Concurrency 2

CCS : Static scoping, bisimulation, coinduction

Pierre-Louis Curien (CNRS – Université Paris 7)

MPRI concurrency course 2006/2007 with :

Francesco Zappa Nardelli (INRIA Rocquencourt)
Catuscia Palamidessi (INRIA - Futurs)
Roberto Amadio (Paris 7)

(`http://mpri.master.univ-paris7.fr/C-2-3.html`)

.

Recommended readings

Courses 2, 3, and 4 build mainly on Milner's "red book" **Communication and Concurrency** (see course's web page).

For a few complements and more examples, we refer to last year's courses 4 and 5, by Catuscia Palamidessi (`http://pauillac.inria.fr/~leifer/teaching/mpri-concurrency-2005`) (course 3 of that series is course 1 of this year).

From automata to CCS (1/6)

Remove final (we are not primarily interested in termination), and initial states (assimilate processes with states, hence any state is "initial" relative to the process it is identified with).

Such an automaton deprived from initial and final states is called a labelled transition system, or **LTS** for short.

From automata to CCS (2/6)

An LTS is given by

- a finite set of states, or $P, Q, \ldots$,
- a finite alphabet *Act* whose members are called actions, and
- transitions between them, written $P \xrightarrow{\mu} Q$.

From automata to CCS (3/6)

A LTS together with one of its states, that is, a process, can be described by the following syntax :

$$P ::= \Sigma_{i \in I} \mu_i \cdot P_i \mid let\ \vec{K} = \vec{P}\ in\ K_j \mid K$$

(empty sum denoted by 0)

From automata to CCS (4/6)

CCS $\quad P ::= \Sigma_{i \in I} \mu_i \cdot P_i \mid let\ \vec{K} = \vec{P}\ in\ K_j \mid K \mid (P \mid Q) \mid (\nu a)P$

Synchronization Trees $\quad P ::= \Sigma_{i \in I} \mu_i \cdot P_i$

Finitary CCS $\quad P ::= \Sigma_{i \in I} \mu_i \cdot P_i \mid (P \mid Q) \mid (\nu a)P \quad (I$ finite)

w.r.t. Catuscia's course : we use guarded sums only (useful to prove that weak bisimulation is a congruence), and mutually recursive definitions ($\mathtt{rec}_K P = (let\ K = P\ in\ K)$).

In practice, one writes a context of (sets of mutually recursive) definitions

$K_1 = P_1\ \ldots\ K_n = P_n$ instead of $(let\ \vec{K} = \vec{P}\ in\ K_1)\ \ldots\ (let\ \vec{K} = \vec{P}\ in\ K_n)$

Not the ultimate syntax yet (see scoping below) !

From automata to CCS (5/6)

in CCS

$$Act = L \cup \overline{L} \cup \{\tau\}$$

(disjoint union), where L is the set of labels, also called names, or channels, and τ is a silent action that records a synchronisation. $\mu \in Act$, $\alpha \in L \cup \overline{L}$, $\overline{\overline{\alpha}} = \alpha$

From automata to CCS (6/6)

We write

$$\Sigma_{i\in I} a_i \cdot P_i = (\Sigma_{i\in I\setminus i_0} a_i \cdot P_i) + a_{i_0} \cdot P_{i_0}$$

(note that the notation implicitly views sums as associative and commutative – this will be made explicit later)

Labelled operational semantics (1/4)

$$\frac{}{\Sigma_{i \in I} \mu_i \cdot P_i \xrightarrow{\mu_i} P_i} \qquad \frac{P \xrightarrow{\mu} P' \quad (\mu \neq a, \overline{a})}{(\nu a)P \xrightarrow{\mu} (\nu a)P'}$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \frac{Q \xrightarrow{\mu} Q'}{P \mid Q \xrightarrow{\mu} P \mid Q'} \qquad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\frac{P_j[\vec{K} \leftarrow (let\ \vec{K} = \vec{P}\ in\ \vec{K})] \xrightarrow{\mu} P'}{let\ \vec{K} = \vec{P}\ in\ K_j \xrightarrow{\mu} P'}$$

Labelled operational semantics (2/4)

τ-transitions (resp. α-transitions) correspond to internal evolutions (resp. interactions with the environment).
Rule COMM involves **both**.

In λ-calculus, one considers only one (internal) reduction : β.

Labelled operational semantics (3/4)

Example :

$$P = (\nu c)(K_1 \mid K_2) \text{ where } \begin{cases} K_1 = a \cdot \overline{c} \cdot K_1 \\ K_2 = b \cdot c \cdot K_2 \end{cases}$$

Behaviour : do a and b independently, then τ, then loop.

Labelled operational semantics (4/4)

It is possible to formulate internal reduction in CCS without reference to the environment.

Price to pay : work modulo structural equivalence.

.

Structural equivalence

$\Sigma_{i\in I}\mu_i \cdot P_i \equiv \Sigma_{i\in I}\mu_{f(i)} \cdot P_{f(i)}$ (f permutation)

$P \mid Q \equiv Q \mid P$

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

$((\nu a)P) \mid Q \equiv (\nu a)\ (P \mid Q)$ (a not free in Q)

$let\ \vec{K} = \vec{P}\ in\ K_j \equiv P_j[\vec{K} \leftarrow (let\ \vec{K} = \vec{P}\ in\ \vec{K})]$

Reduction operational semantics (1/2)

$$\frac{}{P_1 + a \cdot P \mid \overline{a} \cdot Q + Q_1 \rightarrow P \mid Q} \qquad \frac{}{P_1 + \tau \cdot P \rightarrow P}$$

$$\frac{P_1 \rightarrow P_1'}{P_1 \mid P_2 \rightarrow P_1' \mid P_2} \qquad \frac{P \rightarrow P'}{(\nu a)P \rightarrow (\nu a)P'}$$

$$\frac{P_1 \;\equiv\; P_2 \;\rightarrow\; P_2' \;\equiv\; P_1'}{P_1 \rightarrow P_1'}$$

Reduction operational semantics (2/2)

The relations $\rightarrow$ and $\xrightarrow{\tau}\equiv$ coincide.

Exercice 1 Prove it, via the following claims :

- If $P \xrightarrow{\mu} P'$ and $P \equiv Q$, then there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \equiv Q'$.
- If $P \xrightarrow{\alpha} P'$, then $P \equiv (\nu\vec{a})\,(\alpha \cdot Q + P_1 \mid P_2)$ and $P' \equiv (\nu\vec{a})\,(P_1 \mid P_2)$, for some $\vec{a}, P_1, P_2, Q$.

.

Semaphore in CCS

$$\mathit{Sem} = \mathsf{P} \cdot \mathsf{V} \cdot \mathit{Sem}$$

$$\begin{aligned}
&\mathit{Sem} \mid (\overline{\mathsf{P}} \cdot C_0; \overline{\mathsf{V}}) \mid (\overline{\mathsf{P}} \cdot C_1; \overline{\mathsf{V}}) \\
&\quad \rightarrow (\mathsf{V} \cdot \mathit{Sem}) \mid (\overline{\mathsf{P}} \cdot C_0; \overline{\mathsf{V}}) \mid (C_1; \overline{\mathsf{V}}) \\
&\quad \rightarrow^{\star} (\mathsf{V} \cdot \mathit{Sem}) \mid (\overline{\mathsf{P}} \cdot C_0; \overline{\mathsf{V}}) \mid \overline{\mathsf{V}} \\
&\quad \rightarrow \mathit{Sem} \mid (\overline{\mathsf{P}} \cdot C_0; \overline{\mathsf{V}})
\end{aligned}$$

Exercice 2 Encode $P; Q$ in CCS.

Value passing

$$\frac{}{P_1 + a(x) \cdot P \mid \overline{a}\langle v \rangle \cdot Q + Q_1 \rightarrow P[x \leftarrow v] \mid Q}$$

A memory cell

Persistent : $\mathit{Reg}\langle x \rangle = \overline{\mathit{Get}}\langle x \rangle \cdot \mathit{Reg}\langle x \rangle + \mathit{Put}(y) \cdot \mathit{Reg}\langle y \rangle$

One-shot : $\begin{cases} \mathit{Sem}\langle x \rangle = (\overline{\mathit{Get}}\langle x \rangle \cdot K) + K \\ K = \mathit{Put}(y) \cdot \mathit{Sem}\langle y \rangle \end{cases}$

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) :

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

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 not free in P. BUT when substitution arrives on K, how do I know whether y is occurs (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. $K(a,b)[a \leftarrow c] = K(c,b)$.

Exercice 3 Express the LTS rule for constants in this new setting.

Scope and recursion (4/4)

A "real" example : Consider the following linking operation (with implicit substitution) :

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

In particular
$$\begin{cases} C(\mathsf{inc}, \mathsf{zero}, \mathsf{dec}, z, d) \frown C(\mathsf{inc}, \mathsf{zero}, \mathsf{dec}, z, d) \\ = (\nu i', z', d')(C(\mathsf{inc}, \mathsf{zero}, \mathsf{dec}, z', d')|C(i', z', d', z, d)) \end{cases}$$

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 :

$$B \xrightarrow{\mathsf{zero}} B \xrightarrow{\mathsf{inc}} (C \frown B) \xrightarrow{\mathsf{inc}} ((C \frown C) \frown B) \xrightarrow{\mathsf{dec}} ((D \frown C) \frown B)$$
$$\xrightarrow{\tau} ((C \frown D) \frown B) \xrightarrow{\mathsf{dec}} ((D \frown D) \frown B) \xrightarrow{\tau} ((D \frown B) \frown B)$$
$$\xrightarrow{\tau} ((B \frown B) \frown B) \xrightarrow{\mathsf{inc}} ((C \frown B) \frown B \cdots$$

Exercice 4 Make the parameters of C, D, B explicit in the above definition of counter.

Exercice 5 Show that there is no derivation $B \xrightarrow{\tau}{}^{\star} \xrightarrow{\mathsf{inc}} \xrightarrow{\tau}{}^{\star} \xrightarrow{\mathsf{dec}} \xrightarrow{\tau}{}^{\star} \xrightarrow{\mathsf{dec}}$.

CCS encodings (1/4)

(Thanks to Catuscia Palamidessi for these encodings)

Here is a specification P of (up to) n readers in parallel and (at most) one writer :

$$R = \overline{p_R} \cdot \mathsf{read} \cdot \overline{v_R} \qquad S_0 = p_R \cdot S_1 + p_W \cdot v_W \cdot S_0$$

$$W = \overline{p_W} \cdot \mathsf{write} \cdot \overline{v_W} \qquad S_k = p_R \cdot S_{k+1} + v_R \cdot S_{k-1} \quad (0 < k < n)$$

$$S_n = v_R \cdot S_{n-1}$$

in

$(\nu p_R, v_R, p_W, v_W)(S_0|R|\cdots|R|W|\cdots|W)$ (arbitrarily many readers and writers)

If $P \xrightarrow{s} (\nu p_R, v_R, p_W, v_W)P'$, then
$(\nu p_R, v_R, p_W, v_W)P' \xrightarrow{s'} (\nu p_R, v_R, p_W, v_W)P''$, where

- $P'' = S_i|Q$ (up to i threads of Q can perform read and no thread can perform write), or

- $P'' = (v_W \cdot S_0)|Q$ (no thread of Q can perform read and at most one thread can perform write)

CCS encodings (2/4)

The dining philosophers can be encoded by a closed linking (cf. above) of n copies of the following process $\mathsf{Phil}_{n,p,a}$ (each philosopher holds its left fork at the beginning)

$$\mathsf{Phil}_{n,p,a} = \tau \cdot \mathsf{Phil}_{h,p,a} + \tau \cdot \mathsf{Phil}_{n,p,a} + \overline{c_L} \cdot \mathsf{Phil}_{n,a,a}$$

$$\mathsf{Phil}_{n,a,p} = \text{ symmetric}$$

$$\mathsf{Phil}_{n,a,a} = \tau \cdot \mathsf{Phil}_{n,a,a} + \tau \cdot \mathsf{Phil}_{h,a,a}$$

$$\mathsf{Phil}_{h,a,a} = c_L \cdot \mathsf{Phil}_{h,p,a} + c_R \cdot \mathsf{Phil}_{h,a,p}$$

$$\mathsf{Phil}_{h,p,a} = \overline{c_L}\mathsf{Phil}_{h,a,a} + c_R \cdot \mathsf{Phil}_{h,p,p}$$

$$\mathsf{Phil}_{h,a,p} = \text{ symmetric}$$

$$\mathsf{Phil}_{h,p,p} = \mathsf{eat} \cdot \mathsf{Phil}_{n,p,p}$$

$$\mathsf{Phil}_{n,p,p} = \overline{c_L} \cdot \mathsf{Phil}_{n,a,p} + \overline{c_R} \cdot \mathsf{Phil}_{n,p,a}$$

- n/h stand for "not hungry" / "hungry", a/p for absent / present (second and third index for first and second fork, respectively)

- under the linking, c_R (resp. c_L) is (privately) identified with the c_L (resp. c_R) of the right (resp. left) neighbour

CCS encodings (3/4)

We show, at any stage : Fairness $\Rightarrow$ Progress

Fairness A hungry philosopher, or a philosopher who has just eaten, is not ignored forever.

Progress If at least one philosopher is hungry, then eventually one of the hungry philosophers will eat.

By contradiction : Suppose P is the state of the system in which one philosopher at least is hungry, and suppose that there is an infinite fair evolution $P \xrightarrow{\tau}^{\star} \cdots$ that makes no progress. Then :

Step 1 : Eventually, all philosophers hold at most one fork. No philosopher at any stage can be in state (h, p, p), since by fairness eventually this philosopher will eat. If at some stage a philosopher is in state (n, p, p), then by fairness this philosopher will eventually give one of his forks. No philosopher at any styage can be in state (n, p, p) unless it was already in this state in P, since the only way to enter this state is after eating. Hence all the (n, p, p) states will eventually disappear.

CCS encodings (4/4)

Step 2 : Eventually, all philosophers hold exactly one fork. This is because if one philosopher had no fork, then another one would hold two (n forks for $n-1$ philosophers).

Step 3 : When this happens, our philosopher is still hungry (he cannot revert to non-hungry unless he eats), say it is in state (h, p, a), and eventually by Fairness it is his turn. The transition (h, p, p) is forbidden. Hence he gives his fork to the left neighbour. Only a hungry philosopher receives forks, hence the neighbour is in state (h, p, a), but then makes the transition (h, p, p) which is also forbidden.

Exercice 6 Show that the system can never deadlock.

.

Bisimulation on a LTS (1/4)

A simulation is a binary relation $\mathcal{R}$ on the set of processes such that for all P, Q, if $P \,\mathcal{R}\, Q$ then

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

Bisimulation on a LTS (2/4)

A bisimulation is a binary relation $\mathcal{R}$ on the set of processes such that $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations.

$(\mathcal{R}^{-1} = \{(Q, P) \mid P \, \mathcal{R} \, Q\})$

P, Q are bisimilar (notation $P \sim Q$) if there exists a bisimulation $\mathcal{R}$ such that $P \, \mathcal{R} \, Q$.

Bisimulation on a LTS (3/4)

If $\mathcal{R}, \mathcal{S}$ are bisimulations, then so is their composition

$$RS = \{(P, R) \mid \exists Q \;\; P \,\mathcal{R}\, Q \text{ and } Q \,\mathcal{S} R\}$$

In particular, $\sim\sim \subseteq \sim$, i.e., bisimilarity is transitive.

.

Bisimulation on a LTS (4/4)

Two processes that simulate one another, yet are not bisimilar :

$$P_1 = a \cdot P_2 + a \cdot P_4 \qquad Q_1 = a \cdot Q_2$$
$$P_2 = b \cdot P_3 \qquad Q_2 = b \cdot Q_3$$

$$P_1 \,\mathcal{T}\, Q_1 \quad P_4 \,\mathcal{T}\, Q_2 \quad P_2 \,\mathcal{T}\, Q_2 \quad P_3 \,\mathcal{T}\, Q_3$$
$$Q_1 \,\mathcal{S}\, P_1 \quad Q_2 \,\mathcal{S}\, P_2 \quad Q_3 \,\mathcal{S}\, P_3 \;.$$

but for all simulation $\mathcal{R}$ containing (P_1, Q_1) we have :
$P_1 \,\mathcal{R}\, Q_1$ and $P_1 \xrightarrow{a} P_4 \Rightarrow P_4 \,\mathcal{R}\, Q_2$

Induction and coinduction (1/4)

A function $f : D \to E$, where D, E are partial orders, is monotonous if

$$\forall x, y \;\; x \leq y \Rightarrow f(x) \leq f(y)$$

Given (monotonous) $f : D \to D$, a prefixpoint (resp. a postfixpoint, a fixpoint) of f is a point x such that $f(x) \leq x$ (resp. $x \leq f(x)$, $x = f(x)$).

Induction and coinduction (2/4)

Any monotonous function $G : \mathcal{P}(X) \rightarrow \mathcal{P}(X)$ has a least prefixpoint, which is moreover a fixpoint, and a greatest postfixpoint, which is moreover a fixpoint. They are respectively :

$$\mathit{lfp}(G) = \bigcap\{X \mid G(X) \subseteq X\}$$
$$\mathit{gfp}(G) = \bigcup\{X \mid X \subseteq G(X)\}$$

Induction and coinduction (3/4)

Induction principle : To show *lfp*$(\mu) \subseteq X$ is is enough to show X is a prefixpoint, i.e., $\mu(X) \subseteq X$.

In practice, the induction principle is often used for a subset X of *lfp*(μ), and then serves to show that $X =$ *lfp*(μ).

Induction and coinduction (4/4)

Coinduction principle : To show $X \subseteq \mathit{gfp}(\mu)$ it is enough to show $X \subseteq \mu(X)$.

In practice, the principle of coinduction is used to show that some element x is in $\mathit{gfp}(\mu)$, and for this it is enough to find a postfixpoint X such that $x \in X$.

Continuity

$G : \mathcal{P}(X) \to \mathcal{P}(X)$ is continuous if it preserves $\bigcup$ of increasing chains, i.e. $G(\bigcup_{n\in\omega} X_n) = \bigcup_{n\in\omega} G(X_n)$. G is called anti-continuous if it preserves $\bigcap$ of decreasing chains.

$$G \text{ continuous} \quad \Rightarrow \quad \mathit{lfp}(G) = \bigcup_{n\in\omega} G^n(\emptyset)$$
$$G \text{ anti-continuous} \quad \Rightarrow \quad \mathit{gfp}(G) = \bigcap_{n\in\omega} G^n(X)$$

For monotonous (non necessarily continuous) operators, similar formulas hold, using transfinite induction.

.

Operators defined by rules (1/5)

Monotonous operators G_K on $\mathcal{P}(X)$ defined via a set K of rules, each of the form (Y, x), with $Y \subseteq X$ and $x \in X$, or, graphically (for $Y = \{x_1, \ldots, x_n\}$ finite) :

$$\frac{\{x_1, \ldots, x_n\}}{x}$$

Set $G_K(R) = \{x \in X \mid \exists\,(Y, x) \in K \;\; Y \subseteq R\}$.

Operators defined by rules (2/5)

Prefixpoints of G_K =
subsets R closed forwards by the rules :

$$\forall\,(Y,x) \in K \;\; (Y \subseteq R \Rightarrow x \in R)$$

Postfixpoints of G_K =
subsets R closed backwards by the rules :

$$\forall\, x \in R \;\exists\,(Y,x) \in K \quad Y \subseteq R$$

Operators defined by rules (3/5)

Bisimulation is defined by a set of rules : take K to be the set of all

$$\frac{\{(P', f(\mu, P')) \mid P \xrightarrow{\mu} P'\} \cup \{(g(\mu, Q'), Q') \mid Q \xrightarrow{\mu} Q'\}}{(P, Q)}$$

where f is any function mapping each pair μ, P' such that $P \xrightarrow{\mu} P'$ to a process $f(\mu, P')$ such that $Q \xrightarrow{\mu} f(\mu, P')$ (resp. g ...).

Operators defined by rules (4/5)

If all the Y's in the rules of K are finite, then G_K is continuous.
If, for all x, $\{(Y \mid (Y, x) \in K\}$ is finite, then G_K is anti-continuous.

In finitary CCS the bisimulation operator G_K is both continuous and anti-continuous.

NB : finite sum assumption is not enough : take $let\ K = (a \cdot 0 \mid K)\ in\ K$.

Operators defined by rules (5/5)

Consider the following K :

$$\frac{}{\textit{nil}} \qquad \frac{l}{\textit{cons}(a,l)}$$

The *lfp* of G_K is the set of lists. The *gfp* of G_K is the set of finite and infinite lists.

N.B. The right setting is categorical : initial and final algebras for the functor $F(X) = \{*\} \cup A \times X$.

Exercice 7 How to obtain infinite lists (only) ?