Concurrency 4 = CCS (2/4)

Scoping, weak and strong bisimulation

Pierre-Louis Curien (CNRS – Université Paris 7)

MPRI concurrency course 2004/2005 with :

Jean-Jacques Lévy (INRIA-Rocquencourt)
Eric Goubault (CEA)
James Leifer (INRIA - Rocq)
Catuscia Palamidessi (INRIA - Futurs)

(http://pauillac.inria.fr/~leifer/teaching/mpri-concurrency-2004)

Scope and recursion (1/4)

Consider (example of Frank Valencia) (we write μ for $\mu \cdot 0$) :

$$P_1 = (let\ K = \overline{a}|(\nu a)((a \cdot test)|K)\ in\ K)$$

Applying the rules, we have (two unfoldings) :

$$\frac{\dfrac{(\overline{a}|(\nu a)((a \cdot test)|\overline{a}|(\nu a)((a \cdot test)|K)) \xrightarrow{\tau} (\overline{a}|(\nu a)(test)0|(\nu a)((a \cdot test)|K))}{(\overline{a}|(\nu a)((a \cdot test)|K)) \xrightarrow{\tau} (\nu a)(test|0|(\nu a)((a \cdot test)|K))}}{K \xrightarrow{\tau} (\nu a)(test)0|(\nu a)((a \cdot test)|K))}$$

What about $P_2 = (let\ K = \overline{a}|(\nu b)((b \cdot test)|K)\ in\ K)$: the double enfolding yields $\overline{a}|(\nu b)((b \cdot test)|\overline{a}|(\nu b)((b \cdot test)|K)$, which is deadlocked, while the first definition of K allows to perform $test$ (notice the capture of $\overline{a}$).

Scope and recursion (2/4)

$$P_1 = (let\ K = \overline{a}|(\nu a)((a \cdot test)|K)\ in\ K)$$
$$P_2 = (let\ K = \overline{a}|(\nu b)((b \cdot test)|K)\ in\ K)$$

There is a tension :

- These two definitions have a different behaviour.

- The identity of bounded names should be irrelevant (α-conversion).
So let us rename a in the first definition :

$$P_3 = (let\ K = \overline{a}|(\nu b)((b \cdot test)|K[a \leftarrow b])\ in\ K)$$

But what is $K[a \leftarrow b]$? Well, we argue that it is not K, it is a substitution or (explicit) relabelling which is delayed until K is replaced by its actual definition (cf. e.g. λ-calculus with term metavariables and explicit substitutions)

So, all is well, we maintain both α-conversion ($P_1 = P_3$) and the difference of behaviour ($P_1 \neq P_2$), and the tension is resolved . . .

Scope and recursion (3/4)

In an α-conversion $(\nu x)P = (\nu y)P[x \leftarrow y]$, y should be chosen free in P. BUT when substitution arrives on K, how do I know whether y is free in K ? For example, in

$$P_4 = (let\ K = \overline{b}|(\nu a)((a \cdot test)|K)\ in\ K)$$

b is free in K, but I cannot know it from just looking at the subterm $(\nu a)((a \cdot test)|K)$.

Clean solution (definitions with parameters) : maintain the list of free variables of a constant K, and hence write constants always in the form $K(\vec{x})$ and make sure that in a definition $let\ K(\vec{a} = P\ in\ Q$ we have $FV(P) \subseteq \vec{a}$. (cf. syntax adopted in Milner's π-calculus book).

And now, relabelling can be omitted from syntax, i.e. left implicit, since, e.g. $K(a,b)[a \leftarrow c] = K(c,b)$.

Scope and recursion (4/4)

A "real" example : Consider the following linking operation :

$$P \frown Q = (\nu i', z', d')(P[i, z, d \leftarrow i', z', d'] | Q[\mathsf{inc}, \mathsf{zero}, \mathsf{dec} \leftarrow i', z', d'])$$

In particular

$$\begin{aligned} & C(\mathsf{inc}, \mathsf{zero}, \mathsf{dec}, z, d) \frown C(\mathsf{inc}, \mathsf{zero}, \mathsf{dec}, z, d) \\ & \quad = (\nu i', z', d')(C(\mathsf{inc}, \mathsf{zero}, \mathsf{dec}, z', d') | C(i', z', d', z, d)) \end{aligned}$$

A (unbounded) counter :

$$C = \mathsf{inc} \cdot (C \frown C) + \mathsf{dec} \cdot D \quad D = \overline{d} \cdot C + \overline{z} \cdot B \quad B = \mathsf{inc} \cdot (C \frown B) + \mathsf{zero} \cdot B$$

An example of execution :

$$\begin{aligned} & B \stackrel{\mathsf{zero}}{\rightarrow} B \stackrel{\mathsf{inc}}{\rightarrow} (C \frown B) \stackrel{\mathsf{inc}}{\rightarrow} ((C \frown C) \frown B) \stackrel{\mathsf{dec}}{\rightarrow} ((D \frown C) \frown B) \\ & \quad \stackrel{\tau}{\rightarrow} ((C \frown D) \frown B) \stackrel{\mathsf{dec}}{\rightarrow} ((D \frown D) \frown B) \stackrel{\tau}{\rightarrow} ((D \frown B) \frown B) \\ & \quad \stackrel{\tau}{\rightarrow} ((B \frown B) \frown B) \stackrel{\mathsf{inc}}{\rightarrow} ((C \frown B) \frown B \cdots \end{aligned}$$

Exercice 1 Show that there is no derivation $B \stackrel{\tau}{\rightarrow}^{\star} \stackrel{\mathsf{inc}}{\rightarrow} \stackrel{\tau}{\rightarrow}^{\star} \stackrel{\mathsf{dec}}{\rightarrow} \stackrel{\tau}{\rightarrow}^{\star} \stackrel{\mathsf{dec}}{\rightarrow}$.

.

Bisimilarity is not trace equivalence

As automata $P = a \cdot (b + c)$ and $Q = a \cdot b + a \cdot c$ recognize the same language $\{ab, ac\}$ of traces.

As processes, they are not bisimilar (Q does not even simulate P). P keeps the choice after performing a, Q not.

Think of a as inserting 40 cents, b as getting tea and c as getting coffee. Imagine a vending machine with a slot for a and two buttons for b and c. The machine allows you to press b (resp. c) only if action b (resp. c) can be performed. As a customer you will prefer P.

Strucural equivalence

Exercice 2 Show that structural equivalence $\equiv$ is included in (strong) bisimulation $\sim$.

Variations on bisimilarity (1/3)

A bisimulation up to $\sim$ is a relation $\mathcal{R}$ such that for all P, Q :

$P \mathcal{R}\, Q \Rightarrow \forall\, \mu, P' \; (P \xrightarrow{\mu} P' \Rightarrow \exists\, Q' \; Q \xrightarrow{\mu} Q'$ and $P' \sim \mathcal{R} \sim Q')$ and conversely

If $\mathcal{R}$ is strong bisimulation up to $\sim$, then $\mathcal{R} \subseteq \sim$.

Exercice 3 Prove it.

Hence, to show $P \sim Q$, it is enough to find a bisimulation up to $\sim$ such that $P \,\mathcal{R}\, Q$.

Variations on bisimilarity (2/3)

As an example, take

$$\begin{aligned} \mathit{Sem} &= \mathsf{P} \cdot \mathit{Sem}' \\ \mathit{Sem}' &= \mathsf{V} \cdot \mathit{Sem} \end{aligned} \qquad \begin{aligned} \mathit{Sem}^0 &= \mathsf{P} \cdot \mathit{Sem}^1 \\ \mathit{Sem}^1 &= \mathsf{P} \cdot \mathit{Sem}^2 + \mathsf{V} \cdot \mathit{Sem}^0 \\ \mathit{Sem}^2 &= \mathsf{P} \cdot \mathit{Sem}^3 + \mathsf{V} \cdot \mathit{Sem}^1 \\ \mathit{Sem}^3 &= \mathsf{V} \cdot \mathit{Sem}^2 \end{aligned}$$

Then a (strong) bisimulation up-to witnessing that $(\mathit{Sem}|\mathit{Sem}|\mathit{Sem}) \sim \mathit{Sem}^0$ is, say :

$$\begin{aligned} \{ \, & ((\mathit{Sem}|\mathit{Sem}|\mathit{Sem}) \, , \, \mathit{Sem}^0) \\ & ((\mathit{Sem}'|\mathit{Sem}|\mathit{Sem}) \, , \, \mathit{Sem}^1) \\ & ((\mathit{Sem}'|\mathit{Sem}|\mathit{Sem}') \, , \, \mathit{Sem}^2) \\ & ((\mathit{Sem}'|\mathit{Sem}'|\mathit{Sem}') \, , \, \mathit{Sem}^3) \, \} \end{aligned}$$

Variations on bisimilarity (3/3)

For any LTS, one can change Act to $Act^\star$ (words of actions), setting

$$P \stackrel{s}{\rightarrow} Q \text{ if } \begin{cases} s = \mu_1 \ldots \mu_n \text{ and} \\ (\exists\, P_1, \ldots, P_n \ (P_n = Q \text{ and } P \stackrel{\mu_1}{\rightarrow} P_1 \ldots \stackrel{\mu_n}{\rightarrow} P_n)) \end{cases}$$

This yields a new LTS, call it LTS* (the path LTS) . Then the notions of LTS and of LTS* bisimulation coincide.

From strong to weak bisimulation (1/2)

Take the LTS of CCS, with $Act = L \cup \overline{L} \cup \{tau\}$, call it Strong. The bisimulation for this system is called strong bisimulation.

Take Strong* (its path LTS).

Consider the following LTS, call it Weak†, with the same set of actions as Strong* :

$$P \overset{s}{\Rightarrow} Q \text{ if and only if } (\exists t \;\; P \overset{t}{\rightarrow} Q \text{ and } \hat{s} = \hat{t})$$

where the function $s \mapsto \hat{s}$ is defined as follows :

$$\hat{\epsilon} = \epsilon \quad \hat{\tau} = \epsilon \quad \hat{\alpha} = \alpha \quad \hat{s\mu} = \hat{s}\hat{\mu}$$

The idea is that weak bisimulation is bisimulation with possibly τ actions interspersed.

Let Weak be the LTS on *Act* whose transitions are $P \overset{\mu}{\Rightarrow} Q$, that is :

$$P \overset{\tau}{\Rightarrow} Q \text{ if and only if } P \overset{\tau}{\rightarrow}^\star Q \quad P \overset{\alpha}{\Rightarrow} Q \text{ if and only if } P \overset{\tau}{\rightarrow}^\star \overset{\alpha}{\rightarrow} \overset{\tau}{\rightarrow}^\star Q$$

Then one has Weak† = Weak*.

From strong to weak bisimulation (2/2)

None of the three equivalent definition of weak bisimulation (Weak, Weak†, Weak*) is practical. The following is a fourth, equivalent, and more tractable version :

A weak bisimulation is a relation $\mathcal{R}$ such that

$$P \,\mathcal{R}\, Q \Rightarrow \forall\, \mu, P' \;\; (P \xrightarrow{\mu} P' \Rightarrow \exists\, Q' \;\; Q \stackrel{\mu}{\Rightarrow} Q' \text{ and } P' \,\mathcal{R}\, Q') \text{ and conversely}$$

Two processes are weakly bisimilar if (notation $P \approx Q$) if there exists a weak bisimulation $\mathcal{R}$ such that $P \,\mathcal{R}\, Q$.

Bisimulation is a congruence (1/6)

We define $\sim^*$ inductively by the following rules :

$$\frac{P \sim Q}{P \sim^* Q} \qquad \frac{P \sim^* Q}{Q \sim^* P} \qquad \frac{P \sim^* Q \quad Q \sim^* R}{P \sim^* R}$$

$$\frac{\forall\, i \in I \;\; P_i \sim^* Q_i}{\Sigma_{i \in I} \mu_i \cdot P_i \sim^* \Sigma_{i \in I} \mu_i \cdot Q_i} \qquad \frac{P_1 \sim^* Q_1 \;\; P_2 \sim^* Q_2}{P_1 \mid P_2 \sim^* Q_1 \mid Q_2} \qquad \frac{P \sim^* Q}{(\nu a)P \sim^* (\nu a)Q}$$

Clearly $\sim \subseteq \sim^*$ and $\sim^*$ is a congruence, by construction. It is enough to show that $\sim^*$ is a bisimulation (since then $\sim\, = \sim^*$ is a congruence).

Bisimulation is a congruence (2/6)

Proof by rule induction. We look at case $P_1 \mid P_2 \sim^* Q_1 \mid Q_2$:

1. (backward) decomposition phase : if $P_1|P_2 \xrightarrow{\mu} P'$, then $P' = P'_1|P'_2$ and three cases may occur, corresponding to the three rules for parallel composition in the labelled operational semantics. We only consider the synchronisation case. If $P_1 \xrightarrow{a} P'_1$ and $P_2 \xrightarrow{\overline{a}} P'_2$, then

2. by induction there exists Q'_1 such that $Q_1 \xrightarrow{a} Q'_1$ and $P'_1 \sim^* Q'_1$, and there exists Q'_2 such that $Q_2 \xrightarrow{\overline{a}} Q'_2$ and $P'_2 \sim^* Q'_2$.

3. Hence (forward phase) we have $Q_1 \mid Q_2 \xrightarrow{\tau} Q'_1 \mid Q'_2$ and $P'_1 \mid P'_2 \sim^* Q'_1 \mid Q'_2$.

Bisimulation is a congruence (3/6)

$\approx$ is also a congruence (for our choice of language with guarded sums).

Same proof technique : define $\approx^*$. For the forward phase, we use the following properties, which are true :

$$(P \stackrel{\mu}{\Rightarrow} P') \quad \Rightarrow \quad ((\nu a)P \stackrel{\mu}{\Rightarrow} (\nu a)Q')$$
$$(Q_1 \stackrel{\mu}{\Rightarrow} Q_1') \quad \Rightarrow \quad (Q_1 \mid Q_2 \stackrel{\mu}{\Rightarrow} Q_1' \mid Q_2)$$
$$(Q_1 \stackrel{a}{\Rightarrow} Q_1' \text{ and } Q_2 \stackrel{\overline{a}}{\Rightarrow} Q_2') \quad \Rightarrow \quad (Q_1 \mid Q_2 \stackrel{\tau}{\Rightarrow} Q_1' \mid Q_2')$$

Bisimulation is a congruence (4/6)

Consider CCS with prefix and sums instead of guarded sums, i.e., replace $\Sigma_{i\in I}\mu_i \cdot P_i$ by two constructs $\Sigma_{i\in I}P_i$ and $a \cdot P$, with rules

$$\frac{P_i \xrightarrow{\mu} P_i'}{\Sigma_{i\in I}P_i \xrightarrow{\mu} P_i'} \qquad \frac{}{\mu \cdot P \xrightarrow{\mu} P}$$

Then strong bisimulation is a congruence, and weak bisimulation is not a congruence.

The problem does not arise because more processes (like $P + (Q|R)$) are allowed.

Bisimulation is a congruence (5/6)

What goes wrong is the sum rule ? For the forward phase, we would need the property :

$$(Q_1 \stackrel{\mu}{\Rightarrow} Q_1') \quad \Rightarrow \quad (Q_1 + Q_2 \stackrel{\mu}{\Rightarrow} Q_1')$$

which does not hold (take $\mu = \tau$ and $Q_1' = Q_1$).

Counter-example : $\tau \cdot a \cdot 0 + b \cdot 0 \not\approx a \cdot 0 + b \cdot 0$

Bisimulation is a congruence (6/6)

We have left out recursion, but even so we have :

Proposition : For any process S (possibly with recursive definitions) with free variables in $\vec{K}$:

$$\forall \vec{Q}, \vec{Q}' \;\; (\vec{Q} \approx \vec{Q}' \Rightarrow S[\vec{K} \leftarrow \vec{Q}] \approx S[\vec{K} \leftarrow \vec{Q}'])$$

The proof is by induction on the size of S. The non-recursion cases follow by congruence. For the recursive definition case $S = let\ \vec{L} = \vec{P}\ in\ L_j$, the trick is to unfold :

$$\begin{array}{rcl}
S[\vec{K} \leftarrow \vec{Q}] & =_{\mathsf{def}} & let\ \vec{L} = \vec{P}[\vec{K} \leftarrow \vec{Q}]\ in\ L_j \\
 & \approx & P_j[\vec{K} \leftarrow \vec{Q}][\vec{L} \leftarrow (let\ \vec{L} = \vec{P}\ in\ \vec{L})] \\
 & \approx_{\mathsf{ind}} & P_j[\vec{K} \leftarrow \vec{Q}'][\vec{L} \leftarrow (let\ \vec{L} = \vec{P}\ in\ \vec{L})] \\
 & \approx & S[\vec{K} \leftarrow \vec{Q}']
\end{array}$$