

# **A synchronous $\pi$ -calculus**

Roberto AMADIO

Université de Paris 7

Laboratoire Preuves, Programmes et Systèmes

## Synchronous?

A standard classification of concurrent/distributed systems:

- Timing assumptions on relative speed of processes:
  - *Synchronous*.
  - Asynchronous.
  - Partially synchronous, ...
- Inter-process communication:
  - Shared references.
  - Channel based: rendez-vous (a.k.a. *synchronous*), bounded/unbounded, ordered/unordered buffers.
  - Signal based, ...

**Ref.** Lamport-Lynch, Handbook TCS.

## SCCS/Meije: an early example of synchronous process calculus

- An *action* is a function  $\alpha : \mathcal{A} \rightarrow \mathbf{Z}$  such that  $\alpha(a) = 0$  almost everywhere.
- The product of actions is:

$$(\alpha \cdot \beta)(a) = \alpha(a) + \beta(a)$$

- At each instant/round, each process *must* run exactly one action.

### Ref

- Milner, *Calculi for synchrony and asynchrony*, TCS, 1983.
- Austry-Boudol, *Algèbre de processus et synchronisation*, TCS, 1984.

## Programming a NOR gate in SCCS

$a$	$b$	$c = NOR(a, b)$
0	0	1
0	1	0
1	1	0
1	0	0

The process  $A_i$  for  $i = 0, 1$  has 4 inputs  $a_0, a_1, b_0, b_1$  and 2 outputs  $c_0, c_1$ .

$$A_i = \bar{c}_i : \mathbf{1} \quad \times \quad \sum_{i,j \in \{0,1\}} a_i \cdot b_j : A_{NOR(i,j)} \quad i = 0, 1$$

The result is emitted in the following instant.

## What about solving

*SCCS : CCS =? :  $\pi$  - calculus*

### Problems with SCCS

- Not a practical programming notation.
- Infinite data domain?
- Implementation model?

## A second generation model: Esterel/SL

- Processes interact through *signals*.
- They can wait for the *presence of a signal*.
- *Instant ends* when all processes are done or are waiting for a signal that will not come.
- Can react to the *absence of a signal* at the end of the instant.

### Ref

- Berry-Gonthier. The Esterel synchronous programming language. *Science of computer programming*, 1992.
- Boussinot-De Simone. The SL synchronous language. *IEEE Trans. on Software Engineering*, 1996.

## Programming a NOR gate in SL

$$\begin{aligned} N_0 &= \text{pause. } \textit{if } a \textit{ then } N_0 \\ &\quad \textit{else if } b \textit{ then } N_0 \\ &\quad \textit{else } N_1 \end{aligned}$$

$$N_1 = \bar{c}.N_0$$

**NB** We can *program* the boolean function *NOR* rather than writing down its *truth table*.

## SL and its evolution

- The language with *pure* signals is *deterministic*.
- Reasonable extension to *(infinite) data domains*. The resulting language becomes *non-deterministic*.
- Efficient *implementation model*.
- Embedded in many *programming environments*: C, C++, Scheme, ML.
- Significant *applications*: event-driven control, data flow, GUI, simulations, web services, multiplayer games.

### Some Ref

- Boussinot. Reactive C: an extension of C to program reactive systems. *Soft. Practice and Experience*, 1991.
- Mandel-Pouzet. Reactive ML, a reactive extension to ML. In *Proc. ACM PPDP*, 2005.

## Our equation revisited...

$SL : CCS =? : \pi - calculus$

**Some preliminary work** Put  $SL$  in a *process calculus form* comparable to  $CCS$ . Namely use a *CPS-translation* to:

- remove *control operators* (*;*, *when*, *watch*,...),
- reduce to a *tail-recursive form*.

Then provide:

- a *contextual semantics* (hence compositional).
- a *practical characterisation*.

**Ref** A. The SL synchronous language, revisited. Journal of Logic and Algebraic Programming, 2006.

## The $S\pi$ -calculus: a *synchronous* $\pi$ -calculus

Assume  $v_1 \neq v_2$  are two distinct values and

$$P = \nu s_1, s_2 (\overline{s_1}v_1 \mid \overline{s_1}v_2 \mid \\ s_1(x). (s_1(y). (s_2(z). A(x, y) , B(!s_1)) \\ , 0) \\ , 0)$$

$P$  is a  $\pi$ -calculus process if we forget about the **else branches of the read instructions.**

## Spot the differences...

$$P = \nu s_1, s_2 (\overline{s_1}v_1 \mid \overline{s_1}v_2 \mid s_1(x). (s_1(y). (s_2(z). A(x, y) , B(!s_1)), 0), 0)$$

- In  $\pi$ ,  $P$  reduces to

$$P_1 = \nu s_1, s_2 (s_2(z).0, A(\sigma(x), \sigma(y)), B(!s_1))$$

where  $\sigma(x), \sigma(y) \in \{v_1, v_2\}$  and  $\sigma(x) \neq \sigma(y)$ .

- In  $S\pi$ , *signals persist within the instant* and  $P$  reduces to

$$P_2 = \nu s_1, s_2 (\overline{s_1}v_1 \mid \overline{s_1}v_2 \mid (s_2(z).A(\sigma(x), \sigma(y)), B(!s_1)))$$

where  $\sigma(x), \sigma(y) \in \{v_1, v_2\}$ .

- In  $\pi$ ,  $P_1$  is now *deadlocked*.
- In  $S\pi$ , the *current instant ends* and we move to the following one

$$P_2 \mapsto P'_2 = \nu s_1, s_2 \mathbf{B}(\sigma(\ell))$$

where  $\sigma(\ell) \in \{[v_1; v_2], [v_2; v_1]\}$ .

- Thus at the end of the instant,  $!s_1$  becomes *a list of (distinct) values* emitted on  $s_1$  during the instant.
- For this reason,  $S\pi$  includes *lists has a primitive data structure*.

## Some derived operators

**pause**

$$\text{pause}.K = \nu s \ s.0, K$$

**internal choice**

$$P_1 \oplus P_2 = \nu s \ (s(x)[x = 0]P_1, P_2 \mid \bar{s}0 \mid \bar{s}1)$$

with suitable encodings of 0 and 1.

**await**

$$\text{await } s(x).P = s(x).P, A(\mathbf{x})$$

where  $A(\mathbf{x}) = s(x).P, A(\mathbf{x})$  with suitable conditions on the variables.

## Goal (refined)

- Provide a contextual *bisimulation* semantics.
- Characterise it via a labelled transition system and a *labelled bisimulation*.

## What is directly observable?

A synchronous program receives inputs at the beginning of each instant and provides outputs at the end of each instant. Therefore:

- *End of each instant* is observable.
- *Emission on a signal* is observable at the *end of the instant*.

**NB** If a sub-process *loops*, the end of instant is never reached and *nothing* is observable.

## Some standard concepts...

- *Internal reduction* within an instant:

$$P \xrightarrow{\tau} P', \quad P \xRightarrow{\tau} P'.$$

- *Commitment*:  $P \searrow \bar{s}$  if  $P$  can emit immediately on  $s$  (a commitment *persists* till the end of the instant).
- *Static context*  $C$ : a one hole context composed of parallel composition and name generation.

... and some less standard

- *Suspension* (or end of the instant):

$$P \downarrow \text{ if } P \text{ cannot reduce}$$

- *Weak suspension*

$$P \Downarrow \text{ if } P \xrightarrow{\tau} P' \text{ and } P' \downarrow$$

- *L-suspension*

$$P \Downarrow_L \text{ if } \exists Q (P \mid Q) \Downarrow$$

- *Evaluation at the end of the instant* (assuming  $P \downarrow$ ):

$$P \mapsto P'$$

## Barbed bisimulation

A symmetric relation  $\mathcal{R}$  on programs is a *barbed bisimulation* if whenever  $P \mathcal{R} Q$  the following holds:

$$(B1) \quad \frac{P \xrightarrow{\tau} P'}{\exists Q' \quad Q \xRightarrow{\tau} Q' \text{ and } P' \mathcal{R} Q'}$$

$$(B2) \quad \frac{P \searrow \bar{s}, \quad P \Downarrow_L}{\exists Q' \quad (Q \xRightarrow{\tau} Q', \quad Q' \searrow \bar{s}, \quad P \mathcal{R} Q')}$$

$$(B3) \quad \frac{P \downarrow, \quad P \mapsto P''}{\exists Q', Q'' \quad (Q \xRightarrow{\tau} Q', \quad Q' \downarrow, \quad P \mathcal{R} Q', \quad Q' \mapsto Q'', \quad P'' \mathcal{R} Q'')}$$

## Contextual bisimulation

A symmetric relation  $\mathcal{R}$  on programs is a *contextual bisimulation* if it is a *barbed bisimulation* (conditions (B1 – 3)) and moreover

$$\frac{P \mathcal{R} Q \quad C \text{ static context}}{C[P] \mathcal{R} C[Q]}$$

Denote with  $\approx_C$  the largest contextual barbed bisimulation.

## A recipe to build the labelled transition system

- Keep the evaluation relation  $\mapsto$  at the *end of the instant*.
- Extend the internal reduction with a lts  $\xrightarrow{\alpha}$  describing possible interactions within the instant.
  - *Actions* within an instant are the same as in the (polyadic)  $\pi$ -calculus.
  - *Transition rules* are also the same but those for *input* and *output*:

$$\frac{}{\bar{s}v \xrightarrow{\bar{s}v} \bar{s}v}$$

$$\frac{}{s(x).P, K \xrightarrow{sv} [v/x]P \mid \bar{s}v}$$

## Four rules for labelled bisimulation

A symmetric relation  $\mathcal{R}$  is a *labelled bisimulation* if whenever  $P \mathcal{R} Q$  the following holds:

$$(L1) \quad \frac{P \xrightarrow{\tau} P'}{\exists Q' (Q \xRightarrow{\tau} Q' \text{ and } P' \mathcal{R} Q')}$$
$$(L2) \quad \frac{P \xrightarrow{\nu \mathbf{t} \bar{s}v} P', \quad \{\mathbf{t}\} \cap \text{fn}(Q) = \emptyset, \quad P \Downarrow_L}{\exists Q' (Q \xRightarrow{\nu \mathbf{t} \bar{s}v} Q' \text{ and } P' \mathcal{R} Q')}$$

$$\begin{array}{c}
(L3) \quad \frac{P \xrightarrow{sv} P'}{\exists Q' ( Q \xRightarrow{sv} Q' \text{ and } P' \mathcal{R} Q' ) \text{ or} \\
( Q \xRightarrow{\tau} Q' \text{ and } P' \mathcal{R} ( Q' \mid \bar{s}v ) )} \\
(L4) \quad \frac{\begin{array}{c} S = \bar{s}_1 v_1 \mid \cdots \mid \bar{s}_n v_n, \quad n \geq 0 \\ P' = (P \mid S) \downarrow, \text{ and } P' \mapsto P'' \end{array}}{\exists Q', Q'' ( (Q \mid S) \xRightarrow{\tau} Q', Q' \downarrow, P' \mathcal{R} Q', \\
Q' \mapsto Q'', \text{ and } P'' \mathcal{R} Q'' )}
\end{array}$$

We denote with  $\approx_L$  the largest labelled bisimulation.

## Remarks on the definition

(L1) Standard.

(L2) If  $P \xrightarrow{\nu^t \bar{s}v} P'$ , it is equivalent to require  $P \Downarrow_L$  or  $P' \Downarrow_L$ . The reason for choosing  $\Downarrow_L$  rather than  $\Downarrow$  or  $\Downarrow$  is that the latter lead to an equivalence which is *not* preserved by parallel composition.

(L3) Inspired by the  $\pi$ -calculus with asynchronous communication. No reason to distinguish  $s().0, 0$  from  $0$ .

**Ref** A.-Castellani-Sangiorgi, On bisimulations for the asynchronous  $\pi$ -calculus. TCS. 1998.

(L4) Emitted values have an effect on computation at the *end of the instant*.

$$P = s_1().0, A(!s_2) \quad Q = s_1().0, A(\text{nil}) \quad A(l) = [l = \text{nil}]0, \overline{s_3}$$

- Then:

$$P \downarrow, \quad Q \downarrow, \quad P \mapsto A(\text{nil}), \quad Q \mapsto A(\text{nil})$$

- However:

$$P \mid \overline{s_2}^* \not\approx Q \mid \overline{s_2}^*$$

## Main theorem

Let  $P, Q$  be programs. Then  $P \approx_L Q$  if and only if  $P \approx_C Q$ .

**Ref** A. A synchronous  $\pi$ -calculus. Tech. Report. PPS, Université Paris 7.  
June 2006.

## Some side remarks

- In the *pure, deterministic* case, *labelled bisimulation* collapses with a *trace semantics*.
- For the fragment of the language *without recursion*, if the set of distinct values on signals read at the end of the instant is *finite*, then *labelled bisimulation is decidable*.

## Two key insights

- The observation of *signals* is similar to the observation of channels with *asynchronous communication*.
- Good match between *L-suspension* and *labelled bisimulation*:
  - Weaker versions do *not* lead to compositional semantics.
  - All processes that do not L-suspend are identified.
  - Never identifies a process that L-suspends with one that does not.