
Processing Programs: the linear case

Luca Paolini¹ & Mauro Piccolo^{1,2}

¹Dipartimento di Informatica. Università di Torino (Italia).

²Preuves, Programmes et Systèmes. Paris VII (France)

March 2008

Computational processes

▷ Processes
Outline

*λ*PCF

*lin*Solos

Main Results

Conclusions

In [AbelsonSussman85,Ch1], Harold Abelson and Sussman, Gerald Jay and Julie Sussman state,

“We are about to study the idea of a computational process. Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called data. The evolution of a process is directed by a pattern of rules called a program. People create programs to direct processes. In effect, we conjure the spirits of the computer with our spells.”

Computational processes

▷ Processes
Outline

*S*lPCF

*lin*Solos

Main Results

Conclusions

Our goal is **to formalize** such trivial idea !!!

In literature,

many works touch lightly such idea with more specific goals, as

1. to prove the expressiveness of a mobile calculus, by showing how to represent the λ -calculus evaluation strategies
2. to represent strategies of game semantics for paradigmatic programming languages.

Our goal is **to formalize** such trivial idea !!!

In literature,

many works touch lightly such idea with more specific goals, as

1. to prove the expressiveness of a mobile calculus, by showing how to represent the λ -calculus evaluation strategies
2. to represent strategies of game semantics for paradigmatic programming languages.

Our goal is **to formalize** such trivial idea !!!

In literature,

many works touch lightly such idea with more specific goals, as

1. to prove the expressiveness of a mobile calculus, by showing how to represent the λ -calculus evaluation strategies
2. to represent strategies of game semantics for paradigmatic programming languages.

Outline of the Talk

Processes
▷ Outline

*ℓ*PCF

*lin*Solos

Main Results

Conclusions

- *ℓ*PCF: a simple linear programming language
- *lin*Solos:
a typed process calculus based on the calculus of solos
- Interpretation of *ℓ*PCF on *lin*Solos
- Correctness w.r.t the operational semantic of *ℓ*PCF.
- Parallel evaluation of our *ℓ*PCF-processes.

Outline of the Talk

Processes
▷ Outline

*λ*PCF

*lin*Solos

Main Results

Conclusions

- *λ*PCF: a simple linear programming language
- *lin*Solos:
a typed process calculus based on the calculus of solos
- Interpretation of *λ*PCF on *lin*Solos
- Correctness w.r.t the operational semantic of *λ*PCF.
- Parallel evaluation of our *λ*PCF-processes.

Outline of the Talk

Processes
▷ Outline

*ℓ*PCF

*lin*Solos

Main Results

Conclusions

- *ℓ*PCF: a simple linear programming language
- *lin*Solos:
a typed process calculus based on the calculus of solos
- Interpretation of *ℓ*PCF on *lin*Solos
- Correctness w.r.t the operational semantic of *ℓ*PCF.
- Parallel evaluation of our *ℓ*PCF-processes.

Outline of the Talk

Processes
▷ Outline

slPCF

linSolos

Main Results

Conclusions

- *slPCF*: a simple linear programming language
- *linSolos*:
a typed process calculus based on the calculus of solos
- Interpretation of *slPCF* on *linSolos*
- Correctness w.r.t the operational semantic of *slPCF*.
- Parallel evaluation of our *slPCF*-processes.

Outline of the Talk

Processes
▷ Outline

*ℓ*PCF

*lin*Solos

Main Results

Conclusions

- *ℓ*PCF: a simple linear programming language
- *lin*Solos:
a typed process calculus based on the calculus of solos
- Interpretation of *ℓ*PCF on *lin*Solos
- Correctness w.r.t the operational semantic of *ℓ*PCF.
- Parallel evaluation of our *ℓ*PCF-processes.

Processes

Outline

▷ $\mathcal{S}\ell\text{PCF}$

Linearity

A Semantically
Linear Programming
Language

*lin*Solos

Main Results

Conclusions

Introducing $\mathcal{S}\ell\text{PCF}$

Linearity

Processes

Outline

*λ*PCF

▷ Linearity

A Semantically
Linear Programming
Language

*lin*Solos

Main Results

Conclusions

*λ*PCF is a simple programming language, **fully abstract** modeled by **linear function between coherence spaces**.

Linearity

Processes

Outline

λ PCF

▷ Linearity

A Semantically
Linear Programming
Language

*lin*Solos

Main Results

Conclusions

λ PCF is a simple programming language, **fully abstract** modeled by **linear function between coherence spaces**.

λ PCF is not syntactically linear, in fact its programs can contain the same variable more than once.

Linearity can be considered in many respects.

*λ*PCF is a simple programming language, **fully abstract** modeled by **linear function between coherence spaces**.

*λ*PCF is not syntactically linear, in fact its programs can contain the same variable more than once.

Linearity can be considered in many respects.

- Syntactical

λ PCF is a simple programming language, **fully abstract** modeled by **linear function between coherence spaces**.

λ PCF is not syntactically linear, in fact its programs can contain the same variable more than once.

Linearity can be considered in many respects.

- Syntactical
- Denotational

*ℓ*PCF is a simple programming language, **fully abstract** modeled by **linear function between coherence spaces**.

*ℓ*PCF is not syntactically linear, in fact its programs can contain the same variable more than once.

Linearity can be considered in many respects.

- Syntactical
- Denotational
- Operational

A Semantically Linear Programming Language

Types: $\sigma, \tau ::= \iota \mid (\sigma \multimap \tau)$

0^ι	
$\text{succ}^{\iota \multimap \iota}$	
$\text{pred}^{\iota \multimap \iota}$	

$$\frac{}{\underline{0} \Downarrow \underline{0}} (0) \quad \frac{M \Downarrow \underline{n}}{\text{succ } M \Downarrow \text{succ } \underline{n}} (s) \quad \frac{M \Downarrow \text{succ } \underline{n}}{\text{pred } M \Downarrow \underline{n}} (p_n)$$

A Semantically Linear Programming Language

Types: $\sigma, \tau ::= \iota \mid (\sigma \multimap \tau)$

0^ι	
$\text{succ}^{\iota \multimap \iota}$	
$\text{pred}^{\iota \multimap \iota}$	
x^ι	ground variable (GFV(M))
$x^{\sigma \multimap \tau}$	higher order variable (HFV(M))

A Semantically Linear Programming Language

Types: $\sigma, \tau ::= \iota \mid (\sigma \multimap \tau)$

0^ι	
$\text{succ}^{\iota \multimap \iota}$	
$\text{pred}^{\iota \multimap \iota}$	
x^ι $x^{\sigma \multimap \tau}$	ground variable ($\text{GFV}(M)$) higher order variable ($\text{HFV}(M)$)
$(\lambda x^\iota . M^\tau)^{\iota \multimap \tau}$ $(\lambda x^{\sigma \multimap \tau} . M)$	x can occur zero or many time in M x can occur exactly once

$$\frac{N \Downarrow \underline{m} \quad M[\underline{m}/x]P_1 \cdots P_i \Downarrow \underline{n}}{(\lambda x^\iota . M)NP_1 \cdots P_i \Downarrow \underline{n}} (\lambda^\iota) \quad \frac{M[N/x]P_1 \cdots P_i \Downarrow \underline{n}}{(\lambda x^{\sigma \multimap \tau} . M)NP_1 \cdots P_i \Downarrow \underline{n}} (\lambda^{\multimap})$$

A Semantically Linear Programming Language

Types: $\sigma, \tau ::= \iota \mid (\sigma \multimap \tau)$

0^ι	
$\text{succ}^{\iota \multimap \iota}$	
$\text{pred}^{\iota \multimap \iota}$	
x^ι $x^{\sigma \multimap \tau}$	ground variable ($\text{GFV}(M)$) higher order variable ($\text{HFV}(M)$)
$(\lambda x^\iota. M^\tau)^{\iota \multimap \sigma}$ $(\lambda x^{\sigma \multimap \tau}. M)$	x can occur zero or many time in M x can occur exactly once
$(\text{elif } M^\iota \text{ L}^\iota \text{ R}^\iota)^\iota$	if $\text{HFV}(L^\iota) = \text{HFV}(R^\iota)$, $\text{HFV}(M^\iota) \cap \text{HFV}(R^\iota) = \emptyset$

$$\frac{M \Downarrow \underline{0} \quad L \Downarrow \underline{m}}{\text{elif } M \text{ L } R \Downarrow \underline{m}} \text{ (if}_l\text{)} \qquad \frac{M \Downarrow \text{succ}(\underline{n}) \quad R \Downarrow \underline{m}}{\text{elif } M \text{ L } R \Downarrow \underline{m}} \text{ (if}_r\text{)}$$

A Semantically Linear Programming Language

Types: $\sigma, \tau ::= \iota \mid (\sigma \multimap \tau)$

0^ι	
$\text{succ}^{\iota \multimap \iota}$	
$\text{pred}^{\iota \multimap \iota}$	
x^ι $x^{\sigma \multimap \tau}$	ground variable ($\text{GFV}(M)$) higher order variable ($\text{HFV}(M)$)
$(\lambda x^\iota . M^\tau)^{\iota \multimap \tau}$ $(\lambda x^{\sigma \multimap \tau} . M)$	x can occur zero or many time in M x can occur exactly once
$(\ell \text{if } M^\iota \text{ L}^\iota \text{ R}^\iota)^\iota$	if $\text{HFV}(\text{L}^\iota) = \text{HFV}(\text{R}^\iota)$, $\text{HFV}(M^\iota) \cap \text{HFV}(\text{R}^\iota) = \emptyset$
F^σ $(\mu F^\sigma . M^\sigma)^\sigma$	recursion (stable) variable ($\text{SFV}(M)$) if $F^\sigma \in \text{SFV}(M^\sigma)$ and $\text{HFV}(M^\sigma) = \emptyset$

$$\frac{M[\mu F . M / F] P_1 \cdots P_i \Downarrow \underline{n}}{(\mu F . M) P_1 \cdots P_i \Downarrow \underline{n}} (\mu)$$

A Semantically Linear Programming Language

Types: $\sigma, \tau ::= \iota \mid (\sigma \multimap \tau)$

0^ι	
$\text{succ}^{\iota \multimap \iota}$	
$\text{pred}^{\iota \multimap \iota}$	
x^ι $x^{\sigma \multimap \tau}$	ground variable ($\text{GFV}(\mathbb{M})$) higher order variable ($\text{HFV}(\mathbb{M})$)
$(\lambda x^\iota. M^\tau)^{\iota \multimap \tau}$ $(\lambda x^{\sigma \multimap \tau}. M)$	x can occur zero or many time in \mathbb{M} x can occur exactly once
$(\text{lif } M^\iota \text{ L}^\iota \text{ R}^\iota)^\iota$	if $\text{HFV}(\text{L}^\iota) = \text{HFV}(\text{R}^\iota)$, $\text{HFV}(M^\iota) \cap \text{HFV}(\text{R}^\iota) = \emptyset$
F^σ $(\mu F^\sigma. M^\sigma)^\sigma$	recursion (stable) variable ($\text{SFV}(\mathbb{M})$) if $F^\sigma \in \text{SFV}(\mathbb{M}^\sigma)$ and $\text{HFV}(\mathbb{M}^\sigma) = \emptyset$
which?	

$$\frac{\mathbb{M}(\lambda x^\iota. \text{lif}(x \doteq k) \underline{k} \Omega^\iota) \Downarrow \underline{n}}{\text{which? } \mathbb{M}^{(\iota \multimap \iota) \multimap \iota} \Downarrow [\underline{n}, \underline{k}]} \quad (\text{w}^{\underline{k}})$$

Processes

Outline

*sl*PCF

▷ *lin*Solos

Causality

*lin*Solos-calculus

Reduction

Linearity

Typing rules

Main Results

Conclusions

Introducing *lin*Solos

Causality in process calculi

Processes
Outline

slPCF

linSolos

▷ Causality

linSolos-calculus

Reduction

Linearity

Typing rules

Main Results

Conclusions

- There are two possible ways to express causality in process calculi
 - **Explicit**, by using prefixing
 - ▷ $\alpha.P$
 - **Implicit**, by using the restriction construct
 - ▷ $(\nu v)(\bar{u}\langle v \rangle \parallel v\langle w \rangle)$

Causality in process calculi

Processes
Outline

slPCF

linSolos

▷ Causality

linSolos-calculus

Reduction

Linearity

Typing rules

Main Results

Conclusions

- There are two possible ways to express causality in process calculi
 - **Explicit**, by using prefixing
 - ▷ $\alpha.P$
 - **Implicit**, by using the restriction construct
 - ▷ $(\nu v)(\bar{u}\langle v \rangle \parallel v\langle w \rangle)$

Syntax:

Processes $P ::= \quad | P \parallel Q \quad | (\nu v)P \quad | \mathbf{0}$

Structural congruence: α -rule, abelian monoidal laws for $|$ and

$(\nu v)\mathbf{0} \equiv \mathbf{0}$ $(\nu v)(P \parallel Q) \equiv P \parallel (\nu v)Q$ if $v \notin \text{fn}(P)$	$(\nu v)(\nu w)P \equiv (\nu w)(\nu v)P$
--	--

Reduction rules:

$$\frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q}$$

$$\frac{P \longrightarrow P'}{(\nu v)P \longrightarrow (\nu v)P'}$$

$$\frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$$

Syntax:

<i>Solos</i>	$\sigma ::= q\langle\tilde{p}, a\rangle \bar{q}\langle\tilde{p}, a\rangle$
<i>Processes</i>	$P ::= \sigma \quad \quad P\ Q \quad \quad (\nu v)P \quad \quad \mathbf{0}$

Structural congruence: α -rule, abelian monoidal laws for $|$ and

$(\nu v)\mathbf{0} \equiv \mathbf{0}$ $(\nu v)(P\ Q) \equiv P\ (\nu v)Q$ if $v \notin \text{fn}(P)$	$(\nu v)(\nu w)P \equiv (\nu w)(\nu v)P$
--	--

Reduction rules:

$$\frac{|\tilde{v}_1| = |\tilde{v}_2| \quad \theta \text{ agrees with } \{\tilde{v}_1 = \tilde{v}_2\} \quad \text{ran}(\theta) \cap \tilde{w} = \emptyset \quad \text{Dom}(\theta) = \tilde{w}}{(\nu \tilde{w})(u\langle\tilde{v}_1\rangle \| \bar{u}\langle\tilde{v}_2\rangle \| R) \longrightarrow R\theta}$$

$$\frac{P \longrightarrow P'}{P\|Q \longrightarrow P'\|Q}$$

$$\frac{P \longrightarrow P'}{(\nu v)P \longrightarrow (\nu v)P'}$$

$$\frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$$

Syntax:

<i>Expressions:</i>	$E ::= n \mid x \mid \text{succ}(E) \mid \text{pred}(E)$
<i>Solos</i>	$\sigma ::= q\langle \tilde{p}, a \rangle \mid \bar{q}\langle \tilde{p}, a \rangle$
<i>Processes</i>	$P ::= \sigma \quad \mid P \parallel Q \mid (\nu v)P \quad \mid \mathbf{0}$

Structural congruence: α -rule, abelian monoidal laws for \mid and

$(\nu v)\mathbf{0} \equiv \mathbf{0}$	$(\nu v)(\nu w)P \equiv (\nu w)(\nu v)P$
$(\nu v)(P \parallel Q) \equiv P \parallel (\nu v)Q$ if $v \notin \text{fn}(P)$	

Reduction rules:

$\overline{n \longrightarrow n}$	$\frac{E \longrightarrow n}{\text{succ}(E) \longrightarrow n + 1}$	$\frac{E \longrightarrow n}{\text{pred}(E) \longrightarrow n - 1}$
$\frac{ \tilde{v}_1 = \tilde{v}_2 \quad \theta \text{ agrees with } \{\tilde{v}_1 = \tilde{v}_2\} \quad \text{ran}(\theta) \cap \tilde{w} = \emptyset \quad \text{Dom}(\theta) = \tilde{w}}{(\nu \tilde{w})(u\langle \tilde{v}_1 \rangle \parallel \bar{u}\langle \tilde{v}_2 \rangle \parallel R) \longrightarrow R\theta}$		

$$\frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q}$$

$$\frac{P \longrightarrow P'}{(\nu v)P \longrightarrow (\nu v)P'}$$

$$\frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$$

Syntax:

Expressions: $E ::= n \mid x \mid \text{succ}(E) \mid \text{pred}(E)$

$\sigma ::= q\langle\tilde{p}, a\rangle \mid \bar{q}\langle\tilde{p}, a\rangle$

Processes $P ::= \sigma \mid a(x)P \mid \bar{a}\langle E \rangle \mid P\|Q \mid (\nu v)P \mid [P \oplus E Q] \mid F^q \mid \text{rec}F.P^\dagger \mid \mathbf{0}$

† We impose that P contains exactly one free question-name

Structural congruence: α -rule, abelian monoidal laws for \mid and

$(\nu v)\mathbf{0} \equiv \mathbf{0}$

$(\nu v)(P\|Q) \equiv P\|(\nu v)Q$ if $v \notin \text{fn}(P)$

$a(x)(P\|Q) \equiv P\|(a(x)Q)$ if $x \notin \text{FV}(P)$

$\text{rec}F.P \equiv P\{\text{rec}F.P/F\}$

$(\nu v)(\nu w)P \equiv (\nu w)(\nu v)P$

$a(x)b(y)P \equiv b(y)a(x)P$

$a(x)(\nu v)P \equiv (\nu v)(a(x)P)$ if $v \neq a$

Reduction rules:

$\frac{}{n \longrightarrow n} \quad \frac{E \longrightarrow n}{\text{succ}(E) \longrightarrow n + 1} \quad \frac{E \longrightarrow n}{\text{pred}(E) \longrightarrow n - 1}$

$\frac{|\tilde{v}_1| = |\tilde{v}_2| \quad \theta \text{ agrees with } \{\tilde{v}_1 = \tilde{v}_2\} \quad \text{ran}(\theta) \cap \tilde{w} = \emptyset \quad \text{Dom}(\theta) = \tilde{w}}{(\nu \tilde{w})(u\langle\tilde{v}_1\rangle \parallel \bar{u}\langle\tilde{v}_2\rangle \parallel R) \longrightarrow R\theta}$

$\frac{E \longrightarrow n}{a(x)P\bar{a}\langle E \rangle \longrightarrow P[n/x]}$

$\frac{E \longrightarrow 0}{[P \oplus E Q] \longrightarrow P}$

$\frac{E \longrightarrow n \quad n \neq 0}{[P \oplus E Q] \longrightarrow Q}$

$\frac{P \longrightarrow P'}{P\|Q \longrightarrow P'\|Q}$

$\frac{P \longrightarrow P'}{a(\tilde{x})P \longrightarrow a(\tilde{x})P'}$

$\frac{P \longrightarrow P'}{(\nu v)P \longrightarrow (\nu v)P'}$

$\frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$

Example of Solos-reduction

Processes

Outline

*sl*PCF

*lin*Solos

Causality

*lin*Solos-calculus

▷ Reduction

Linearity

Typing rules

Main Results

Conclusions

Reduction fire requires unification of names.

$$(\nu p_0, p_1, a)(q\langle p, p, a \rangle \parallel \bar{q}\langle p_0, p_1, b \rangle \parallel R) \longrightarrow R[p/p_0, p/p_1, b/a]$$

$$(\nu p_0, p, b)(q\langle p, p, a \rangle \parallel \bar{q}\langle p_0, p_1, b \rangle \parallel R) \longrightarrow R[p_1/p_0, p_1/p, a/b]$$

$$(\nu p)(q\langle p, p, a \rangle \parallel \bar{q}\langle p_0, p_1, b \rangle \parallel R) \not\longrightarrow$$

Linearity

Processes
Outline

*sl*PCF

*lin*Solos

Causality

*lin*Solos-calculus

Reduction

▷ Linearity

Typing rules

Main Results

Conclusions

- The notion of syntactical linearity used, literally, in a process language become meaningless.

Linearity

Processes
Outline

*sl*PCF

*lin*Solos

Causality

*lin*Solos-calculus

Reduction

▷ Linearity

Typing rules

Main Results

Conclusions

- The notion of syntactical linearity used, literally, in a process language become meaningless.
- A clever adaptation of such a notion has been has done by Kobayashi, Yoshida, Honda and Berger.

Linearity

Processes
Outline

slPCF

linSolos

Causality

linSolos-calculus

Reduction

▷ Linearity

Typing rules

Main Results

Conclusions

- The notion of syntactical linearity used, literally, in a process language become meaningless.
- A clever adaptation of such a notion has been done by Kobayashi, Yoshida, Honda and Berger.
- There is exactly one input and exactly one output on a name, for each possible evolution of the process
 - + output – input \updownarrow consumed.

Processes
Outline

*sl*PCF

*lin*Solos

Causality

*lin*Solos-calculus

Reduction

▷ Linearity

Typing rules

Main Results

Conclusions

- Linear channel types: sorting augmented with a modality

Linearity

Processes
Outline

*sl*PCF

*lin*SoLos

Causality

*lin*SoLos-calculus

Reduction

▷ Linearity

Typing rules

Main Results

Conclusions

- Linear channel types: sorting augmented with a modality
- A match operator \odot defined as $S^+ \odot S^- = S^- \odot S^+ = S^\updownarrow$

Linearity

Processes
Outline

*sl*PCF

*lin*SoLos

Causality

*lin*SoLos-calculus

Reduction

▷ Linearity

Typing rules

Main Results

Conclusions

□ Linear channel types: sorting augmented with a modality

□ A match operator \odot defined as $S^+ \odot S^- = S^- \odot S^+ = S^\updownarrow$

□ The straightforward match operator \odot for environment

$$A \odot B = (A \setminus B) \cup (B \setminus A) \cup \{v : \tau_1 \odot \tau_2 \mid (v : \tau_1) \in A, (v : \tau_2) \in B\}$$

□ Linear channel types: sorting augmented with a modality

□ A match operator \odot defined as $S^+ \odot S^- = S^- \odot S^+ = S^\downarrow$

□ The straightforward match operator \odot for environment

$$A \odot B = (A \setminus B) \cup (B \setminus A) \cup \{v : \tau_1 \odot \tau_2 \mid (v : \tau_1) \in A, (v : \tau_2) \in B\}$$

□ Process variable environments for **Recursion**

Typing rules

Typing judgment in a natural deduction style: $\Gamma \vdash P \triangleright A$

$$\frac{}{\Gamma \vdash \mathbf{0} \triangleright _} \text{ (z)} \quad \frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright B \quad A \asymp B}{\Gamma \vdash P \parallel Q \triangleright A \odot B} \text{ (par)} \quad \frac{\Gamma \vdash P \triangleright A^+ \quad \Gamma \vdash Q \triangleright A^+}{\Gamma \vdash [P \oplus Q] \triangleright A^+} \text{ (sum)}$$

Typing rules

Typing judgment in a natural deduction style: $\Gamma \vdash P \triangleright A$

$$\begin{array}{c} \frac{}{\Gamma \vdash \mathbf{0} \triangleright _} \text{(z)} \quad \frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright B \quad A \asymp B}{\Gamma \vdash P \parallel Q \triangleright A \odot B} \text{(par)} \quad \frac{\Gamma \vdash P \triangleright A^+ \quad \Gamma \vdash Q \triangleright A^+}{\Gamma \vdash [P \oplus Q] \triangleright A^+} \text{(sum)} \\ \\ \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash q\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^-, \tilde{p} : \tilde{\phi}^-, a : \iota^-} \text{(in)} \quad \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash \bar{q}\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^+, \tilde{p} : \tilde{\phi}^+, a : \iota^+} \text{(out)} \end{array}$$

Typing rules

Typing judgment in a natural deduction style: $\Gamma \vdash P \triangleright A$

$$\begin{array}{c}
 \frac{}{\Gamma \vdash \mathbf{0} \triangleright _} \text{ (z)} \qquad \frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright B \quad A \asymp B}{\Gamma \vdash P \parallel Q \triangleright A \odot B} \text{ (par)} \qquad \frac{\Gamma \vdash P \triangleright A^+ \quad \Gamma \vdash Q \triangleright A^+}{\Gamma \vdash [P \oplus Q] \triangleright A^+} \text{ (sum)} \\
 \\
 \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash q\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^-, \tilde{p} : \tilde{\phi}^-, a : \iota^-} \text{ (in)} \qquad \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash \bar{q}\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^+, \tilde{p} : \tilde{\phi}^+, a : \iota^+} \text{ (out)} \\
 \\
 \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash a(x) P \triangleright a : \iota^-, A} \text{ (d-in)} \qquad \frac{}{\Gamma \vdash \bar{a}\langle E \rangle \triangleright a : \iota^+} \text{ (a-out)} \qquad \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash P \triangleright A, v : S \updownarrow} \text{ (w)}
 \end{array}$$

Typing rules

Typing judgment in a natural deduction style: $\Gamma \vdash P \triangleright A$

$$\begin{array}{c}
 \frac{}{\Gamma \vdash \mathbf{0} \triangleright _} \text{(z)} \qquad \frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright B \quad A \asymp B}{\Gamma \vdash P \parallel Q \triangleright A \odot B} \text{(par)} \qquad \frac{\Gamma \vdash P \triangleright A^+ \quad \Gamma \vdash Q \triangleright A^+}{\Gamma \vdash [P \oplus Q] \triangleright A^+} \text{(sum)} \\
 \\
 \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash q\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^-, \tilde{p} : \tilde{\phi}^-, a : \iota^-} \text{(in)} \qquad \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash \bar{q}\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^+, \tilde{p} : \tilde{\phi}^+, a : \iota^+} \text{(out)} \\
 \\
 \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash a(x) P \triangleright a : \iota^-, A} \text{(d-in)} \qquad \frac{}{\Gamma \vdash \bar{a}\langle E \rangle \triangleright a : \iota^+} \text{(a-out)} \qquad \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash P \triangleright A, v : S^\updownarrow} \text{(w)} \\
 \\
 \frac{}{\Gamma, F : \phi \vdash F^\natural \triangleright q : \phi^-} \text{(p-v)} \qquad \frac{\Gamma, F : \phi \vdash P \triangleright q : \phi^-}{\Gamma \vdash \text{rec} F.P \triangleright q : \phi^-} \text{(rec)} \qquad \frac{\Gamma \vdash P \triangleright A, v : S^\updownarrow}{\Gamma \vdash (\nu v) P \triangleright A} \text{(Res)}
 \end{array}$$

Typing rules

Typing judgment in a natural deduction style: $\Gamma \vdash P \triangleright A$

$$\begin{array}{c}
 \frac{}{\Gamma \vdash \mathbf{0} \triangleright _} \text{ (z)} \qquad \frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright B \quad A \asymp B}{\Gamma \vdash P \parallel Q \triangleright A \odot B} \text{ (par)} \qquad \frac{\Gamma \vdash P \triangleright A^+ \quad \Gamma \vdash Q \triangleright A^+}{\Gamma \vdash [P \oplus Q] \triangleright A^+} \text{ (sum)} \\
 \\
 \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash q\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^{-}, \tilde{p} : \tilde{\phi}^{-}, a : \iota^{-}} \text{ (in)} \qquad \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash \bar{q}\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^{+}, \tilde{p} : \tilde{\phi}^{+}, a : \iota^{+}} \text{ (out)} \\
 \\
 \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash a(x) P \triangleright a : \iota^{-}, A} \text{ (d-in)} \qquad \frac{}{\Gamma \vdash \bar{a}\langle E \rangle \triangleright a : \iota^{+}} \text{ (a-out)} \qquad \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash P \triangleright A, v : S^{\uparrow\downarrow}} \text{ (w)} \\
 \\
 \frac{}{\Gamma, F : \phi \vdash F^{\mathfrak{q}} \triangleright q : \phi^{-}} \text{ (p-v)} \qquad \frac{\Gamma, F : \phi \vdash P \triangleright q : \phi^{-}}{\Gamma \vdash \text{rec} F.P \triangleright q : \phi^{-}} \text{ (rec)} \qquad \frac{\Gamma \vdash P \triangleright A, v : S^{\uparrow\downarrow}}{\Gamma \vdash (\nu v) P \triangleright A} \text{ (Res)}
 \end{array}$$

□ **linearity**

Typing rules

Typing judgment in a natural deduction style: $\Gamma \vdash P \triangleright A$

$$\begin{array}{c}
 \frac{}{\Gamma \vdash \mathbf{0} \triangleright _} \text{ (z)} \qquad \frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright B \quad A \asymp B}{\Gamma \vdash P \parallel Q \triangleright A \odot B} \text{ (par)} \qquad \frac{\Gamma \vdash P \triangleright A^+ \quad \Gamma \vdash Q \triangleright A^+}{\Gamma \vdash [P \oplus Q] \triangleright A^+} \text{ (sum)} \\
 \\
 \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash q\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^{-}, \tilde{p} : \tilde{\phi}^{-}, a : \iota^{-}} \text{ (in)} \qquad \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash \bar{q}\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^{+}, \tilde{p} : \tilde{\phi}^{+}, a : \iota^{+}} \text{ (out)} \\
 \\
 \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash a(x) P \triangleright a : \iota^{-}, A} \text{ (d-in)} \qquad \frac{}{\Gamma \vdash \bar{a}\langle E \rangle \triangleright a : \iota^{+}} \text{ (a-out)} \qquad \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash P \triangleright A, v : S^{\uparrow\downarrow}} \text{ (w)} \\
 \\
 \frac{}{\Gamma, F : \phi \vdash F^{\natural} \triangleright q : \phi^{-}} \text{ (p-v)} \qquad \frac{\Gamma, F : \phi \vdash P \triangleright q : \phi^{-}}{\Gamma \vdash \text{rec} F.P \triangleright q : \phi^{-}} \text{ (rec)} \qquad \frac{\Gamma \vdash P \triangleright A, v : S^{\uparrow\downarrow}}{\Gamma \vdash (\nu v) P \triangleright A} \text{ (Res)}
 \end{array}$$

- linearity**
- subject reduction**

Typing rules

Typing judgment in a natural deduction style: $\Gamma \vdash P \triangleright A$

$$\begin{array}{c}
 \frac{}{\Gamma \vdash \mathbf{0} \triangleright _} \text{ (z)} \qquad \frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright B \quad A \asymp B}{\Gamma \vdash P \parallel Q \triangleright A \odot B} \text{ (par)} \qquad \frac{\Gamma \vdash P \triangleright A^+ \quad \Gamma \vdash Q \triangleright A^+}{\Gamma \vdash [P \oplus Q] \triangleright A^+} \text{ (sum)} \\
 \\
 \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash q\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^-, \tilde{p} : \tilde{\phi}^-, a : \iota^-} \text{ (in)} \qquad \frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash \bar{q}\langle \tilde{p}, a \rangle \triangleright q : [\tilde{\phi}, \iota]^+, \tilde{p} : \tilde{\phi}^+, a : \iota^+} \text{ (out)} \\
 \\
 \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash a(x)P \triangleright a : \iota^-, A} \text{ (d-in)} \qquad \frac{}{\Gamma \vdash \bar{a}\langle E \rangle \triangleright a : \iota^+} \text{ (a-out)} \qquad \frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash P \triangleright A, v : S^\updownarrow} \text{ (w)} \\
 \\
 \frac{}{\Gamma, F : \phi \vdash F^\natural \triangleright q : \phi^-} \text{ (p-v)} \qquad \frac{\Gamma, F : \phi \vdash P \triangleright q : \phi^-}{\Gamma \vdash \text{rec}F.P \triangleright q : \phi^-} \text{ (rec)} \qquad \frac{\Gamma \vdash P \triangleright A, v : S^\updownarrow}{\Gamma \vdash (\nu v)P \triangleright A} \text{ (Res)}
 \end{array}$$

- linearity**
- subject reduction**
- confluence**

Processes

Outline

*sl*PCF

*lin*Solos

▷ Main Results

Encoding

Example

Equivalences

Results

Can we code a more
parallel evaluation
strategy?

Conclusions

Main Results

Encoding

We denote $q(\tilde{v}) = (\nu\tilde{v})q\langle\tilde{v}\rangle$ and $\bar{q}(\tilde{v}) = (\nu\tilde{v})\bar{q}\langle\tilde{v}\rangle$

$$\llbracket 0^l \rrbracket^{q_\epsilon} = q_\epsilon(\mathbf{a}_\epsilon) \bar{\mathbf{a}}_\epsilon \langle 0 \rangle$$

$$\text{(i.e. } \llbracket \underline{\mathbf{n}}^l \rrbracket^{q_\epsilon} = q_\epsilon(\mathbf{a}_\epsilon) \bar{\mathbf{a}}_\epsilon \langle \underline{\mathbf{n}} \rangle)$$

$$\llbracket \text{succ}^{l \rightarrow o_l} \rrbracket^{q_\epsilon} = q_\epsilon(q_1, \mathbf{a}_\epsilon) \bar{q}_1(\mathbf{a}_1) \mathbf{a}_1(x) \bar{\mathbf{a}}_\epsilon \langle \text{succ}(x) \rangle$$

$$\llbracket \text{pred}^{l \rightarrow o_l} \rrbracket^{q_\epsilon} = q_\epsilon(q_1, \mathbf{a}_\epsilon) \bar{q}_1(\mathbf{a}_1) \mathbf{a}_1(x) \bar{\mathbf{a}}_\epsilon \langle \text{pred}(x) \rangle$$

Encoding

We denote $q(\tilde{v}) = (\nu\tilde{v})q\langle\tilde{v}\rangle$ and $\bar{q}(\tilde{v}) = (\nu\tilde{v})\bar{q}\langle\tilde{v}\rangle$

$$\llbracket F^\sigma \rrbracket^{q_\epsilon} = F \smile^{q_\epsilon}$$

$$\llbracket \mu F^\sigma . M^\sigma \rrbracket^{q_\epsilon} = \text{rec} F . \llbracket M^\sigma \rrbracket^{q_\epsilon}$$

Encoding

We denote $q(\tilde{v}) = (\nu\tilde{v})q\langle\tilde{v}\rangle$ and $\bar{q}(\tilde{v}) = (\nu\tilde{v})\bar{q}\langle\tilde{v}\rangle$

$$[[x^\iota]]^{q_\epsilon} = q_\epsilon(\mathbf{a}_\epsilon) \bar{\mathbf{a}}_\epsilon\langle\mathbf{x}\rangle$$

$$[[\lambda x^\iota . M^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \iota}]]^{q_\epsilon} = q_\epsilon(q_0, q_1, \dots, q_n, \mathbf{a}_\epsilon) \bar{q}_0(\mathbf{a}_0) \mathbf{a}_0\langle\mathbf{x}\rangle \left[\begin{array}{l} (\nu p)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel [[M]]^p) \\ \oplus \mathbf{x} \\ (\nu p)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel [[M]]^p) \end{array} \right]$$

Encoding

We denote $q(\tilde{v}) = (\nu\tilde{v})q\langle\tilde{v}\rangle$ and $\bar{q}(\tilde{v}) = (\nu\tilde{v})\bar{q}\langle\tilde{v}\rangle$

$$\llbracket \mathbf{x}^\iota \rrbracket^{q_\epsilon} = q_\epsilon(\mathbf{a}_\epsilon) \bar{\mathbf{a}}_\epsilon\langle\mathbf{x}\rangle$$

$$\llbracket \lambda \mathbf{x}^\iota . M^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \iota} \rrbracket^{q_\epsilon} = q_\epsilon(q_0, q_1, \dots, q_n, \mathbf{a}_\epsilon) \bar{q}_0(\mathbf{a}_0) \mathbf{a}_0(\mathbf{x}) \left[\begin{array}{l} (\nu p)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel \llbracket M \rrbracket^p) \\ \oplus \mathbf{x} \\ (\nu p)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel \llbracket M \rrbracket^p) \end{array} \right]$$

$$\llbracket \mathbf{f}^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \iota} \rrbracket^{q_\epsilon} = q_\epsilon(q_1, \dots, q_n, \mathbf{a}_\epsilon) \bar{\mathbf{f}}\langle q_1, \dots, q_n, \mathbf{a}_\epsilon \rangle \quad \text{where } n \geq 1$$

$$\llbracket \lambda \mathbf{f}^\tau . M^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \iota} \rrbracket^{q_\epsilon} = q_\epsilon(\mathbf{f}, q_1, \dots, q_n, \mathbf{a}) (\nu p)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel \llbracket M \rrbracket^p)$$

Encoding

We denote $q(\tilde{v}) = (\nu\tilde{v})q\langle\tilde{v}\rangle$ and $\bar{q}(\tilde{v}) = (\nu\tilde{v})\bar{q}\langle\tilde{v}\rangle$

$$\llbracket \mathbf{x}^\iota \rrbracket^{q_\epsilon} = q_\epsilon(\mathbf{a}_\epsilon) \bar{\mathbf{a}}_\epsilon\langle\mathbf{x}\rangle$$

$$\llbracket \lambda \mathbf{x}^\iota . M^{\sigma_1 \multimap \dots \multimap \sigma_n \multimap \iota} \rrbracket^{q_\epsilon} = q_\epsilon(q_0, q_1, \dots, q_n, \mathbf{a}_\epsilon) \bar{q}_0(\mathbf{a}_0) \mathbf{a}_0(\mathbf{x}) \left[\begin{array}{l} (\nu p)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel \llbracket M \rrbracket^p) \\ \oplus \mathbf{x} \\ (\nu p)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel \llbracket M \rrbracket^p) \end{array} \right]$$

$$\llbracket \mathbf{f}^{\sigma_1 \multimap \dots \multimap \sigma_n \multimap \iota} \rrbracket^{q_\epsilon} = q_\epsilon(q_1, \dots, q_n, \mathbf{a}_\epsilon) \bar{\mathbf{f}}\langle q_1, \dots, q_n, \mathbf{a}_\epsilon \rangle \quad \text{where } n \geq 1$$

$$\llbracket \lambda \mathbf{f}^\tau . M^{\sigma_1 \multimap \dots \multimap \sigma_n \multimap \iota} \rrbracket^{q_\epsilon} = q_\epsilon(\mathbf{f}, q_1, \dots, q_n, \mathbf{a}) (\nu p)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel \llbracket M \rrbracket^p)$$

$$\llbracket M^{\sigma_1 \multimap \dots \multimap \sigma_n \multimap \iota} N^{\sigma_1} \rrbracket^{q_\epsilon} = q_\epsilon(q_2, \dots, q_n, \mathbf{a}) (\nu p, q_1)(\bar{p}\langle q_1, \dots, q_n, \mathbf{a} \rangle \parallel \llbracket M \rrbracket^p \parallel \llbracket N \rrbracket^{q_1})$$

$$\llbracket \text{which? } ((\iota \multimap \iota) \multimap \iota) \multimap \iota \rrbracket^{q_\epsilon} = q_\epsilon(q_1, \mathbf{a}_\epsilon) \bar{q}_1(q_{11}, \mathbf{a}_1) q_{11}(q_{111}, \mathbf{a}_{11}) \bar{q}_{111}(\mathbf{a}_{111}) \\ \mathbf{a}_{111}(\mathbf{x}) \bar{\mathbf{a}}_{11}\langle\mathbf{x}\rangle \mathbf{a}_1(\mathbf{y}) \bar{\mathbf{a}}_\epsilon\langle\mathbf{x}, \mathbf{y}\rangle$$

Example

Processes

Outline

slPCF

linSolos

Main Results

Encoding

▷ Example

Equivalences

Results

Can we code a more
parallel evaluation
strategy?

Conclusions

$$q_1(q'_{11}, a') \bar{q}'_{11}(a_{11}) a_{11}(x) \bar{a}'\langle x \rangle$$

Identity on natural numbers ($\lambda x^t . x$)

Example

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

▷ Example

Equivalences

Results

Can we code a more
parallel evaluation
strategy?

Conclusions

$$q_1(q'_{11}, a') \bar{q}'_{11}(a_{11}) a_{11}(x) \bar{a}'\langle x \rangle$$

$$q_{11}(a'_{11}) \bar{a}'_{11}\langle 5 \rangle$$

Numeral five ([5](#))

Example

Processes

Outline

slPCF

linSolos

Main Results

Encoding

▷ Example

Equivalences

Results

Can we code a more
parallel evaluation
strategy?

Conclusions

$$\bar{q}_1 \langle q_{11}, a \rangle$$
$$q(a) (\nu q_1, q_{11}) q_1(q'_{11}, a') \bar{q}'_{11}(a_{11}) a_{11}(x) \bar{a}' \langle x \rangle$$
$$q_{11}(a'_{11}) \bar{a}'_{11} \langle 5 \rangle$$

Identity applied to five $((\lambda x'.x)\underline{5})$

Example

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

▷ Example


Equivalences

Results

Can we code a more
parallel evaluation
strategy?

Conclusions

$$\bar{q}_1 \langle q_{11}, a \rangle$$

 $q(a) (\nu q_1, q_{11}) q_1(q'_{11}, a') \bar{q}'_{11}(a_{11}) a_{11}(x) \bar{a}' \langle x \rangle$

$$q_{11}(a'_{11}) \quad \bar{a}'_{11} \langle 5 \rangle$$

External invocation on q

Example

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

▷ Example

Equivalences


Results

Can we code a more
parallel evaluation
strategy?


Conclusions

$$\bar{q}_1 \langle q_{11}, a \rangle$$

q_{11}, a



$q(a) (\nu q_1, q_{11}) q_1(q'_{11}, a') \bar{q}'_{11}(a_{11}) a_{11}(x) \bar{a}' \langle x \rangle$



$q_{11}(a'_{11}) \bar{a}'_{11} \langle 5 \rangle$

Communication along q_1 of the names q_{11}, a

Example

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

▷ Example

Equivalences

Results

Can we code a more
parallel evaluation
strategy?

Conclusions

$$\bar{q}_1 \langle q_{11}, a \rangle$$

q_{11}, a ↓

$q(a) (\nu q_1, q_{11}) \quad q_1(q_{11}, a) \quad \bar{q}_{11}(a_{11}) \quad a_{11}(x) \quad \bar{a} \langle x \rangle$

$$q_{11}(a'_{11}) \quad \bar{a}'_{11} \langle 5 \rangle$$

Unification of the names q'_{11}, a' with q_{11}, a

Example

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

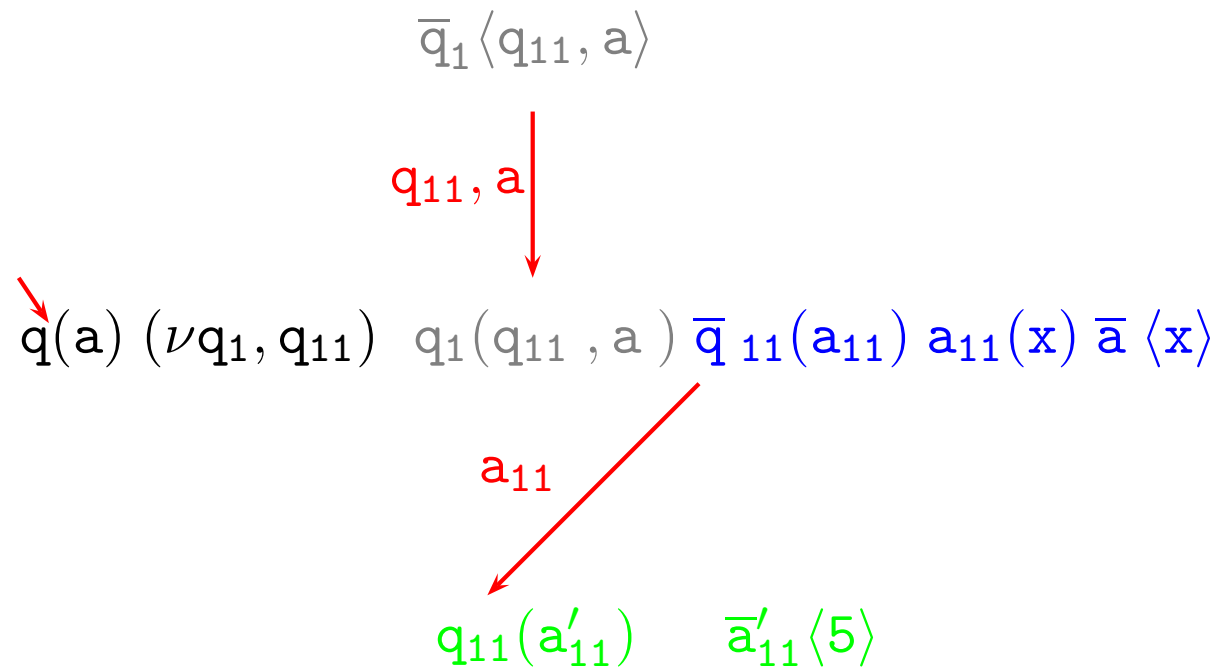
▷ Example

Equivalences

Results

Can we code a more
parallel evaluation
strategy?

Conclusions



Communication along q_{11} of the name a_{11}

Example

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

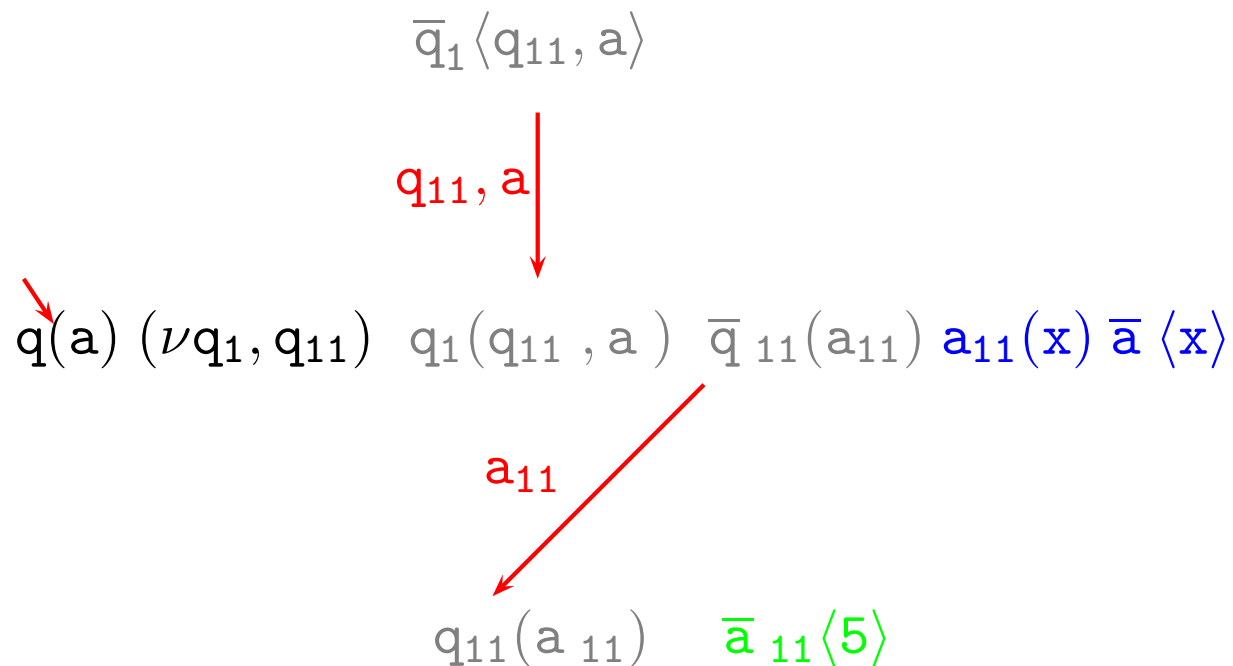
▷ Example

Equivalences

Results

Can we code a more parallel evaluation strategy?

Conclusions



Unification of the name a'_{11} with a_{11}

Example

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

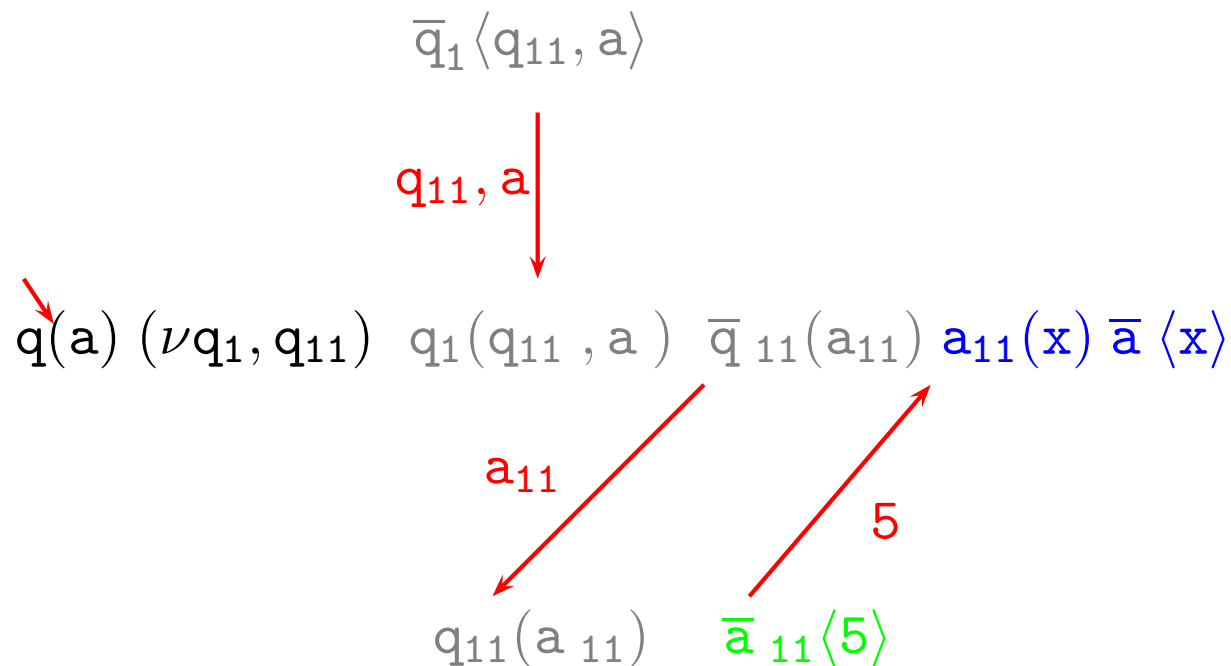
▷ Example

Equivalences

Results

Can we code a more parallel evaluation strategy?

Conclusions



Communication along a_{11} of the number 5

Example

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

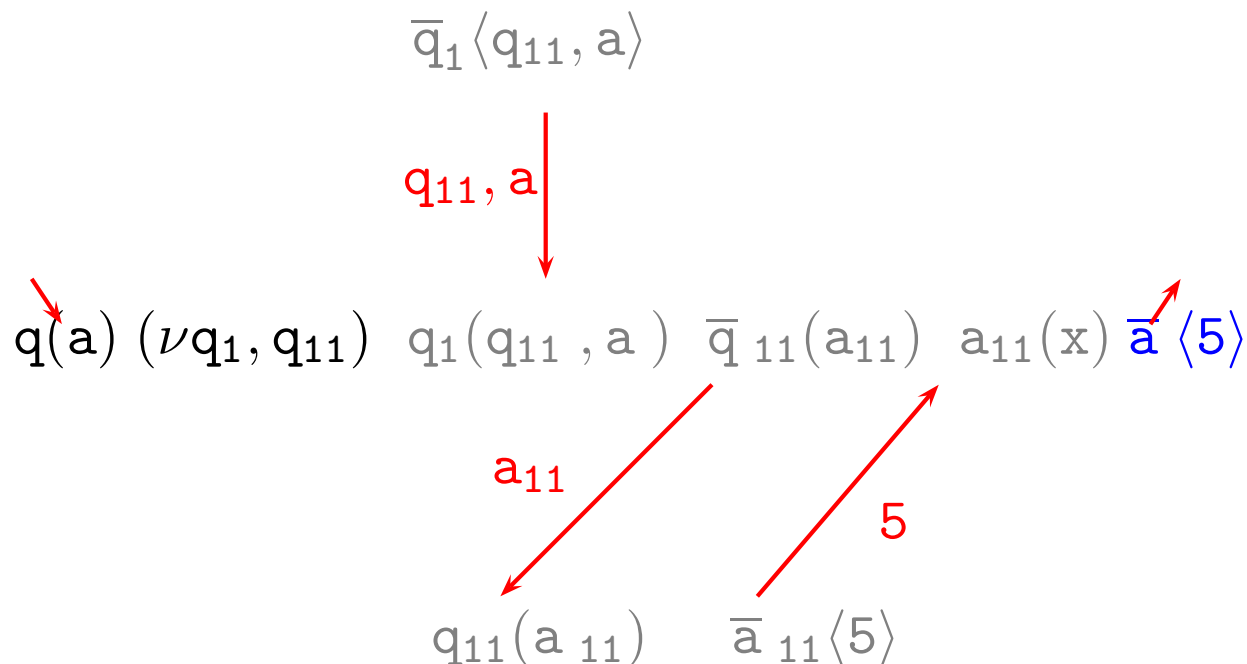
▷ Example

Equivalences

Results

Can we code a more parallel evaluation strategy?

Conclusions



Output of 5 to the external invocation

Equivalences

Processes

Outline

SlPCF

linSolos

Main Results

Encoding

Example

▷ Equivalences

Results

Can we code a more
parallel evaluation
strategy?

Conclusions

Operational equivalence in \mathcal{SlPCF}

□ Let $M^\sigma, N^\sigma \in \mathcal{SlPCF}$.

1. $M \lesssim_\sigma N$ whenever $\forall C[\sigma]$ s.t. $C[M], C[N] \in \mathcal{P}$, if $C[M] \Downarrow \underline{n}$ then $C[N] \Downarrow \underline{n}$.
2. $M \approx_\sigma N$ if and only if $M \lesssim_\sigma N$ and $N \lesssim_\sigma M$.

Operational equivalence in *SlPCF*

- Let $M^\sigma, N^\sigma \in \text{SlPCF}$.
 1. $M \lesssim_\sigma N$ whenever $\forall C[\sigma]$ s.t. $C[M], C[N] \in \mathcal{P}$, if $C[M] \Downarrow \underline{n}$ then $C[N] \Downarrow \underline{n}$.
 2. $M \approx_\sigma N$ if and only if $M \lesssim_\sigma N$ and $N \lesssim_\sigma M$.

Behavioral equivalence in *linSolos*

- Let $\Gamma \vdash P \triangleright a : i^+$. We use $\Gamma \vdash P \Downarrow_{\bar{a}\langle n \rangle}$ to denote that $P \longrightarrow^* P' \equiv (\nu \tilde{\kappa})(\bar{a}\langle E \rangle \| P'')$ where $E \longrightarrow n$.
- \cong_E is the maximum typed congruence on processes such that, $\Gamma \vdash P \cong_E Q$ and $\Gamma \vdash P \Downarrow_{\bar{a}\langle n \rangle}$ imply $\Gamma \vdash Q \Downarrow_{\bar{a}\langle n \rangle}$.

Results

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

Example

Equivalences

▷ Results

Can we code a more
parallel evaluation
strategy?

Conclusions

We can show the following results:

Computational Adequacy: $M^\ell \Downarrow n \iff \llbracket M^\ell \rrbracket^q \longrightarrow^* \llbracket n \rrbracket^q$

Results

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

Example

Equivalences

▷ Results

Can we code a more
parallel evaluation
strategy?

Conclusions

We can show the following results:

Computational Adequacy: $M^l \Downarrow n \iff \llbracket M^l \rrbracket^q \longrightarrow^* \llbracket n \rrbracket^q$

□ \Rightarrow : we can simulate lazy reductions in *lin*Solos.

We can show the following results:

Computational Adequacy: $M^\ell \Downarrow n \iff \llbracket M^\ell \rrbracket^q \longrightarrow^* \llbracket n \rrbracket^q$

- \Rightarrow : we can simulate lazy reductions in *lin*Solos.
- \Leftarrow : using a computability argument.

We can show the following results:

Computational Adequacy: $M^\ell \Downarrow n \iff \llbracket M^\ell \rrbracket^q \longrightarrow^* \llbracket n \rrbracket^q$

- \Rightarrow : we can simulate lazy reductions in *linSolos*.
- \Leftarrow : using a computability argument.

Correctness $\llbracket M^\sigma \rrbracket^q \cong_E \llbracket N^\sigma \rrbracket^q \Rightarrow M^\sigma \approx_\sigma N^\sigma$

We can show the following results:

Computational Adequacy: $M^\ell \Downarrow n \iff \llbracket M^\ell \rrbracket^q \longrightarrow^* \llbracket n \rrbracket^q$

- \Rightarrow : we can simulate lazy reductions in *linSolos*.
- \Leftarrow : using a computability argument.

Correctness $\llbracket M^\sigma \rrbracket^q \cong_E \llbracket N^\sigma \rrbracket^q \Rightarrow M^\sigma \approx_\sigma N^\sigma$

What about **completeness**?

We can show the following results:

Computational Adequacy: $M^\ell \Downarrow n \iff \llbracket M^\ell \rrbracket^q \longrightarrow^* \llbracket n \rrbracket^q$

- \Rightarrow : we can simulate lazy reductions in *lin*Solos.
- \Leftarrow : using a computability argument.

Correctness $\llbracket M^\sigma \rrbracket^q \cong_E \llbracket N^\sigma \rrbracket^q \Rightarrow M^\sigma \approx_\sigma N^\sigma$

What about **completeness**?

- **Conjecture:** $M^\sigma \approx_\sigma N^\sigma \Rightarrow \llbracket M^\sigma \rrbracket^q \cong_E \llbracket N^\sigma \rrbracket^q$ holds too.

Work in progress: Study of relevant contexts in *lin*Solos.

Can we code a more parallel evaluation strategy?

Processes
Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

Example

Equivalences

Results

Can we code a
more parallel
evaluation
strategy?

Conclusions

- Parallel β -reduction (without recursion)
 - It requires duplication

$$(\lambda f. \text{lif } x \ f \underline{5} \ f \underline{6}) I \rightarrow_{\beta} \text{lif } x \ I \underline{5} \ I \underline{6}$$

- *lin*Solos-reduction is not able to duplicate terms
- $\llbracket (\lambda f. M) N \rrbracket \longrightarrow^* \llbracket M[N/f] \rrbracket$,
if f does not appear under a *lif*-construct.

Can we code a more parallel evaluation strategy?

Processes
Outline

*sl*PCF

*lin*Solos

Main Results

Encoding

Example

Equivalences

Results

Can we code a
more parallel
evaluation
strategy?

Conclusions

- Parallel β -reduction (without recursion)

- It requires duplication

$$(\lambda f. \text{lif } x \ f \underline{5} \ f \underline{6}) I \rightarrow_{\beta} \text{lif } x \ I \underline{5} \ I \underline{6}$$

- *lin*Solos-reduction is not able to duplicate terms

- $\llbracket (\lambda f. M) N \rrbracket \longrightarrow^* \llbracket M[N/f] \rrbracket$,
if f does not appear under a *lif*-construct.

- Operational linearity

- Reduction does not duplicate terms
- *lin*Solos-reduction: a good point of departure

Processes

Outline

*sl*PCF

*lin*Solos

Main Results

▷ Conclusions

Motivation

Work in Progress

Future works

Conclusions

Processes

Outline

slPCF

linSolos

Main Results

Conclusions

▷ Motivation

Work in Progress

Future works

Motivations behind this paper are manyfolds.

- From a programming language point of view we provide a tool for study both parallel evaluation and equivalence of programs.

Processes

Outline

slPCF

linSolos

Main Results

Conclusions

▷ Motivation

Work in Progress

Future works

Motivations behind this paper are manyfolds.

- From a programming language point of view we provide a tool for study both parallel evaluation and equivalence of programs.
- From a process calculi point of view we give representation of a sequential language where redexes can be actually reduced in parallel.

Motivations behind this paper are manyfolds.

- From a programming language point of view we provide a tool for study both parallel evaluation and equivalence of programs.
- From a process calculi point of view we give representation of a sequential language where redexes can be actually reduced in parallel.
- From a game-semantics point of view, we open the road for a parallel description of strategies, since we avoid useless causality.

Work in Progress

Processes

Outline

*λ*PCF

linSolos

Main Results

Conclusions

Motivation

▷ Work in Progress

Future works

- Full abstraction
 - Study of the relevant contexts
 - Give a characterization of processes which are interpretation of *λ*PCF-programs
- Correspondence between proof-nets, differential nets and *linSolos*-processes
- Relational model of *linSolos*

Future works

Processes

Outline

*λ*PCF

*lin*Solos

Main Results

Conclusions

Motivation

Work in Progress

▷ Future works

- Connection with game semantics
 - What kind of Game Semantics strategies are behind our process calculus?
 - Connections with ludics and true concurrent models (event structures)
- Connection with classical denotational models for PCF
 - Continuous, stable and strongly stable model
 - Possible extensions?
- application to learning theory (problem of Stolzenberg [Barbanera, Berardi])
- parallel compilers of functional languages
- Optimality à la Levy