

$$[[k]] \;=\; (x).x(z,o).(\overline{o}\langle\rangle)^k.\overline{z}\langle\rangle$$

where $(\overline{o}\langle\rangle)^k$ abbreviates $\overline{o}\langle\rangle.\overline{o}\langle\rangle.\ldots.\overline{o}\langle\rangle$ (k occurrences).

Operations on integers can be expressed as processes. For instance,

$$\mathrm{succ} \;=\; (x,y).!x(z,o).\overline{o}\langle\rangle.\overline{y}\langle z,o\rangle$$

Which is the role of the final output on z? (Hint: omit it, and try to define the test for zero).

Another representation for integers

```
type Nat = zero | succ Nat
```

Define:

$$[\![\texttt{zero}]\!] \quad = \quad (x).!x(z,s).\overline{z}\langle\rangle$$
$$[\![\texttt{succ}]\!] \quad = \quad (x,y).!x(z,s).\overline{s}\langle y\rangle$$

and for each e of type `Nat`:

$$[\![\texttt{succ}\ e]\!] = (x).(\boldsymbol{\nu} y)([\![\texttt{succ}]\!]\lfloor x,y\rfloor \ \Big|\Big|\ [\![e]\!]\lfloor y\rfloor)$$

This approach generalises to arbitrary datatypes.

A step backward: defining a language

Recipe:

1. define the *syntax* of the language (that is, specify what a program is);
2. define its *reduction semantics* (that is, specify how programs are executed);
3. define when *two terms are equivalent* (via LTS + bisimulation?).

Lifting CCS techniques to name-passing is not straightforward

Actually, the original paper on pi-calculus defines *two* LTSs (excerpts):

Early LTS

$$\overline{x}\langle v\rangle.P \xrightarrow{\overline{x}\langle v\rangle} P$$

$$x(y).P \xrightarrow{x(v)} \{^v/_y\}P$$

$$\frac{P \xrightarrow{\overline{x}\langle v\rangle} P' \qquad Q \xrightarrow{x(v)} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

Late LTS

$$\overline{x}\langle v\rangle.P \xrightarrow{\overline{x}\langle v\rangle} P$$

$$x(y).P \xrightarrow{x(y)} P$$

$$\frac{P \xrightarrow{\overline{x}\langle v\rangle} P' \qquad Q \xrightarrow{x(y)} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel \{^v/_y\}Q'}$$

These LTSs define the same τ-transitions, where is the problem?

Problem

Definition: Weak bisimilarity, denoted $\approx$, is the largest symmetric relation such that whenever $P \approx Q$ and $P \xrightarrow{\ell} P'$ there exists Q' such that $Q \stackrel{\hat{\ell}}{\Longrightarrow} Q'$ and $P' \approx Q'$.

But the bisimilarity built on top of them observe all the labels: do the resulting bisimilarities coincide? No!

Which is the right one? Which is the role of the LTS?

Equivalent?

Suppose that P and Q are equivalent (in symbols: $P \simeq Q$).

Which properties do we expect?

Preservation under contexts For all contexts $C[-]$, we have $C[P] \simeq C[Q]$;

Same observations If $P \downarrow x$ then $Q \downarrow x$, where $P \downarrow x$ means that we can *observe* x at P (or P *can do* x);

Preservation of reductions P and Q must mimic their reduction steps (that is, they realise the same nondeterministic choices).

Formally

A relation $\mathcal{R}$ between processes is

preserved by contexts: if $P \mathcal{R} Q$ implies $C[P] \mathcal{R} C[Q]$ for all contexts $C[-]$.

barb preserving: if $P \mathcal{R} Q$ and $P \downarrow x$ imply $Q \Downarrow x$, where $P \Downarrow x$ holds if there exists P' such that $P \rightarrowtriangle^* P'$ and $P' \downarrow x$, while

$$P \equiv (\boldsymbol{\nu}\tilde{n})(\overline{x}\langle y\rangle.P' \,\big|\big|\, P'') \text{ or } P \equiv (\boldsymbol{\nu}\tilde{n})(x(u).P' \,\big|\big|\, P'') \text{ for } x \notin \tilde{n} \;;$$

reduction closed: if $P \mathcal{R} Q$ and $P \rightarrowtriangle P'$, imply that there is a Q' such that $Q \rightarrowtriangle^* Q'$ and $P' \mathcal{R} Q'$ ($\rightarrowtriangle^*$ is the reflexive and transitive closure of $\rightarrowtriangle$).

Reduction-closed barbed congruence

Let reduction barbed congruence, denoted $\simeq$, be the largest symmetric relation over processes that is preserved by contexts, barb preserving, and reduction closed.

Remark: reduction barbed congruence is a weak equivalence: the number of internal reduction steps is not important in the bisimulation game imposed by "reduction closed".

Example: local names are different from global names

Show that in general

$$(\boldsymbol{\nu} x)!P \;\not\approx\; !(\boldsymbol{\nu} x)P$$

Intuition: the copies of P in $(\boldsymbol{\nu} x)!P$ can interact over x, while the copies of $(\boldsymbol{\nu} x)P$ cannot.

We need a process that interacts with another copy of itself over x, but that cannot interact with itself over x. Take

$$P = \overline{x}\langle\rangle \oplus x().\overline{b}\langle\rangle$$

where $Q_1 \oplus Q_2 = (\boldsymbol{\nu} w)(\overline{w}\langle\rangle \;\|\; w().Q_1 \;\|\; w().Q_2$.

We have that $(\boldsymbol{\nu} x)!P \Downarrow b$, while $!(\boldsymbol{\nu} x)P \not\Downarrow b$.

Some equivalences (?)

Compare the processes

1. $P = \overline{x}\langle y \rangle$ and $Q = \mathbf{0}$

2. $P = \overline{a}\langle x \rangle$ and $Q = \overline{a}\langle z \rangle$

3. $P = (\boldsymbol{\nu} x)\overline{x}\langle\rangle.R$ and $Q = \mathbf{0}$

4. $P = (\boldsymbol{\nu} x)(\overline{x}\langle y \rangle.R_1 \parallel x(z).R_2)$ and $Q = (\boldsymbol{\nu} x)(R_1 \parallel R_2\{^y/_z\})$

Argh... we need other proof techniques to show that processes are equivalent!

Remark: we can reformulate *barb preservation* as "if $P \mathcal{R} Q$ and $P \Downarrow x$ imply $Q \Downarrow x$". This is sometimes useful...

The role of bisimilarity

Observation: the definition of bisimilarity does not involve a universal quantification over all contexts!

Question: is there any relationship between (weak) bisimilarity and reduction barbed congruence?

Theorem:

1. $P \approx Q$ implies $P \simeq Q$ (soundness of bisimilarity);

2. $P \simeq Q$ implies $P \approx Q$ (completeness of bisimilarity).

Point 2. does not hold in general.
Point 1. ought to hold (otherwise your LTS/bisimilarity is very odd!).

Soundness and completeness for a fragment of CCS

Consider the fragment of CCS without sums and replication:

$$a.P \xrightarrow{a} P \qquad \overline{a}.P \xrightarrow{\overline{a}} P \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\frac{P \xrightarrow{\ell} P'}{P \parallel Q \xrightarrow{\ell} P' \parallel Q} \qquad \frac{P \xrightarrow{\ell} P' \quad a \notin \mathrm{fn}(\ell)}{(\boldsymbol{\nu} a)P \xrightarrow{\ell} (\boldsymbol{\nu} a)P'}$$

symmetric rules omitted.

Barbs are defined as $P \downarrow a$ iff $P \equiv (\boldsymbol{\nu}\tilde{n})(a.P' \parallel P'')$ or $P \equiv (\boldsymbol{\nu}\tilde{n})(\overline{a}.P' \parallel P'')$ for $a \notin \tilde{n}$.

Soundness of weak bisimilarity: $P \approx Q$ implies $P \simeq Q$.

Proof We show that $\approx$ is contextual, barb preserving, and reduction closed.

Contextuality of $\approx$ can be shown by induction on the structure of the contexts, and by case analysis of the possible interactions between the processes and the contexts. (Congurence of bisimilarity).

Suppose that $P \approx Q$ and $P \downarrow a$. Then $P \equiv (\boldsymbol{\nu}\tilde{n})(a.P_1 \parallel P_2)$, with $a \notin \tilde{n}$. We derive $P \xrightarrow{a} (\boldsymbol{\nu}\tilde{n})(P_1 \parallel P_2)$. Since $P \approx Q$, there exists Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$, that is $Q \xrightarrow{\tau}^{*} Q'' \xrightarrow{a} \ldots$. But Q'' must be of the form $(\boldsymbol{\nu}\tilde{m})(a.Q_1 \parallel Q_2)$ with $a \notin \tilde{m}$. This implies that $Q'' \downarrow a$, and in turn $Q \Downarrow a$, as required.

Suppose that $P \approx Q$ and $P \rightarrowtriangle P'$. We have that $P \xrightarrow{\tau} P'' \equiv P'$. Since $P \approx Q$, there exists Q' such that $Q \xrightarrow{\tau}^{*} Q'$ and $P' \equiv P'' \approx Q'$. Since $Q \xrightarrow{\tau}^{*} Q'$ it holds that $Q \rightarrowtriangle^{*} Q'$. Since $P' \equiv P''$ implies $P' \approx P''$, by transitivity of $\approx$ we conclude $P' \approx Q'$, as required. $\square$

Completeness of weak bisimilarity: $P \simeq Q$ implies $P \approx Q$.

Proof We show that $\simeq$ is a bisimulation.

Suppose that $P \simeq Q$ and $P \xrightarrow{a} P'$ (the case $P \simeq Q$ and $P \xrightarrow{\tau} P'$ is easy). Let

$$\begin{array}{lcl} C_a[-] & = & - \parallel \overline{a}.d \\ C_{\overline{a}}[-] & = & - \parallel a.d \end{array} \qquad \begin{array}{rcl} Flip & = & \overline{d}.(o \oplus f) \\ -_1 \oplus -_2 & = & (\boldsymbol{\nu} z)(z.\,-_1 \parallel z.\,-_2 \parallel \overline{z}) \end{array}$$

where the names z, o, f, d are *fresh* for P and Q.

Lemma 1. $C_a[P] \twoheadrightarrow^* P' \parallel d$ if and only if $P \stackrel{a}{\Longrightarrow} P'$. Similarly for $C_{\overline{a}}[-]$.

Since $\simeq$ is contextual, we have $C_a[P] \parallel Flip \simeq C_a[Q] \parallel Flip$. By Lemma 1. we have $C_a[P] \parallel Flip \twoheadrightarrow^* P_1 \equiv P' \parallel o \parallel (\boldsymbol{\nu} z)z.f$.

Lemma 2. If $P \simeq Q$ and $P \twoheadrightarrow^* P'$ then there exists Q' such that $Q \twoheadrightarrow^* Q'$ and $P' \simeq Q'$.

By Lemma 2. there exists Q_1 such that $C_a[Q] \parallel Flip \twoheadrightarrow^* Q_1$ and $P_1 \simeq Q_1$. Now, $P_1 \downarrow o$ and $P_1 \not\downarrow f$. Since $\simeq$ is barb preserving, we have $Q_1 \Downarrow o$ and $Q_1 \not\Downarrow f$. The absence of the barb f implies that the $\oplus$ operator reduced, and in turn that the d action has been consumed: this can only occur if Q realised the a action. Thus we can conclude $Q_1 \equiv Q' \parallel o \parallel (\boldsymbol{\nu} z)z.f$, and by Lemma 1. we also have $Q \stackrel{a}{\Longrightarrow} Q'$.

It remains to show that $P' \simeq Q'$.

Lemma 3. $(\boldsymbol{\nu} z)z.P \simeq \mathbf{0}$.

Since $P_1 \simeq Q_1$ and $\simeq$ is contextual, we have $(\boldsymbol{\nu} o)P_1 \simeq (\boldsymbol{\nu} o)Q_1$. By Lemma 3., we have

$$P' \simeq P' \parallel (\boldsymbol{\nu} o)o \parallel (\boldsymbol{\nu} z)z.f \;\equiv\; (\boldsymbol{\nu} o)P_1 \;\simeq\; (\boldsymbol{\nu} o)Q_1 \;\equiv\; Q' \parallel (\boldsymbol{\nu} o)o \parallel (\boldsymbol{\nu} z)z.f \simeq Q' \;.$$

The equivalence $P' \simeq Q'$ follows because $\equiv \subseteq \simeq$ and $\simeq$ is transitive. $\square$

Exercise: explain the role of the $Flip$ process.

LTSs revisited

Reduction barbed congruence involves a universal quantification over all contexts. Weak bisimilarity does not, yet bisimilarity *is a sound proof technique* for reduction barbed congruence. How is this possible?

> An LTS captures all the interactions that a term can have with an arbitrary context. In particular, each label correspond to a minimal context.

For instance, in CCS, $P \xrightarrow{a} P'$ denotes the fact that P can interact with the context $C[-] = - \parallel \overline{a}$, yielding P'.

And τ transitions characterises all the interactions with an *empty context*.

Pi-calculus: labels

Given a process P, which are the contexts[1] that yield a reduction?

- if $P \equiv (\boldsymbol{\nu}\tilde{n})(\overline{x}\langle v\rangle.P_1 \mid\mid P_2)$ with $x, v \notin \tilde{n}$, then P interacts with the context

$$C[-] = - \;\Big|\Big|\; x(y).Q$$

  yielding:

$$C[P] \rightarrow \underbrace{(\boldsymbol{\nu}\tilde{n})(P_1 \;\Big|\Big|\; P_2)}_{P'} \;\Big|\Big|\; Q\{{}^{v}/_{y}\}$$

  We record this interaction with the label $\overline{x}\langle v\rangle$: $P \xrightarrow{\overline{x}\langle v\rangle} P'$.

[1] to simplify the notations, we will not write the most general contexts.

- if $P \equiv (\boldsymbol{\nu}\tilde{n})(x(y).P_1 \;||\; P_2)$ with $x \notin \tilde{n}$, then P interacts with the context

$$C[-] = - \;\Big|\Big|\; \overline{x}\langle v\rangle.Q \qquad \text{for } v \notin \tilde{n}\text{, yielding:}$$

$$C[P] \rightarrowtriangle \underbrace{(\boldsymbol{\nu}\tilde{n})(P_1\{^{v}/_{y}\} \;\Big|\Big|\; P_2)}_{P'} \;\Big|\Big|\; Q$$

We record this interaction with the label $x(v)$: $P \xrightarrow{x(v)} P'$

- If $P \rightarrowtriangle P'$, then P reduces without interacting with a context $C[-] = - \;||\; Q$:

$$C[P] \rightarrowtriangle P' \;\Big|\Big|\; Q$$

We record this interaction with the label τ: $P \xrightarrow{\tau} P'$.

Intermezzo

What if we define a labelled bisimilarity using the previous labels?

Consider the processes:

$$P = (\boldsymbol{\nu} v)\overline{x}\langle v\rangle \quad \text{and} \quad Q = \mathbf{0}$$

Obviously, $P \not\simeq Q$ because $P \downarrow x$ while $Q \not\Downarrow x$.

But both P and Q realise no labels: they are equated by the bisimilarity.

The bisimilarity is not *sound*!

Maybe we forgot a label...

The missing interaction

- if $P \equiv (\boldsymbol{\nu}\tilde{n})(\overline{x}\langle v\rangle.P_1 \parallel P_2)$ with $x \notin \tilde{n}$ and $v \in \tilde{n}$, then P interacts with the context

$$C[-] = - \,\big\|\, x(y).Q$$

yielding:

$$C[P] \rightarrow (\boldsymbol{\nu}v)(\underbrace{(\boldsymbol{\nu}\tilde{n} \setminus v)(P_1 \,\big\|\, P_2)}_{P'} \,\big\|\, Q\{^{v}/_{y}\})$$

We record this interaction with the label $(\boldsymbol{\nu}v)\overline{x}\langle v\rangle$: $P \xrightarrow{(\boldsymbol{\nu}v)\overline{x}\langle v\rangle} P'$.

Intuition: in P' the scope of v has been opened.

Summary of actions

ℓ	kind	$\mathrm{fn}(\ell)$	$\mathrm{bn}(\ell)$	$\mathsf{n}(\ell)$
$\overline{x}\langle y\rangle$	free output	$\{x, y\}$	$\emptyset$	$\{x, y\}$
$(\boldsymbol{\nu} y)\overline{x}\langle y\rangle$	bound output	$\{x\}$	$\{y\}$	$\{x, y\}$
$x(y)$	input	$\{x, y\}$	$\emptyset$	$\{x, y\}$
τ	internal	$\emptyset$	$\emptyset$	$\emptyset$

Pi-calculus: LTS

$$\overline{x}\langle v\rangle.P \xrightarrow{\overline{x}\langle v\rangle} P \qquad x(y).P \xrightarrow{x(v)} \{^{v}/_{y}\}P \qquad \frac{P \xrightarrow{\overline{x}\langle v\rangle} P' \qquad Q \xrightarrow{x(v)} Q'}{P \mid\mid Q \xrightarrow{\tau} P' \mid\mid Q'}$$

$$\frac{P \xrightarrow{\ell} P' \qquad \mathrm{bn}(\ell) \cap \mathrm{fn}(Q) = \emptyset}{P \mid\mid Q \xrightarrow{\ell} P \mid\mid Q} \qquad \frac{P \xrightarrow{\ell} P' \qquad v \notin \mathsf{n}(\ell)}{(\boldsymbol{\nu} v)P \xrightarrow{\ell} (\boldsymbol{\nu} v)P'} \qquad \frac{P \mid\mid !P \xrightarrow{\ell} P'}{!P \xrightarrow{\ell} P'}$$

$$\frac{P \xrightarrow{\overline{x}\langle v\rangle} P' \qquad x \neq v}{(\boldsymbol{\nu} v)P \xrightarrow{(\boldsymbol{\nu} v)\overline{x}\langle v\rangle} P'} \qquad \frac{P \xrightarrow{(\boldsymbol{\nu} v)\overline{x}\langle v\rangle} P' \qquad Q \xrightarrow{x(v)} Q' \qquad v \notin \mathrm{fn}(Q)}{P \mid\mid Q \xrightarrow{\tau} (\boldsymbol{\nu} v)(P' \mid\mid Q')}$$

Pi-calculus: bisimilarity

We can define bisimilarity for pi-calculus in the standard way.

Let $\stackrel{\hat{\ell}}{\Longrightarrow}$ be $\stackrel{\tau}{\longrightarrow}^* \stackrel{\ell}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^*$ if $\ell \neq \tau$, and $\stackrel{\tau}{\longrightarrow}^*$ otherwise.

Definition: Weak bisimilarity, denoted $\approx$, is the largest symmetric relation such that whenever $P \approx Q$ and $P \stackrel{\ell}{\longrightarrow} P'$ there exists Q' such that $Q \stackrel{\hat{\ell}}{\Longrightarrow} Q'$ and $P' \approx Q'$.

Back to the examples

1. $\overline{x}\langle y\rangle \not\approx \mathbf{0}$: trivial because $\overline{x}\langle y\rangle \xrightarrow{\overline{x}\langle y\rangle}$ and $\mathbf{0} \not\xrightarrow{\overline{x}\langle y\rangle}$.

2. $(\boldsymbol{\nu} x)\overline{x}\langle\rangle.R \approx \mathbf{0}$: the relation $\mathcal{R} = \{((\boldsymbol{\nu} x)\overline{x}\langle\rangle.R, \mathbf{0})\}^{=}$ is a bisimulation.

3. $(\boldsymbol{\nu} x)(\overline{x}\langle y\rangle.R_1 \parallel x(z).R_2) \approx (\boldsymbol{\nu} x)(R_1 \parallel R_2\{^{y}/_{z}\})$

   The relation

   $$\mathcal{R} = \{((\boldsymbol{\nu} x)(\overline{x}\langle y\rangle.R_1 \,\Big\|\, x(z).R_2), (\boldsymbol{\nu} x)(R_1 \,\Big\|\, R_2\{^{y}/_{z}\}))\}^{=} \cup \mathcal{I}$$

   is a bisimulation.

   $\mathcal{I}$ is the identity relation over processes, and $\mathcal{R}^{=}$ denotes the symmetric closure of $\mathcal{R}$.

Exercises

1. Compare the transitions of $F\lfloor u, v\rfloor$, where $F = (x, y).x(y).F\lfloor y, x\rfloor$ to those of its encoding in the recursion free calculus (use replication).

2. Consider the pair of mutually recursive definitions

$$\begin{array}{rcl} G & = & (u, v).(u().H\lfloor u, v\rfloor \parallel k().H\lfloor u, v\rfloor) \\ H & = & (u, v).v().G\lfloor u, v\rfloor \end{array}$$

   Write the process $G\lfloor x, y\rfloor$ in terms of replication (you have to invent the tecnique to translate mutually recursive definitions yourself).

3. Implement a process that negates at location a the truth-value found at location b. Implement a process that sums of two integers (using both the representations we have seen).

4. Design a representation for lists using π-calculus processes. Implement list append.

References

Books

- Robin Milner, Communicating and mobile systems: the π-calculus. (CUP,1999).
- Davide Sangiorgi, David Walker, The π-calculus: a theory of mobile processes. (CUP, 2001).

Tutorials available online:

- Robin Milner, The polyadic pi-calculus: a tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh.
- Joachim Parrow, An introduction to the pi-calculus. `http://user.it.uu.se/~joachim/intro.ps`
- Peter Sewell. Applied pi — a brief tutorial. Technical Report 498, University of Cambridge. `http://www.cl.cam.ac.uk/users/pes20/apppi.ps`