Lecture 10: Confluence continued

Termination and Local confluence

- It would be nice if we just had to consider diagrams where *both* Q_1 *and* Q_2 take *one step*.
- By analogy with rewriting theory (Newman's lemma), we seek a situation where *local confluence* plus *termination* entails *confluence*.

Definition

- A process P is *terminating* (or strongly normalising) if there is no infinite sequence
$$P \xrightarrow{\tau} P_1 \xrightarrow{\tau} \cdots$$
- A process P is *fully terminating* if all its derivatives are terminating.

Exercise

Consider again the process

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

Is the process $A(a, b)$ (fully) terminating? Consider the cases $a \neq b$ and $a = b$.

Proposition (Conf 3) Let P be a *fully terminating* process. Then P is *confluent* iff it is *τ-inert* and for all its derivatives Q we have:

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

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

Proof idea

($\Rightarrow$)

- If P is confluent then it is τ-inert.
- The hypotheses in (Conf 3) are particular cases of (Conf 0) and the conclusions are the same, using τ-inertness.

($\Leftarrow$) We show directly that the diagrams of (Conf 0) commute.

- Define $P > P'$ if P reduces to P' by at least a τ-action.
- Because of *full convergence*, this is a *well-founded order*.
- We can show the commutation of the diagrams (Conf 0), *by induction* on the well-founded order.

- The *base case of the induction* is when the process cannot perform τ-reductions. For instance, suppose

$$Q \xrightarrow{\alpha} Q_1 \stackrel{\tau}{\Rightarrow} Q_2 \quad Q \xrightarrow{\beta} Q_3 \stackrel{\tau}{\Rightarrow} Q_4 \quad \alpha \neq \beta$$

  – By local confluence,

$$Q_1 \stackrel{\beta}{\Rightarrow} Q_5 \quad Q_3 \stackrel{\alpha}{\Rightarrow} Q_6 \quad Q_5 \approx Q_6$$

  – By τ-inertness

$$Q_1 \approx Q_2 \quad Q_3 \approx Q_4$$

  – By weak bisimulation

$$Q_2 \stackrel{\beta}{\Rightarrow} Q_7 \quad Q_5 \approx Q_7 \quad Q_4 \stackrel{\alpha}{\Rightarrow} Q_8 \quad Q_6 \approx Q_8$$

  and by transitivity of weak bisimulation $Q_7 \approx Q_8$.

- Next, let us consider a situation where the *inductive hypothesis* applies. Suppose

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

- By τ-inertness, $Q \approx Q_2$ and therefore

$$Q_2 \overset{\alpha}{\Rightarrow} Q_4 \quad Q_1 \approx Q_4$$

- By inductive hypothesis

$$Q_3 \overset{\alpha}{\Rightarrow} Q_5 \quad Q_4 \overset{\beta}{\Rightarrow} Q_6 \quad Q_5 \approx Q_6$$

- Since $Q_1 \approx Q_4$, we derive that

$$Q_1 \overset{\beta}{\Rightarrow} Q_7 \quad Q_6 \approx Q_7$$

  and by transitivity of weak bisimulation, $Q_5 \approx Q_7$.

Exercise

Complete the proof by considering the remaining cases.

Exercise

Consider the process:

$$A = a.b + \tau.(a.c + \tau.A)$$

Check whether A is:

1. τ-inert,
2. locally confluent,
3. (fully) terminating.
4. determinate.
5. confluent.

Difference of sequences

Finally, 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 4) A process P is *confluent* iff for all $r, s \in \mathcal{L}^*$ we have:

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

Proof idea

($\Leftarrow$) It suffices to check that if P has property (Conf 4) 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 4) 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:

$$\frac{P \stackrel{\tau}{\Rightarrow} P_1 \quad P \stackrel{s}{\Rightarrow} P_2}{P_1 \stackrel{s}{\Rightarrow} P_1' \quad P_2 \stackrel{\tau}{\Rightarrow} P_2' \quad P_1' \approx P_2'}$$

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

$$\frac{P \stackrel{\ell}{\Rightarrow} P_1 \quad P \stackrel{s}{\Rightarrow} P_2}{P_1 \stackrel{s \backslash \ell}{\Rightarrow} P_1' \quad P_2 \stackrel{\ell \backslash s}{\Rightarrow} P_2' \quad P_1' \approx P_2'}$$

3. Finally we prove the commutation of diagram (Conf 4) by induction on $|r|$ when $P \stackrel{r}{\Rightarrow} P_1$

Exercise

Complete the proof.

Summary on the definitions of confluence

1. We have 5 alternative characterisations of confluence.

2. A *confluent* process is always τ-*inert* and *determinate*.

3. Having checked τ-inertness, the simplest commuting diagrams to consider are:

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

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

4. Moreover, if we have (τ-inertness) and *full termination*, then it is enough to consider the following diagrams:

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

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

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:

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

thus closing the diagram.

Exercise

Complete the proof.

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 of Marc Pouzet 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 $\overline{c}^n = \overline{c} \ldots \overline{c}$, 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
$$Q(n,m) \stackrel{\bar{b}}{\Rightarrow} Q(n-1, 2m+1)$$
- Thus
$$P(n) \stackrel{\bar{b}}{\Rightarrow} \cdots \stackrel{\bar{b}}{\Rightarrow} Q(0, 2^n - 1) \approx 0$$
- 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:

$$\nu a', b, b', c, c', d, d', e, e'$$
$$(a \mapsto a' \mid b \mapsto b' \mid c \mapsto c' \mid d \mapsto d' \mid e \mapsto e' \mid$$
$$A_1(a', b', c', d, e, f) \mid A_2(b, d') \mid A_3(c, e')\ )$$

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.

Confluence in CCS with value passing

Consider the process P

$$P = a(b).\overline{a}b$$

- It seems reasonable to regard P as *determinate*.
- However, according to a straightforward extension of the concept of confluence to CCS with values, P is *not confluent*.
- Relaxation: do not require confluence for distinct input actions with the same subject.

Confluence in the π-calculus

- Consider

$$P = \nu a \ (\bar{b}a \mid \bar{c}a)$$

- Again, a straightforward definition of confluence would lead us to conclude that P is *not* confluent.

- One has to take into account the fact that an output may free names bound in another output action.

References

- This lecture is largely based on chapter 11 of:

  Robin Milner. Communication and Concurrency, Prentice-Hall, 1989.

- Amazingly, this book does not refer to Kahn networks which were introduced in:

  Gilles Kahn. The semantics of a simple language for parallel programming, IFIP Conf. on Information Processing 74, North-Holland, 1974.

  Incidentally, synchronous data flow languages such as Lustre can be regarded as a refinement of this model.

- A rather complete study of the notion of confluence in the more general framework of the π-calculus is in:

  Anna Philippou, David Walker. On confluence in the pi-Calculus. ICALP 1997: 314-324. (See also Anna Philippou PhD thesis, University of Warwick 1996).

- This builds on the PhD thesis of Sanderson and Tofts (in Edinburgh in the early 90's) where notions of confluence for CCS with value passing were proposed.

- This paper observes that full termination plus τ-inertness plus local confluence imply confluence.

- The proof discussed here is slightly different and follows the pattern of the classical proof of Newman's lemma.

- There have been various attempts to provide static conditions that guarantee (partial) confluence in the π-calculus. Besides the quoted work by Philippou and Walker, two early references are:
  - Naoki Kobayashi, Benjamin C. Pierce, David N. Turner. Linearity and the pi-calculus. ACM Transactions on Programming Languages and Systems (TOPLAS), 21(5), 1999. Extended abstract in POPL 1996.
  - Uwe Nestmann. On determinacy and nondeterminacy in concurrent programming. PhD thesis, Universität Erlangen, 1996.
- There is still space to do research on this topic!