MPRI C-2-3- Concurrence - 2007-2008

Lectures 13-16 (Determinacy and Synchrony)

Roberto Amadio

Université Paris-Diderot

Laboratoire Preuves, Programmes et Systèmes

Programme of these lectures

We will cover the notions of:

- Determinacy, Confluence, and Linearity.
- Synchrony and Time.

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

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 input alphabet.

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

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

Depending on the compilation, the design of the virtual machine, the processors timing,... we might always run $\overline{d}$ rather than $\overline{c}$ (or the other way around).

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

- For the time being, 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 *labelled transitions*.

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:

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

- If $P \stackrel{s}{\Rightarrow} Q$ we say that Q is a *derivative* of P.

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

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

NB This definition relies on the notion of *labelled transition system*. In $P \xrightarrow{\ell} P'$, ℓ represents a *minimal* and *deterministic* interaction with the environment and P' is the *residual* after the interaction.

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

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

   - By definition of *weak bisimulation*:

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

     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:

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

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

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

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

- 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 $(b_1, \dots, b_n)$ and produces a subset $\rho(A)(b_1, \dots, b_n)$ of

$$\{b_1, \dots, b_n, \overline{b}_1, \dots \overline{b}_n\}$$

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

- 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 correspondence 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

$$P \stackrel{s}{\Rightarrow} P' \stackrel{\tau}{\Rightarrow} P'' \;\Rightarrow\; P' \approx P''$$

2. For *deterministic* processes,

$$\textit{bisimulation} \;=\; \textit{trace equivalence}.$$

3. A simple iterative method to extract from a process P an *approximated sorting information*

$$\mathcal{L}(P) \subseteq \mathit{fn}(P) \cup \overline{\mathit{fn}(P)}\ .$$

4. We rely on the 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 (while being *deterministic*).

Confluence: rewriting vs. concurrency

- Notion reminiscent of *confluence* in term rewriting systems and λ-calculus (Church-Rosser theorem)

$$\frac{t \xrightarrow{*} t_1, \quad t \xrightarrow{*} t_2}{\exists\, s \;\; (t_1 \xrightarrow{*} s, \quad t_2 \xrightarrow{*} s)}$$

- 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 define a notion of *action difference*:

$$\alpha \backslash \beta = \begin{cases} \alpha & \text{if } \alpha \neq \beta \\ \tau & \text{otherwise} \end{cases}$$

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

$$\frac{Q \stackrel{\alpha}{\Rightarrow} Q_1 \quad Q \stackrel{\beta}{\Rightarrow} Q_2}{\exists\, Q_1', Q_2' \;\; (Q_1 \stackrel{\beta\backslash\alpha}{\Rightarrow} Q_1' \quad Q_2 \stackrel{\alpha\backslash\beta}{\Rightarrow} Q_2' \quad Q_1' \approx Q_2')}$$

NB If $\alpha = \beta$ then we close the diagram with τ actions only.

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 $\approx$.

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

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

(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 $P \stackrel{\tau}{\Rightarrow} Q$ implies $P \approx Q$.

   - We show that
   $$R = \{(P, Q) \mid P \stackrel{\tau}{\Rightarrow} Q\}$$
   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, $P \stackrel{\alpha}{\Rightarrow} P_1$ with $\alpha \neq \tau$ (case $\alpha = \tau$ left as exercise).
   - By (Conf 0),
   $$Q \stackrel{\alpha}{\Rightarrow} Q_1 \quad P_1 \stackrel{\tau}{\Rightarrow} P_2 \quad Q_1 \approx P_2$$
   - 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 τ-inertness*

$$P_1 \approx P \approx P_2 \ .$$

- For the inductive case, suppose $P \stackrel{\ell}{\Rightarrow} P_i' \stackrel{r}{\Rightarrow} P_i$ for $i = 1, 2$.
- By confluence and τ-inertness, 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.

Exercise

We have seen that confluence implies determinacy which implies τ-inertness. Give examples that show that these implications cannot be reversed.

Characterisations of Confluence

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:

$$\frac{Q \xrightarrow{\alpha} Q_1 \quad Q \stackrel{\beta}{\Rightarrow} Q_2}{\exists\, Q_1', Q_2' \;(\; Q_1 \stackrel{\beta \backslash \alpha}{\Rightarrow} Q_1' \quad Q_2 \stackrel{\alpha \backslash \beta}{\Rightarrow} Q_2' \quad Q_1' \approx Q_2' \;)}$$

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

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

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

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

- Therefore

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

as required.

Exercise

Consider another case of the proof. For instance, when $Q \xrightarrow{\alpha} Q_1 \stackrel{\tau}{\Rightarrow} Q_2$.

Difference of sequences

In another direction we seek a more general definition of confluence where one commutes *sequences of actions*.

- Let $r, s \in \mathcal{L}^*$. To compute the *difference* $r \backslash s$ of r by s we scan r from left to right deleting each label which occurs in s taking into account the multiplicities (cf. difference of *multi-sets*).

$$
\begin{aligned}
(\epsilon \backslash s) &= \epsilon \\
(\ell r \backslash s) &= \begin{cases} \ell \cdot (r \backslash s) & \text{if } \ell \notin s \\ r \backslash (s_1 \cdot s_2) & \text{if } s = s_1 \ell s_2, \ell \notin s_1 \end{cases}
\end{aligned}
$$

- For instance

$$aba \backslash ca = ba \qquad ca \backslash aba = c$$

Exercise

Let $r, s, t \in \mathcal{L}^*$. Show that:

1. $(rs)\backslash(rt) = s\backslash t$.
2. $r\backslash(st) = (r\backslash s)\backslash t$.
3. $(rs)\backslash t = (r\backslash t)(s\backslash(t\backslash r))$.

A final characterisation of confluence

Proposition (Conf 2) A process P is *confluent* iff for all $r, s \in \mathcal{L}^*$ we have:

$$\frac{P \stackrel{r}{\Rightarrow} P_1 \qquad P \stackrel{s}{\Rightarrow} P_2}{\exists\, P_1', P_2' \;\; P_1 \stackrel{s \backslash r}{\Rightarrow} P_1' \qquad P_2 \stackrel{r \backslash s}{\Rightarrow} P_2' \qquad P_1' \approx P_2'}$$

Proof idea

($\Leftarrow$) It suffices to check that if P has property (Conf 2) then its derivatives have it too.

- Suppose $P \stackrel{t}{\Rightarrow} Q$ for $t \in \mathcal{L}^*$.
- Suppose further $Q \stackrel{r}{\Rightarrow} Q_1$ and $Q \stackrel{s}{\Rightarrow} Q_2$.
- By composing diagrams and applying (Conf 2) we get:

$$Q_1 \stackrel{(ts\backslash tr)}{\Rightarrow} Q_1' \quad Q_2 \stackrel{(tr\backslash ts)}{\Rightarrow} Q_2' \quad Q_1' \approx Q_2'$$

- Applying the previous exercise we derive, *e.g.*:

$$ts\backslash tr = s\backslash r$$

($\Rightarrow$) We proceed in three steps.

1. By induction on $|s|$ we show that:

$$\frac{P \overset{\tau}{\Rightarrow} P_1 \quad P \overset{s}{\Rightarrow} P_2}{\exists\, P_1', P_2' \;\; P_1 \overset{s}{\Rightarrow} P_1' \quad P_2 \overset{\tau}{\Rightarrow} P_2' \quad P_1' \approx P_2'}$$

2. Then, again by induction on $|s|$, we show that:

$$\frac{P \overset{\ell}{\Rightarrow} P_1 \quad P \overset{s}{\Rightarrow} P_2}{\exists\, P_1', P_2' \;\; P_1 \overset{s \backslash \ell}{\Rightarrow} P_1' \quad P_2 \overset{\ell \backslash s}{\Rightarrow} P_2' \quad P_1' \approx P_2'}$$

3. Finally we prove the commutation of diagram (Conf 2) by induction on $|r|$ when $P \overset{r}{\Rightarrow} P_1$

Exercise

Complete the proof.

Building confluent processes

Building confluent processes

Next, we return to the issue of building confluent (and therefore determinate) processes.

Proposition If P, Q are confluent processes then so are:

1. 0, $\alpha.P$.
2. $\nu a\ P$.
3. σP where σ is an injective substitution on the free names of P.

Proof Routine analysis of transitions (cf. similar statement for determinacy).

Remark on sum

- In general, $a + b$ is *determinate* but it is *not confluent* for $a \neq b$
- To have confluence, one may consider a special kind of 'commuting sum'

$$(a \mid b).P =_{def} a.b.P + b.a.P$$

Restricted composition

We allow a parallel composition with restriction

$$\nu a_1, \ldots, a_n \ (P \mid Q)$$

when:

1. P and Q do not share visible actions:

$$\mathcal{L}(P) \cap \mathcal{L}(Q) = \emptyset$$

2. P and Q may interact only on the names in $\{\mathbf{a}\}$:

$$\mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \subseteq \{a_1, \ldots, a_n\}$$

Proposition Confluence is preserved by restricted composition.

Proof idea

- First we observe that any derivative of $\nu\mathbf{a}\ (P \mid Q)$ will have the shape $\nu\mathbf{a}\ (P' \mid Q')$ where P' is a derivative of P and Q' is a derivative of Q.
- Since sorting is preserved by transitions, the two conditions on sorting will be satisfied.
- Therefore, it is enough to show that the diagrams in (Conf 1) commute for processes of the shape $R = \nu\mathbf{a}\ (P \mid Q)$ under the given hypotheses.

- We consider one case. Suppose:

$$R \xrightarrow{\ell} \nu a\ (P_1 \mid Q), \qquad \text{because } P \xrightarrow{\ell} P_1$$

- Also assume:

$$R \stackrel{\ell}{\Rightarrow} \nu a\ (P_2 \mid Q_2)$$

  because $P \stackrel{s\ell r}{\Rightarrow} P_2$ and $Q \stackrel{\overline{s}\cdot\overline{r}}{\Rightarrow} Q_2$ with $s \cdot r \in \{\mathbf{a}, \overline{\mathbf{a}}\}^*$ and $\ell \notin \{\mathbf{a}, \overline{\mathbf{a}}\}$.

- Since P is confluent we have:

$$\frac{P \xrightarrow{\ell} P_1 \quad P \stackrel{s\ell r}{\Rightarrow} P_2}{P_1 \stackrel{sr}{\Rightarrow} P_1' \quad P_2 \stackrel{\tau}{\Rightarrow} P_2' \quad P_1' \approx P_2'}$$

- Then we have that:

$$\nu\mathbf{a}\ (P_1 \mid Q) \stackrel{\tau}{\Rightarrow} \nu\mathbf{a}\ (P_1' \mid Q_2) \approx \nu\mathbf{a}\ (P_2' \mid Q_2)$$

thus closing the diagram (note that we use the congruence properties of $\approx$).

Exercise

Consider another case of the proof, for instance:

$$\begin{array}{ll} \nu a \ (P \mid Q) \xrightarrow{\tau} \nu a \ (P \mid Q) & \text{as } P \xrightarrow{a} P_1, \quad Q \xrightarrow{\overline{a}} Q_1 \\ \nu a \ (P \mid Q) \stackrel{\tau}{\Rightarrow} \nu a \ (P_2 \mid Q_2) & \text{as } P \xrightarrow{s} P_2, \quad Q \stackrel{\overline{s}}{\Rightarrow} Q_2 \end{array}$$

A case study: Kahn networks

Point-to-point communication for every channel there is at most one sender and one receiver.

Ordered buffers of unbounded capacity send is non blocking and the order of emission is preserved at the reception.

Each thread may:

1. perform arbitrary *sequential deterministic computation*,
2. *insert a message in a buffer*,
3. *receive a message from a buffer*. If the buffer is empty then the thread *must* suspend,

A thread *cannot* try to receive a message from several channels at once.

Semantics (informal)

- We regard the unbounded buffers as finite or infinite words over some data domain.
- The nodes of the networks are functions over words.
- Kahn observes that the associated system of equations has a least fixed point.

- Kahn networks is an important (practical) case where *concurrency* and *determinism* coexist. For instance, they are frequently used in the *signal processing* community.
- We refer to the course on Synchronous Systems for more information on Kahn networks and related applications.
- Our modest goal is to:
  1. Formalise Kahn networks as a fragment of CCS.
  2. Apply the developed theory to show that the fragment is confluent and therefore deterministic.

CCS formalisation of Kahn networks

- We will work with a 'data domain' that contains just one element.
- The generalisation to arbitrary data domains is not difficult, but we would need to formalise determinacy and confluence in the framework of *CCS with values* (a word on this later...).
- First problem: how do we model *unbounded buffers* in CCS?

Representing an unbounded buffer in CCS

A unbounded buffer taking inputs on a and producing outputs on b can be written as (yes, you have already seen this!):

$$Buf(a, b) = a.\nu c\ (Buf(a, c) \mid \bar{b}.Buf(c, b))$$

- We will write more suggestively $a \mapsto b$ for $Buf(a, b)$, assuming $a \neq b$.
- We have already analysed the sorting of this system:

$$\mathcal{L}(a \mapsto b) = \{a, \bar{b}\}$$

- Moreover, this system falls within the class of *confluent processes* we have considered as it relies on *restricted composition*:

$$\begin{array}{rcl} \mathcal{L}(a \mapsto c) \cap \mathcal{L}(\bar{b}.c \mapsto b) & = & \emptyset \\ \mathcal{L}(a \mapsto c) \cap \overline{\mathcal{L}(\bar{b}.c \mapsto b)} & \subseteq & \{c, \bar{c}\} \end{array}$$

- We would like to show that $a \mapsto b$ works indeed as an unbounded buffer.

- Let $\overline{c}^n = \overline{c} \dots \overline{c}$, n times, $n \geq 0$.

- We should have:

$$P(n) = \nu a \ (\overline{a}^n \mid a \mapsto b) \approx \overline{b}^n$$

- This is an interesting exercise because:
  - The process $P(n)$ has a non trivial *dynamics*.
  - We can prove the statement just by considering *finite traces*.

Computing the trace of $P(n)$

- Obviously:
$$tr(\overline{b}^n) = \{\epsilon, \overline{b}, \overline{bb}, \ldots, \overline{b}^n\}$$

- We have $\mathcal{L}(P(n)) = \{\overline{b}\}$, thus $tr(P(n))$ is a non-empty prefix closed set of finite words over $\overline{b}$.

- For $n = 0$, $P(n)$ can do no transition.

- For $n > 0$ we need to generalise a bit the form of the process $P(n)$. Let $Q(n, m)$ be a process of the form:

$$Q(n, m) = \nu a, c_1, \ldots, c_m \; (\overline{a}^n \mid a \mapsto c_1 \mid \cdots \mid c_m \mapsto b)$$

for $m \geq 0$. Note that $P(n) = Q(n, 0)$ and $Q(0, k) \approx 0$ for any k.

- Moreover

$$Q(n,m) \overset{\bar{b}}{\Rightarrow} Q(n-1, 2m+1)$$

- Thus

$$P(n) \overset{\bar{b}}{\Rightarrow} \cdots \overset{\bar{b}}{\Rightarrow} Q(0, 2^n - 1) \approx 0$$

- Because $P(n)$ is confluent we can conclude that:

$$tr(P(n)) = tr(\bar{b}^n)$$

CCS processes representing Kahn networks

We define a class of CCS processes sufficient to represent Kahn networks.

- Let KP be the least set of processes such that $0 \in KP$ and if $P, Q \in KP$ and α is an action then
  1. $\alpha.P \in KP$,
  2. $\nu\mathbf{a}\ (P \mid Q) \in KP$ provided $\mathcal{L}(P) \cap \mathcal{L}(Q) = \emptyset$ and $\mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \subseteq \{\mathbf{a}, \overline{\mathbf{a}}\}$,
  3. $B(\mathbf{b}) \in KP$ if the names $\mathbf{b}$ are all distinct.
- We admit a recursive equation $A(\mathbf{a}) = P$ only if $P \in KP$.
- We admit processes that are in KP and that depend on recursive equations of the shape above.
- It is easily checked that $a \mapsto b$ is admissible and that Kahn processes are confluent.

From a Kahn network to CCS process

Suppose we have a Kahn network with three nodes, and the following ports and behaviours where we use ! for output and ? for input.

Node	Ports	Behaviours
1	$?a, ?b, ?c, !d, !e, !f$	$A_1 = ?a.!d.!e.?b.?c.!f.A_1$
2	$!b, ?d$	$A_2 = ?d.!b.A_2$
3	$!c, ?e$	$A_3 = ?e.!c.A_3$

The corresponding CCS system relies on the equations for *Buf* plus:

$$
\begin{aligned}
A_1(a,b,c,d,e,f) &= a.\overline{d}.\overline{e}.b.c.\overline{f}.A_1(a,b,c,d,e,f)\\
A_2(b,d) &= d.\overline{b}.A_2(b,d)\\
A_3(c,e) &= e.\overline{c}.A_3(c,e)
\end{aligned}
$$

The sorting is easily derived:

$$
\begin{aligned}
\mathcal{L}(A_1(a,b,c,d,e,f) &= \{a,b,c,\overline{d},\overline{e},\overline{f}\}\\
\mathcal{L}(A_2(b,d)) &= \{\overline{b},d\}\\
\mathcal{L}(A_3(c,e)) &= \{\overline{c},e\}
\end{aligned}
$$

To build the system, we have to introduce a buffer before every input channel. Thus the initial configuration is:

$$
\begin{array}{l}
\nu a', b, b', c, c', d, d', e, e' \\
(a \mapsto a' \mid b \mapsto b' \mid c \mapsto c' \mid d \mapsto d' \mid e \mapsto e' \mid \\
A_1(a', b', c', d, e, f) \mid A_2(b, d') \mid A_3(c, e')\ )
\end{array}
$$

It is easily checked that the resulting processes belong to the class *KP*.

NB Via recursion, we can represent Kahn networks with a dynamically changing number of nodes (*e.g.*, the buffer).

Summary on building confluent processes

To build confluent processes we can use:

- nil and input prefix,
- restricted composition,
- injective recursive calls,
- recursive equations $A(\mathbf{a}) = P$, where P is built according to the rules above.

This class of processes is enough to represent Kahn networks.