MPRI - Cours de Concurrence - 2006

Lectures 9-12

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

Roberto Amadio

Université de Paris 7

Laboratoire Preuves, Programmes et Systèmes

Programme of these lectures

We will cover the notions of:

- Determinacy and Confluence.
- Synchrony.
- Termination and Reactivity.

in the framework of *process calculi* (specifically, CCS, π-calculus, and variations thereof).

NB These lectures aim both at presenting some *basic results* and at introducing to some *areas of ignorance* (a.k.a research).

Advertising

- On January 8th, 15th, 22nd, 29th (Monday, last slot) there will be 4 lectures by Robin Milner. Attendance is recommended. You should be able to get 2 credits for this (to be confirmed).

- Those who want to do research on the topics of this course might be interested in the *Groupe de travail Concurrence Chevaleret, Thursday 2 pm.*

$$http://www.pps.jussieu.fr/\sim amadio/cc/$$

Determinacy

What is a deterministic system?

In automata theory, one can consider various definitions. For instance, look at *finite automata*:

Def 1 There is no word w that admits two computation paths in the graph such that one leads to an accepting state and the other to a non-accepting state.

Def 2 Each reachable configuration admits at most one successor.

Def 3 For each state:

- either there is exactly one outgoing transition labelled with ϵ,
- or all outgoing transitions are labelled with distinct symbols of the alphabet Σ.

Thus one can go from *'extensional' conditions* (intuitive but hard to verify) to *'syntactic' conditions* (verifiable but not as general).

Why did we allow non-determinism?

Race conditions Two clients request the same service.

$$\nu a\ (\overline{a}.P_1 \mid \overline{a}.P_2 \mid a)$$

General specification and portability We do not want to commit on a particular behaviour. For instance, consider

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

to minimize context switches, in a mono-processor implementation we might always run $\overline{d}$ after c. However, in a multi-processor, we might run $\overline{e}$ at the place of $\overline{d}$.

Why is determinism desirable?

- Easier to *test and debug*.
- Easier to *prove correct*.

NB Often the implementation seems ‘deterministic’ because:

- either the program is inherently deterministic,
- or the scheduler determinizes the program’s behaviour.

Towards a definition of determinacy

- If we run an *'experiment'* twice we always get the same 'result'.
- If P and P' are *'equivalent'* then one is determinate if and only if the other is.
- If P is determinate and we run an experiment then *the residual of* P after the experiment should still be determinate.

- Most of the time, we will place ourselves in the context of a simple model such as *CCS*.
- We take *equivalent* to mean *weak bisimilar*.
- We take *experiment* to be a finite sequence of interactions.

A formal definition of determinacy

- Denote with $\mathcal{L}$ the set of *visible actions and co-actions* with generic elements $\ell, \ell', \ldots$
- Denote with $Act = \mathcal{L} \cup \{\tau\}$ the set of *actions*, with generic elements $\alpha, \beta, \ldots$
- Let $s \in \mathcal{L}^*$ denote a finite word over $\mathcal{L}$. Then:

$$\begin{array}{ll} P \stackrel{\epsilon}{\Rightarrow} P' & \text{if } P \stackrel{\tau}{\Rightarrow} P' \\ P \stackrel{\ell_1 \ldots \ell_n}{\Rightarrow} P',\ n \geq 1 & \text{if } P \stackrel{\ell_1}{\Rightarrow} \cdots \stackrel{\ell_n}{\Rightarrow} P' \end{array}$$

Definition A process P is *determinate* if for any $s \in \mathcal{L}^*$,

$$\frac{P \stackrel{s}{\Rightarrow} P' \quad P \stackrel{s}{\Rightarrow} P''}{P' \approx P''}$$

Exercise

Are the following CCS processes determinate?

1. $a.(b + c)$.
2. $a.b + ac$.
3. $a + a.\tau$.
4. $a + \tau.a$.
5. $a + \tau$.

Proposition

1. If P is determinate and $P \xrightarrow{\alpha} P'$ then P' is determinate.
2. If P is determinate and $P \approx P'$ then P' is determinate.

Proof idea

1. Suppose $P \xrightarrow{\alpha} P'$ and $P' \stackrel{s}{\Rightarrow} P_i$ for $i = 1, 2$.
   - If $\alpha = \tau$ then $P \stackrel{s}{\Rightarrow} P_i$ for $i = 1, 2$. Hence $P_1 \approx P_2$.
   - If $\alpha = \ell$ then $P \stackrel{\ell \cdot s}{\Rightarrow} P_i$ for $i = 1, 2$. Hence $P_1 \approx P_2$.

2. Suppose $P \approx P'$ and $P' \stackrel{s}{\Rightarrow} P'_i$ for $i = 1, 2$.

   - By definition of *weak bisimulation*:

   $$P \stackrel{s}{\Rightarrow} P_i \text{ and } P_i \approx P'_i$$

   for $i = 1, 2$.

   - Since P is *determinate*, we have $P_1 \approx P_2$.
   - Therefore, we conclude by *transitivity* of $\approx$:

   $$P'_1 \approx P_1 \approx P_2 \approx P'_2$$

   NB Most proofs in this lecture will be by *diagram chasing*.

τ-inertness and determinacy

Definition We say that a process P is *τ-inert* if for all its derivatives Q, if $Q \stackrel{\tau}{\Rightarrow} Q'$ then $Q \approx Q'$.

Proposition If P is determinate then it is τ-inert.

Proof idea

- Suppose $P \overset{s}{\Rightarrow} Q$ and $Q \overset{\tau}{\Rightarrow} Q'$.
- Then $P \overset{s}{\Rightarrow} Q$ and $P \overset{s}{\Rightarrow} Q'$.
- Thus by determinacy, $Q \approx Q'$.

Trace equivalence

We define the *traces* of a process P as

$$tr(P) = \{s \in \mathcal{L}^* \mid P \stackrel{s}{\Rightarrow} \cdot\}$$

and say that two processes P, Q are *trace equivalent* if $tr(P) = tr(Q)$.

NB The traces of a process form a *non-empty*, *prefix-closed* set of *finite words* over $\mathcal{L}$.

Exercise

Are the following equations valid for *trace equivalence* and/or *weak bisimulation*?

1. $a + \tau = a$.
2. $\alpha.(P + Q) = \alpha.P + \alpha.Q$.
3. $(P + Q) \mid R = P \mid R + Q \mid R$.
4. $P = \tau.P$.

Proposition

1. If $P \approx Q$ then $tr(P) = tr(Q)$.
2. Moreover, if P, Q are *determinate* then $tr(P) = tr(Q)$ implies $P \approx Q$.

Proof idea

1. Suppose $P \approx Q$ and $P \stackrel{s}{\Rightarrow} \cdot$. Then $Q \stackrel{s}{\Rightarrow} \cdot$ by induction on $|s|$ using the properties of weak bisimulation.

2. Suppose P, Q determinate and $tr(P) = tr(Q)$.

   - We show that

   $$\{(P, Q) \mid tr(P) = tr(Q)\}$$

   is a bisimulation.

- If $P \xrightarrow{\tau} P'$ then $P \approx P'$ by *determinacy*.
- Thus taking $Q \stackrel{\tau}{\Rightarrow} Q$ we have:

$$P' \approx P \qquad tr(P) = tr(Q) \ .$$

- By (1), we conclude:

$$tr(P') = tr(P) = tr(Q) \ .$$

- If $P \xrightarrow{\ell} P'$ then we note that:

$$tr(P) = \{\epsilon\} \cup \{\ell\} \cdot tr(P') \cup \bigcup_{\ell \neq \ell', P \stackrel{\ell'}{\Rightarrow} P''} \{\ell'\} \cdot tr(P'')$$

- This is because all the processes P' such that $P \stackrel{\ell}{\Rightarrow} P'$ are bisimilar, hence trace equivalent.
- A similar reasoning applies to $tr(Q)$.
- Thus there must be a Q' such that $Q \stackrel{\ell}{\Rightarrow} Q'$ and $tr(P') = tr(Q')$.

How do we build deterministic systems?

- Start with *deterministic components*.
- Look for *methods to combine* them that *preserve determinacy*.

Exercise

Consider the process $P \mid Q$ where P, Q are as follows.

1. $P = a.b$, $Q = a$.
2. $P = a$, $Q = \overline{a}$.
3. $P = a + b$, $Q = \overline{a}$.

Are P, Q, and $(P \mid Q)$ determinate?

Sorting

Sorting information is useful when trying to combine processes so as to preserve some property such as determinacy.

Let $\mathcal{L}$ be the set of visible actions and $L, L', \ldots$ range over $2^{\mathcal{L}}$.

Definition We say that a process P has sort L if all the actions performed by P and its derivatives lie in $L \cup \{\tau\}$.

Remarks on sorting

- In CCS, it is easy to provide an *upper bound for sorting* since:

$$P : \mathit{fn}(P) \cup \overline{\mathit{fn}(P)}$$

  where $\mathit{fn}(P)$ are the *free names* in P.

- Sorting is *closed under intersection*: if $P : L_i$ for $i = 1, 2$ then $P : L_1 \cap L_2$.

- Thus each process has a *minimum sort*.

- In general, the minimum sort *cannot be computed* because CCS can simulate Turing machines (TM) and the firing of a transition may correspond to the TM reaching the halting state…

- We discuss a method to compute an *over-approximation* of the minimum sort that we denote with $\mathcal{L}(P)$.

Computing the over-approximation

- Non-trivial programs in CCS are given via a *system of recursive equations*:

$$A(a_1, \ldots, a_n) = P$$

  where the names $a_1, \ldots, a_n$ are all distinct and $fn(P) \subseteq \{a_1, \ldots, a_n\}$.

- An *assignment* ρ is a function that associates with every thread identifier A of arity n a function $\rho(A)$ that takes a vector of n names $(b_1, \ldots, b_n)$ and produces a subset $\rho(A)(b_1, \ldots, b_n)$ of

$$\{b_1, \ldots, b_n, \bar{b}_1, \ldots \bar{b}_n\}$$

- The *least assignment* $\rho_\emptyset$ is the function where the 'subset' produced is always the empty set: $\rho_\emptyset(A)(b_1, \ldots, b_n) = \emptyset$.

- We define the sort $[\![P]\!]\rho$ of a process P *relatively* to an assignment ρ:

$$
\begin{aligned}
[\![0]\!]\rho &= \emptyset \\
[\![\alpha.P]\!]\rho &= \begin{cases} [\![P]\!]\rho & \text{if } \alpha = \tau \\ \{\alpha\} \cup [\![P]\!]\rho & \text{otherwise} \end{cases} \\
[\![P_1 + P_2]\!]\rho &= [\![P_1]\!]\rho \cup [\![P_2]\!]\rho \\
[\![P_1 \mid P_2]\!]\rho &= [\![P_1]\!]\rho \cup [\![P_2]\!]\rho \\
[\![\nu a\ P]\!]\rho &= [\![P]\!]\rho \backslash \{a, \overline{a}\} \\
[\![A(\mathbf{b})]\!]\rho &= \rho(A)(\mathbf{b})
\end{aligned}
$$

- Now we compute iteratively $\rho_0 = \rho_\emptyset$ and ρ_{n+1} so that:

$$\rho_{n+1}(A)(\mathbf{a}) = [\![P]\!]\rho_n$$

  for all identifiers A defined by an equation $A(\mathbf{a}) = P$.

- This defines a *growing sequence* (check this!) that is guaranteed *to converge* after finitely many steps to a *least fixed point* ρ since $\rho_n(A)(\mathbf{a}) \subseteq \{\mathbf{a}\} \cup \overline{\{\mathbf{a}\}}$ which is a *finite set*.

Example

- We consider the system composed of one equation:

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

- Then

$$\begin{aligned}
&\rho_1(A)(a,b)\\
&= [\![a.\nu c\ (A(a,c) \mid \overline{b}.A(c,b))]\!]\rho_\emptyset\\
&= \{a\} \cup (\rho_\emptyset(A)(a,c) \cup \{\overline{b}\} \cup \rho_\emptyset(A)(c,b)) \backslash \{c,\overline{c}\}\\
&= \{a,\overline{b}\}
\end{aligned}$$

- The following iteration reaches the *fixed point*:

$$
\begin{aligned}
&\rho_2(A)(a,b)\\
&= [\![a.\nu c\ (A(a,c) \mid \overline{b}.A(c,b))]\!]\rho_1\\
&= \{a\} \cup (\rho_1(A)(a,c) \cup \{\overline{b}\} \cup \rho_1(A)(c,b))\backslash\{c,\overline{c}\}\\
&= \{a\} \cup (\{a,\overline{c}\} \cup \{\overline{b}\} \cup \{c,\overline{b}\})\backslash\{c,\overline{c}\}\\
&= \{a,\overline{b}\}
\end{aligned}
$$

Thus $\mathcal{L}(P) = \{a,\overline{b}\}$.

Some sufficient conditions for building determinate processes

Proposition Suppose P, Q, P_i are determinate processes for $i \in I$. Then:

1. $0, \alpha.P, \nu a\ P$ are determinate.
2. $\Sigma_{i \in I} \ell_i.P_i$ is determinate if the ℓ_i are *all distinct*.
3. $P \mid Q$ is determinate if P, Q *do not communicate* and *do not share actions* (that is $\mathcal{L}(P) \cap \mathcal{L}(Q) = \emptyset$ and $\mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} = \emptyset$).
4. σP is determinate if σ is an *injective substitution* on the free names in P.

Proof idea

1. For instance, for $\nu a\ P$ one checks that if $\nu a\ P \stackrel{s}{\Rightarrow} Q$ then $P \stackrel{s}{\Rightarrow} P'$ and $Q = \nu a\ P'$.

2. Routine. Note that it is essential that all the actions are distinct and visible.

3. Because of the hypothesis on the sorting, an action of $(P_1 \mid P_2)$ can be attributed *uniquely* to either P_1 or P_2. Then we can rely on the determinacy of P_1 and P_2.

4. The transitions of P and σP are in perfect correspondance as long as σ is injective. Note that if σ is not injective then σP could perform some additional synchronisations.

Summary on determinacy

1. Deterministic processes are τ-inert.
2. For deterministic processes, *bisimulation* collapses to *trace equivalence*.
3. A simple method to extract *approximated sorting information*.
4. Use approximated sorting information to *build deterministic processes*.
5. Unfortunately, rules for *parallel composition* are *too restrictive*: no synchronisation.

Confluence

Refining the conditions

We want to allow some form of *communication*, but. . .

- We have to *avoid race conditions*: two processes compete on the same resource.
- We also have to *avoid that an action preempts* other actions.
- We introduce a notion of *confluence* that strengthens determinacy and is preserved by some form of communication (parallel composition + restriction).
- For instance,

  $$\nu a \ ((a + b) \mid \overline{a})$$

  will be *rejected* because $a + b$ is *not* confluent.

Confluence: rewriting vs. concurrency

- Notion reminiscent of *confluence* in term rewriting systems and λ-calculus (Church-Rosser theorem).
- By analogy one calls confluence the related theory in process calculi but bear in mind that:
  1. Confluence is relative to a *labelled transition system*.
  2. We close diagrams *up to equivalence*.

Definition of confluence

We start with a rather *natural* notion of confluence.

Definition (Conf 0) A process P is *confluent* if for every derivative Q of P we have:

$$\frac{Q \stackrel{\alpha}{\Rightarrow} Q_1 \quad Q \stackrel{\alpha}{\Rightarrow} Q_2}{Q_1 \stackrel{\tau}{\Rightarrow} Q_1' \quad Q_2 \stackrel{\tau}{\Rightarrow} Q_2' \quad Q_1' \approx Q_2'}$$

$$\frac{Q \stackrel{\alpha}{\Rightarrow} Q_1 \quad Q \stackrel{\beta}{\Rightarrow} Q_2 \quad \alpha \neq \beta}{Q_1 \stackrel{\beta}{\Rightarrow} Q_1' \quad Q_2 \stackrel{\alpha}{\Rightarrow} Q_2' \quad Q_1' \approx Q_2'}$$

Some properties

A first *sanity check* is to verify that the definition is invariant under *transitions* and *equivalence*.

Proposition

1. If P is confluent and $P \xrightarrow{\alpha} P'$ then P' is confluent.
2. If P is confluent and $P \approx P'$ then P' is confluent.

Proof idea (cf. similar proof for determinacy)

1. If Q is a derivative of P' then it is also a derivative of P.
2. It is enough to apply the fact that:

$$(P \approx P' \text{ and } P \stackrel{\alpha}{\Rightarrow} P_1) \text{ implies } (P' \stackrel{\alpha}{\Rightarrow} P_1' \text{ and } P_1 \approx P_1')$$

and the transitivity of $\approx$.

A first characterisation

We consider a first 'asymmetric' characterisation where the move from Q to Q_1 just concerns a *single action*.

Proposition (Conf 1) A process P is confluent iff for every derivative Q of P, we have:

$$\frac{Q \xrightarrow{\alpha} Q_1 \quad Q \stackrel{\alpha}{\Rightarrow} Q_2}{Q_1 \stackrel{\tau}{\Rightarrow} Q_1' \quad Q_2 \stackrel{\tau}{\Rightarrow} Q_2' \quad Q_1' \approx Q_2'}$$

$$\frac{Q \xrightarrow{\alpha} Q_1 \quad Q \stackrel{\beta}{\Rightarrow} Q_2 \quad \alpha \neq \beta}{Q_1 \stackrel{\beta}{\Rightarrow} Q_1' \quad Q_2 \stackrel{\alpha}{\Rightarrow} Q_2' \quad Q_1' \approx Q_2'}$$

Proof idea

- The diagrams of (Conf 1) are a particular case of (Conf 0).
- Thus we just have to show that the diagrams of (Conf 1) suffice to complete the diagrams of (Conf 0).
- We may proceed by induction on the length of the transition $Q \stackrel{\alpha}{\Rightarrow} Q_1$. For instance suppose $\alpha \neq \beta$, $\beta \neq \tau$, and

$$Q \xrightarrow{\tau} Q_1 \stackrel{\alpha}{\Rightarrow} Q_2 \quad Q \stackrel{\beta}{\Rightarrow} Q_3$$

- By (Conf 1),

$$Q_1 \stackrel{\beta}{\Rightarrow} Q_4 \quad Q_3 \stackrel{\tau}{\Rightarrow} Q_5 \quad Q_4 \approx Q_5$$

- By inductive hypothesis

$$Q_2 \stackrel{\beta}{\Rightarrow} Q_6 \quad Q_4 \stackrel{\alpha}{\Rightarrow} Q_7 \quad Q_4 \approx Q_7$$

- From $Q_4 \approx Q_5$ and $Q_4 \stackrel{\alpha}{\Rightarrow} Q_7$ we derive

$$Q_5 \stackrel{\alpha}{\Rightarrow} Q_8 \quad Q_7 \approx Q_8$$

- Therefore

$$Q_2 \overset{\beta}{\Rightarrow} Q_6 \quad Q_3 \overset{\alpha}{\Rightarrow} Q_8 \quad Q_6 \approx Q_8$$

as required.

Exercise

Complete the proof by considering the remaining cases.

Determinacy vs. Confluence

Confluence implies τ-inertness, and from this we can show that it implies determinacy too.

Proposition Suppose P is *confluent*. Then P is:

1. τ-*inert*, and
2. *determinate*.

Reminder

A relation R is a *weak bisimulation up to* $\approx$ if

$$\frac{P\ R\ Q \quad P \stackrel{\alpha}{\Rightarrow} P'}{Q \stackrel{\alpha}{\Rightarrow} Q' \quad P'(\approx \circ R \circ \approx)Q'}$$

(and symmetrically for Q).

NB It is important that we work with the *weak moves* on both sides, otherwise the relation R is *not* guaranteed to be contained in $\approx$. E.g. consider

$$R = \{(\tau.a, 0)\}$$

Proof idea

1. We want to show that $P \overset{\tau}{\Rightarrow} Q$ implies $P \approx Q$.
   - We show that
   $$R = \{(P, Q) \mid P \overset{\tau}{\Rightarrow} Q\}$$
   is a weak bisimulation up to $\approx$.
   - It is clear that whatever Q does, P can do too with some extra moves.
   - Suppose, for instance, $P \overset{\alpha}{\Rightarrow} P_1$ with $\alpha \neq \tau$ (case $\alpha = \tau$ left as exercise).
   - By (Conf 0),
   $$Q \overset{\alpha}{\Rightarrow} Q_1 \quad P_1 \overset{\tau}{\Rightarrow} P_2 \quad Q_1 \approx P_2$$
   - That is
   $$P_1 (R \circ \approx) Q_1$$

2. We want to show that if P is confluent then it is determinate.

   - Suppose $P \stackrel{s}{\Rightarrow} P_i$ for $i = 1, 2$ and $s \in \mathcal{L}^*$.
   - We proceed by *induction* on the length $|s|$ of s.
   - If $|s| = 0$ and $P \stackrel{\tau}{\Rightarrow} P_i$ for $i = 1, 2$ then *by confluence*

$$P_i \stackrel{\tau}{\Rightarrow} P_i' \quad i = 1, 2 \quad P_1' \approx P_2'$$

   - By *τ-inertness*

$$P_1 \approx P_1' \approx P_2' \approx P_2$$

- For the inductive case, suppose $P \stackrel{\ell}{\Rightarrow} P'_i \stackrel{r}{\Rightarrow} P_i$ for $i = 1, 2$.
- As in the basic case we derive that $P'_1 \approx P'_2$.
- By *weak bisimulation*, $P'_2 \stackrel{r}{\Rightarrow} P''_2$ and $P''_2 \approx P_1$.
- By *inductive hypothesis*, $P_2 \approx P''_2$.
- Thus $P_2 \approx P''_2 \approx P_1$ as required.

τ-inertness and confluence

Assuming that the process is τ-inert we can simplify a bit more the commuting diagrams. Let

$$P \stackrel{\alpha]}{\Rightarrow} P' = \begin{cases} P \stackrel{\tau}{\Rightarrow} P' & \text{if } \alpha = \tau \\ P \stackrel{\tau}{\Rightarrow} \cdot \stackrel{\ell}{\rightarrow} & \text{if } \alpha = \ell \end{cases}$$

Thus in $\stackrel{\ell]}{\Rightarrow}$, there is no τ action after ℓ.

Exercise (Conf 2)

Show that a process P is confluent iff it is τ-inert and for all its derivatives Q we have:

$$\frac{Q \xrightarrow{\alpha} Q_1 \qquad Q \overset{\alpha]}{\Longrightarrow} Q_2}{Q_1 \approx Q_2}$$

$$\frac{Q \xrightarrow{\alpha} Q_1 \quad Q \overset{\beta]}{\Longrightarrow} Q_2 \quad \alpha \neq \beta}{Q_1 \overset{\beta]}{\Longrightarrow} Q_1' \quad Q_2 \overset{\alpha]}{\Longrightarrow} Q_2' \quad Q_1' \approx Q_2'}$$