Linearity, Persistence and Testing Semantics in the Asynchronous π-Calculus.

Speaker : Jesús A. Aranda
Joint work with Diletta R. Cacciagrano, Flavio Corradini and Frank D. Valencia

AVISPA seminar, 2008

Outline

Linearity.

Linearity of messages and input processes.

In the π-calculus outputs (messages) and inputs are ***linear***.

- E.g. the parallel composition

$$\bar{x}z \mid x(y).P \mid x(w).Q$$

reduces either

- to

$$P\{z/y\} \mid x(w).Q$$

- or to

$$x(y).P \mid Q\{z/w\}$$

Linearity.

Linearity of messages and input processes.

In the π-calculus outputs (messages) and inputs are ***linear***.

- E.g. the parallel composition

$$\bar{x}z \mid x(y).P \mid x(w).Q$$

reduces either

- to

$$P\{z/y\} \mid x(w).Q$$

- or to

$$x(y).P \mid Q\{z/w\}$$

Linearity.

Linearity of messages and input processes.

In the π-calculus outputs (messages) and inputs are ***linear***.

- E.g. the parallel composition

$$\bar{x}z \mid x(y).P \mid x(w).Q$$

reduces either

- to

$$P\{z/y\} \mid x(w).Q$$

- or to

$$x(y).P \mid Q\{z/w\}$$

Linearity.

Linearity of messages and input processes.

In the π-calculus outputs (messages) and inputs are ***linear***.

- E.g. the parallel composition

$$\bar{x}z \mid x(y).P \mid x(w).Q$$

reduces either

- to

$$P\{z/y\} \mid x(w).Q$$

- or to

$$x(y).P \mid Q\{z/w\}$$

Persistence.

Persistence of messages.

Other calculi follow a different principle: *Messages are **persistent***. E.g.:

- ***Concurrent Constraint Programming*** (CCP)[Saraswat, Rinard, Panangaden '91] where

  information can only increase during computation

Persistence.

Persistence of messages.

Other calculi follow a different principle: *Messages are* ***persistent***. E.g.:

- ***Concurrent Constraint Programming*** (CCP)[Saraswat, Rinard, Panangaden '91] where

  information can only increase during computation

Persistence.

Persistence of messages and input process.

- Persistent π input processes model functions, procedures and higher-order communication.
- Persistent messages and input processes can be used to reason about protocols that can run unboundedly.

Linearity vs Persistence.

Does the persistence assumption restrict the kind of systems that can be reasoned about ? E.g.

- Could some security attacks based on linear messages be impossible to model under a persistent message assumption?
- Is π more expressive than **Persistent** π ?
- Is **Linear** CCP more expressive than CCP ?

Linearity vs Persistence.

We study the expressiveness of fragments of π capturing the above sources of persistence:

- π: *Asynch. π-calculus*, here denoted simply as π:

$$P, Q := \bar{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

- *PO*π: *Persistent-output* (messages) π:

$$P, Q := !\bar{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

- *PI*π: *Persistent-input* π:

$$P, Q := \bar{x}\langle\vec{z}\rangle \mid !x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

Linearity vs Persistence.

We study the expressiveness of fragments of π capturing the above sources of persistence:

- π: *Asynch. π-calculus*, here denoted simply as π:

$$P, Q := \bar{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

- $PO\pi$: *Persistent-output* (messages) π:

$$P, Q := !\bar{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

- $PI\pi$: *Persistent-input* π:

$$P, Q := \bar{x}\langle\vec{z}\rangle \mid !x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

Linearity vs Persistence.

We study the expressiveness of fragments of π capturing the above sources of persistence:

- π: *Asynch. π-calculus*, here denoted simply as π:

$$P, Q := \bar{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

- *PO*π: *Persistent-output* (messages) π:

$$P, Q := !\bar{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

- *PI*π: *Persistent-input* π:

$$P, Q := \bar{x}\langle\vec{z}\rangle \mid !x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

Linearity vs Persistence.

We study the expressiveness of fragments of π capturing the above sources of persistence:

- π: *Asynch. π-calculus*, here denoted simply as π:

$$P, Q := \bar{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

- *PO*π: *Persistent-output* (messages) π:

$$P, Q := !\bar{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

- *PI*π: *Persistent-input* π:

$$P, Q := \bar{x}\langle\vec{z}\rangle \mid !x(\vec{y}).P \mid P \mid Q \mid (\nu x)P \mid !P$$

Reduction semantics

COM: $\dfrac{}{\bar{x}\langle\vec{z}\rangle|x(\vec{y}).P \longrightarrow P\{\vec{z}/\vec{y}\}}$ if $|\vec{z}| = |\vec{y}|$

PAR: $\dfrac{P \longrightarrow P'}{P|Q \longrightarrow P'|Q}$ RES: $\dfrac{P \longrightarrow P'}{(\nu x)P \longrightarrow (\nu x)P'}$

STRUCT: $\dfrac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$

Some known results

In [Valencia'06] have been investigated the existence of encodings from $A\pi$ into $PIA\pi$, $POA\pi$, $PA\pi$:

- There is at least one encoding from $A\pi$ into $PIA\pi$
- There is at least one encoding from $A\pi$ into $POA\pi$
- There is no encoding from $A\pi$ into $PA\pi$

Notion of correct encoding is based on barbed bisimulation.

Notion of expressiveness

Is the model M' as expressive as the model M ?, $M' \succeq M$?

- In computability, $M' \succeq M$ *iff* there exists a f : $M \rightarrow M'$. s.t. for each M $\in M$, $L(f(M)) = L(M)$
- e.g. TM $\succeq$ PDS $\succeq$ FSA (automata theory)

The notion is well-understood in computability.

Expressiveness in process calculi

Is the calculus C' as expressive as the calculus C?, C' *succeq* C ?

- In Concurrency Theory there are several criteria to measure the expressiveness, there is no agreement on some specific criterion ,
- Intuitively, $C' \succeq C$ iff there exists an encoding $[\![\cdot]\!] : C \rightarrow C'$. s.t. for each $P \in C$, P *equivalent to* $[\![P]\!]$
- However, there are several notion of *equivalence*:
  - $!(\nu x)(\bar{x} < y > |x(y').\bar{z} < w >)$ and $!\bar{z} < w >$ are **barbed congruent**.
  - but $!(\nu x)(\bar{x} < y > |x(y').\bar{z} < w >)$ is divergent, unlike $!\bar{z} < w >$, they are not equivalent if we consider divergence. Note that ω is not visible in some evolution from $!(\nu x)(\bar{x} < y > |x(y').\bar{z} < w >)|z(w').\omega$, however ω is always visible in any evolution from $!\bar{z} < w > |z(w').\omega$.

Some basic notions

Barbed Bisimilarity, Barbed Congruence

A (weak) barbed bisimulation is a symmetric relation R satisfying the following: $(P, Q) \in R$ implies that:

1. $P \longrightarrow P'$ then $\exists Q' : Q \longrightarrow^* Q' \wedge (P', Q') \in R$.
2. $P \downarrow_{\bar{x}}$ then $Q \Downarrow_{\bar{x}}$.

P and Q are barbed bisimilar, $P \sim Q$, *iff* $(P, Q) \in R$ for some barbed bisimulation R. Furthermore, P and Q, $P \approx Q$ are barbed congruence *iff* for every process context $C[\cdot]$:, $C[P] \sim C[Q]$.

Divergence

Divergence

A process P is divergent *iff* $P\ (\rightarrow)^{\omega}$, i.e. there exists an infinite sequence $P = P_0 \rightarrow P_1 \rightarrow \ldots$

Examples

- $!x(y) \mid !x < z >$
- $!x(y).\bar{x} < y > \mid \bar{x} < z >$

Testing semantics

Maximal Computations

Given $P \in P$ and $o \in O$, a maximal computation from $P \mid o$ is either an infinite sequence of the form:

$$P \mid o = E_0 \rightarrow E_1 \rightarrow E_0 \rightarrow \ldots$$

or a finite sequence of the form:

$$P \mid o = E_0 \rightarrow E_1 \rightarrow E_0 \rightarrow \ldots E_n \not\rightarrow$$

Testing semantics

May, must and fair relations

Given $P \in \mathcal{P}$ and $o \in \mathcal{O}$, define:

- P may o *iff* there is a maximal computation such that $E_i \xrightarrow{\omega}$ for some $i \geq 0$
  e.g. $\bar{a} < x > may\ b(z).\omega$, $\bar{a} < x > may\ a(y)|a(y).\omega$
- P fair o *iff* for every maximal computation $\forall i \geq 0\ \exists E_i'$ such that $E_i \xrightarrow{*} E_i'$ and $E_i' \xrightarrow{\omega}$
  e.g. $\bar{a} < x > fair\ \ a(y)|a(y).\omega$, $!\bar{a} < x > |!a(y)\ fair\ a(y).\omega$
- P must o *iff* for every maximal computation there exists $i \geq 0$ such that $E_i \xrightarrow{\omega}$
  e.g. $!\bar{a} < x > |!a(y) must\ \ a(y).\omega$,
  $!a < x > |a(y) must\ a(y).\omega$

Testing semantics

May, must and fair preserving

- An encoding $[\![\cdot]\!]$ is sat-preserving iff $\forall P \in C$, $\forall o \in O$, $[\![P]\!]$ sat o iff $[\![P]\!]$ sat $[\![o]\!]$
- e.g. $[\![\cdot]\!]$ is must-preserving iff $\forall P \in C$, $\forall o \in O$, $[\![P]\!]$ must o iff $[\![P]\!]$ must $[\![o]\!]$

An expressiveness study

- We have proved that both $PIA\pi$ and $POA\pi$ preserve fair and may-testing from $A\pi$.
- We have proved the first separability results between $A\pi$ and its semi-persistent subsets: there is no encoding from $A\pi$ into $PIA\pi$ or $POA\pi$ must-preserving

Encoding $A\pi$ in $PIA\pi$

[Valencia 06] The encoding $[\![\cdot]\!] : A\pi \longrightarrow PIA\pi$ is a homomorphism for all operators except the input-prefix:

$$[\![x(\vec{y}).P]\!] = (\nu t\ f)(\bar{t} \mid !x(\vec{y}).(\nu \ell) \mid \begin{array}{l} \bar{\ell} \\ !t.!\ell.([\![P]\!] \mid !\bar{f}) \mid \\ !f.!\ell.\bar{x}\langle\vec{y}\rangle)) \end{array}$$

- The idea is a suitable combination of locking and forwarding mechanisms: If the $[\![x(y).P]\!]$ has already received a message then it forwards the current message.

Encoding $A\pi$ in $PIA\pi$ is may and fair preserving

[Valencia 06] The encoding $[\![\cdot]\!] : A\pi \rightarrow PIA\pi$ is a homomorphism for all operators except the input-prefix:

$$[\![x(\vec{y}).P]\!] = (\nu t\ f)(\bar{t} \mid !x(\vec{y}).(\nu \ell) \mid \bar{\ell} \mid !t.!\ell.([\![P]\!] \mid !\bar{f}) \mid !f.!\ell.\bar{x}\langle\vec{y}\rangle))$$

To prove that this encoding is may and fair preserving is important to consider:

- $[\![\cdot]\!]$ preserves barbed bisimilarity.
- ω is considered as a barb.
- Homomorphism w.r.t $\mid$ $[\![P|o]\!] = [\![P]\!]|[\![o]\!]$.

Encoding $A\pi$ in $PIA\pi$ is may and fair preserving

[Valencia 06] The encoding $[\![\cdot]\!] : A\pi \rightarrow PIA\pi$ is a homomorphism for all operators except the input-prefix:

$$[\![x(\vec{y}).P]\!] = (\nu t\ f)(\bar{t} \mid !x(\vec{y}).(\nu \ell) \mid \bar{\ell} \mid !t.!\ell.([\![P]\!] \mid !\bar{f}) \mid !f.!\ell.\bar{x}\langle\vec{y}\rangle))$$

To prove that this encoding is may and fair preserving is important to consider:

- $[\![\cdot]\!]$ preserves barbed bisimilarity.
- ω is considered as a barb.
- Homomorphism w.r.t $\mid$ $[\![P|o]\!] = [\![P]\!]|[\![o]\!]$.

Encoding $A\pi$ in $PIA\pi$ is may and fair preserving

[Valencia 06] The encoding $[\![\cdot]\!] : A\pi \rightarrow PIA\pi$ is a homomorphism for all operators except the input-prefix:

$$[\![x(\vec{y}).P]\!] = (\nu t\ f)(\bar{t} \mid !x(\vec{y}).(\nu \ell) \mid \bar{\ell} \mid !t.!\ell.([\![P]\!] \mid !\bar{f}) \mid !f.!\ell.\bar{x}\langle\vec{y}\rangle))$$

To prove that this encoding is may and fair preserving is important to consider:

- $[\![\cdot]\!]$ preserves barbed bisimilarity.
- ω is considered as a barb.
- Homomorphism w.r.t $\mid$ $[\![P|o]\!] = [\![P]\!]|[\![o]\!]$.

Encoding $A\pi$ in $PIA\pi$ is may and fair preserving

[Valencia 06] The encoding $[\![\cdot]\!] : A\pi \rightarrow PIA\pi$ is a homomorphism for all operators except the input-prefix:

$$[\![x(\vec{y}).P]\!] = (\nu t\ f)(\bar{t} \mid !x(\vec{y}).(\nu \ell) \mid \bar{\ell} \mid !t.!\ell.([\![P]\!] \mid !\bar{f}) \mid !f.!\ell.\bar{x}\langle\vec{y}\rangle))$$

To prove that this encoding is may and fair preserving is important to consider:

- $[\![\cdot]\!]$ preserves barbed bisimilarity.
- ω is considered as a barb.
- Homomorphism w.r.t | $[\![P|o]\!] = [\![P]\!]|[\![o]\!]$.

Encoding $A\pi$ in $PIA\pi$ is may and fair preserving

To prove that this encoding is may and fair preserving is important to consider:

- $[\![\cdot]\!]$ preserves barbed bisimilarity.
- ω is considered as a barb.
- Homomorphism w.r.t $|$ $[\![P|o]\!] = [\![P]\!]|[\![o]\!]$.

Theorem

Every (asynch) π process P is (weak) fair and may equivalent to $[\![P]\!]$
Idea: As $P|o \sim [\![P|o]\!] = [\![P]\!]|[\![o]\!]$. $P \mid o \Downarrow_\omega$ iff $[\![P]\!] \mid [\![o]\!] \Downarrow_\omega$, therefore $[\![\cdot]\!]$ is may-preserving. $[\![\cdot]\!]$ is fair -preserving as $P|o \rightarrow E_1 \ldots E_n \ldots$ then $E_i \sim R$ for some R s.t. $[\![P]\!]|[\![o]\!] \rightarrow^* R$.

Impossibility to encode $A\pi$ into $PIA\pi$ and $POA\pi$

To prove that there is no must-preserving encoding , we divide the proof into two parts:

- $\forall x, y, x', z'$, either $[\![!x(y).0|\bar{x} < z > |\bar{x'} < z' >]\!]$ or $[\![x(y).0|!(\bar{x} < z > |\bar{x'} < z' >)]\!]$ is divergent.
- Consider $P = \bar{x} < z > |\bar{x'} < z' >$, $o = !x(y).x'(y').\omega.0$ $(x \neq x')$, $o' = x(y).x'(y').\omega.0$, $P' = !(\bar{x} < z > |\bar{x'} < z' >)$. there is no encoding which preserves P *must* o or P' *must* o'

Impossibility to encode $A\pi$ into $PIA\pi$

Theorem

$\forall x, y, x', z'$, *either* $[\![!x(y).0|\bar{x} < z > |\bar{x'} < z' >]\!]$ *or* $[\![x(y).0|!(\bar{x} < z > |\bar{x'} < z' >)]\!]$ *is divergent.*
Idea: (By contradiction) we can prove that $[\![x(y)]\!]$ *and* $[\![\bar{x} < z > |\bar{x'} < z' >]\!]$ *can interact otherwise* $[\![!x(y).\omega.0]\!]$ *must* $[\![\bar{x} < z > |\bar{x'} < z' >]\!]$ *, therefore as either input or output is replicated then either* $[\![!x(y).0|\bar{x} < z > |\bar{x'} < z' >]\!]$ *or* $[\![x(y).0|!(\bar{x} < z > |\bar{x'} < z' >)]\!]$ *is divergent*

Impossibility to encode $A\pi$ into $PIA\pi$

Theorem

There is no must-preserving encoding from $A\pi$ into $PIA\pi$ or $POA\pi$.

Idea: As $[\![!x(y).0|\bar{x} < z > |\bar{x'} < z' >]\!]$ *or* $[\![x(y).0|!(\bar{x} < z > |\bar{x'} < z' >)]\!]$ *is divergent, therefore either* $[\![!x(y).x'(y').\omega.0$ *must* $[\![\bar{x} < z > |\bar{x'} < z' >]\!]$ *or* $[\![x(y).x'(y').\omega.0]\!]$ *must* $[\![!(\bar{x} < z > |\bar{x'} < z' >)]\!]$, .

Future work

1. To study the relation between different equivalences and testing-equivalence.
2. To prove similar results on polyadic π- calculus and monadic π- calculus.
3. To study other notions of must by using scheduler to rule out those computations which don't follow the expected behaviour.