Comparing Deadlock-Free Session Typed Processes

Ornela Dardha (University of Glasgow)
Jorge A. Pérez (University of Groningen)

September 17, 2015

This work

A **formal comparison** between two behavioural type systems that enforce deadlock freedom for π-calculus processes

We consider two fundamentally different systems:

- Session types based on linear logic [Caires, Pfenning et al; Wadler]
- Usage types [Kobayashi et al]

We study $\mathcal{L}$ and $\mathcal{K}$, the deadlock-free languages they induce

Outline

Deadlock Freedom

Intuition: Processes do not "get stuck"

Examples (in CCS)

- A deadlocked process:

$$(\nu a)(\nu b)(\overline{a}.b.\mathbf{0} \mid \overline{b}.a.\mathbf{0})$$

- Two deadlock-free processes:

$$(\nu a)(\nu b)(b.\overline{a}.\mathbf{0} \mid \overline{b}.a.\mathbf{0}) \quad \text{and} \quad (\nu a)(\nu b)(\overline{a}.b.\mathbf{0} \mid a.\overline{b}.\mathbf{0})$$

Deadlock Freedom

Intuition: Processes do not "get stuck"

Examples (in CCS)

- A deadlocked process:

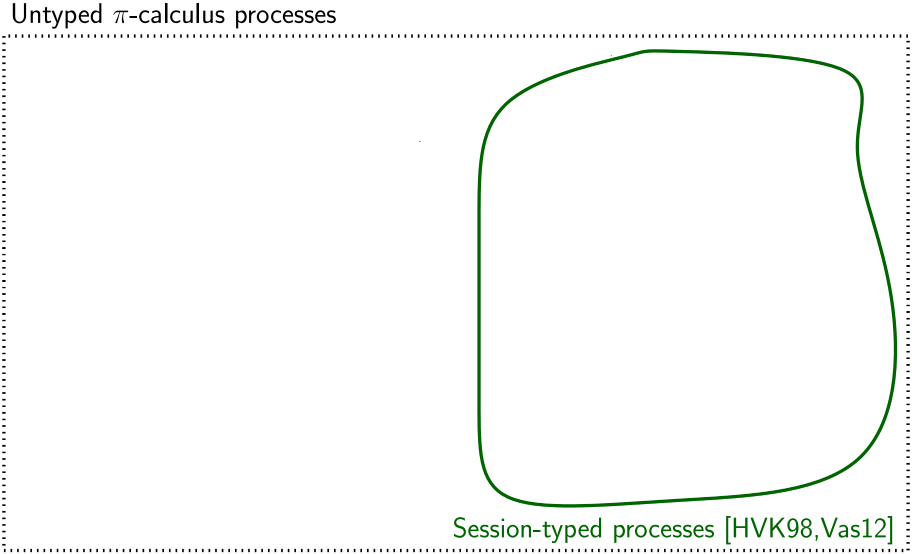
$$(\nu a)(\nu b)(\overline{a}.b.\mathbf{0} \mid \overline{b}.a.\mathbf{0})$$

- Two deadlock-free processes:

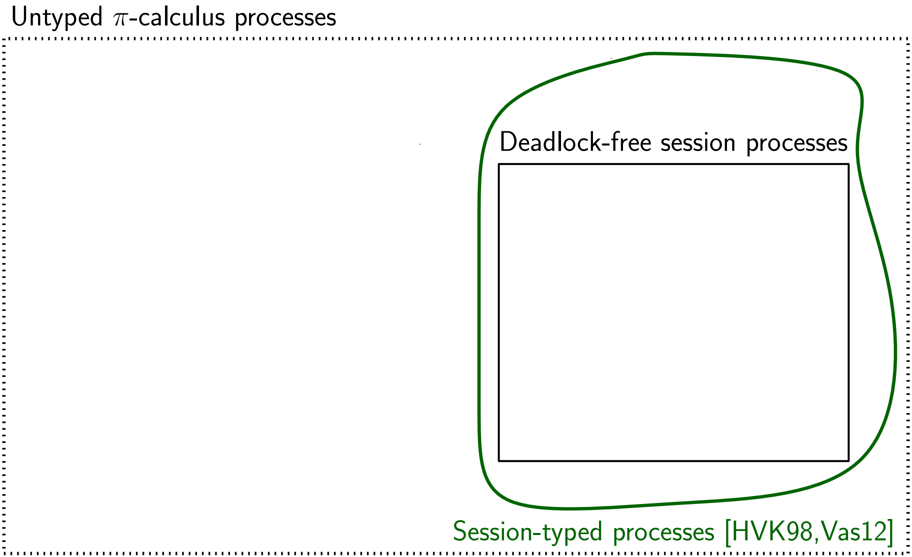
$$(\nu a)(\nu b)(b.\overline{a}.\mathbf{0} \mid \overline{b}.a.\mathbf{0}) \quad \text{and} \quad (\nu a)(\nu b)(\overline{a}.b.\mathbf{0} \mid a.\overline{b}.\mathbf{0})$$

- A liveness property: "A process is deadlock-free if it can always reduce until it eventually terminates, unless it diverges"
- Relevant in practice. Many type systems proposed [cf. Kobayashi et al; Dezani, de' Liguoro & Yoshida; Carbone & Debois]
- Our motivation: clarifying how these different systems are related

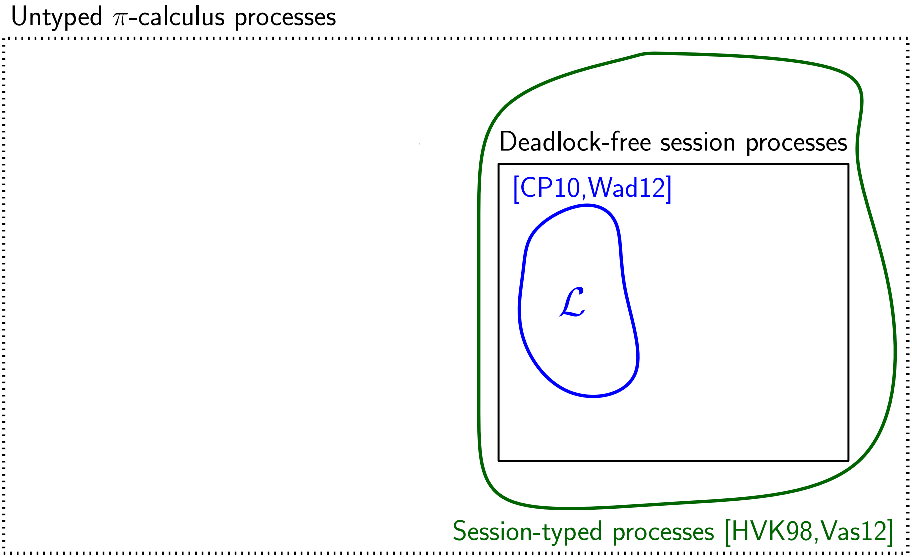
The landscape of (un)typed processes



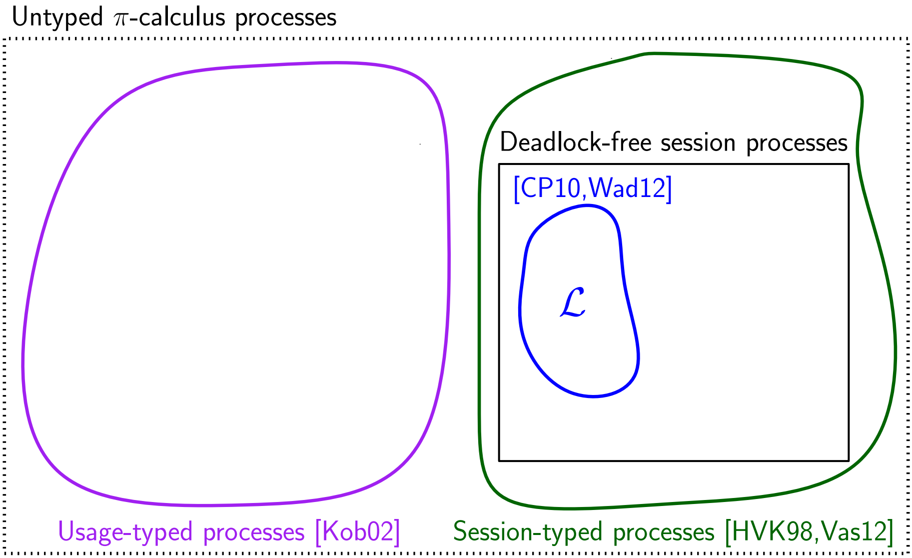
The landscape of (un)typed processes



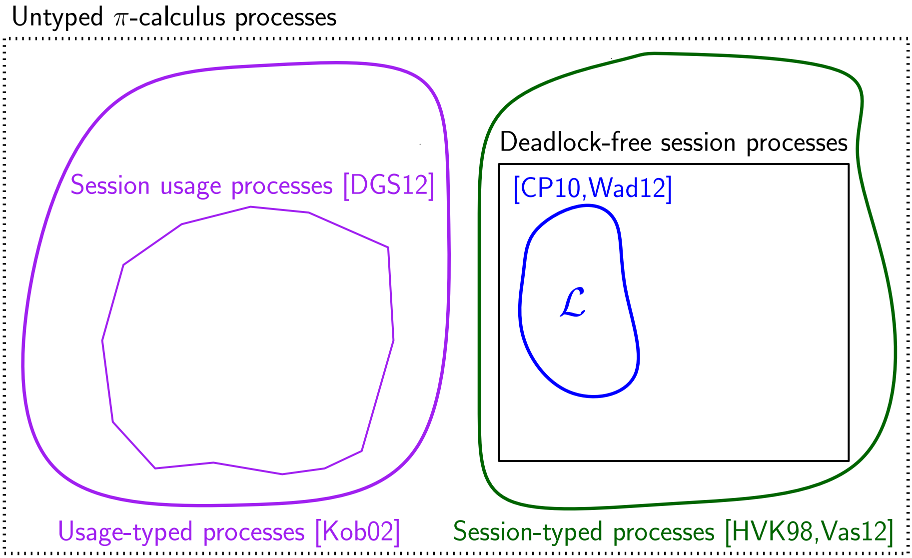
The landscape of (un)typed processes



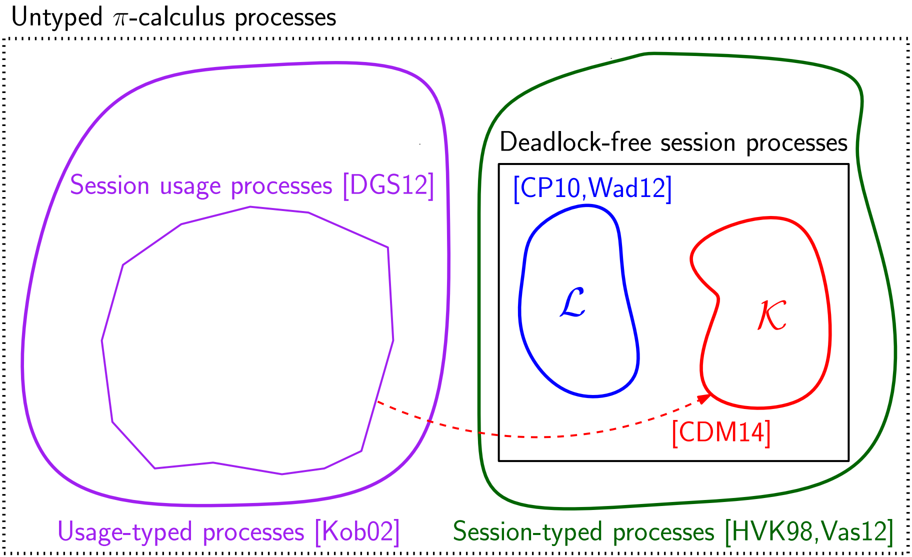
The landscape of (un)typed processes



The landscape of (un)typed processes



The landscape of (un)typed processes



The typed languages $\mathcal{L}$ and $\mathcal{K}$

$\mathcal{L}$: session processes obtained from the correspondence of linear logic propositions as session types [CP10, Wad12, Caires14].

$\mathcal{K}$: session processes are deadlock-free by combining a usage-based type system [Kob02] with the encodability result in [DGS12]

Notice

- $\mathcal{L}$ is **canonic**: due to its deep logical foundations, this is the most principled yardstick for comparisons
- $\mathcal{K}$ is **general**: it strictly includes classes of processes induced by previous type systems for deadlock-free sessions

The degree of sharing

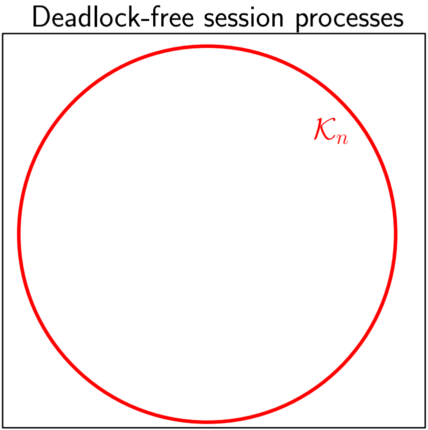
$\mathcal{K}$ is a **family** of classes of deadlock-free processes: $\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_n$

The family is defined by the **degree of sharing** between parallel components:

- $\mathcal{K}_0$ is the subclass with independent parallel composition: for all $P \mid Q \in \mathcal{K}_0$, P and Q **do not share** sessions
- $\mathcal{K}_1$ is the subclass which contains $\mathcal{K}_0$ but also processes with parallel components that share **at most** one session

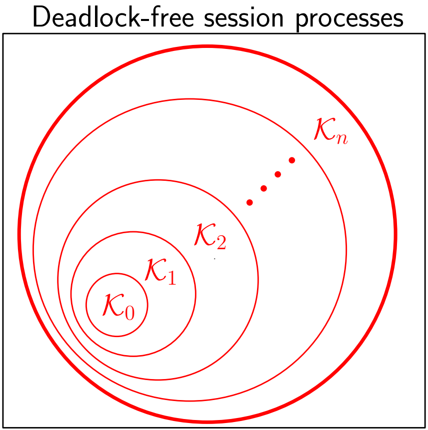
In general, $\mathcal{K}_n$ $(n \geq 0)$ contains deadlock-free session processes whose parallel components share at most n sessions

Our contributions



$\mathcal{K}_n$ deadlock-free processes by encodability [DGS12]

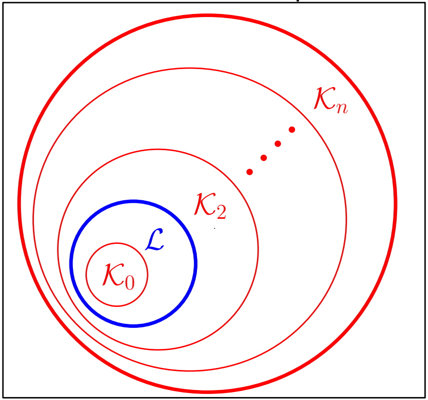
Our contributions



$\mathcal{K}_n$ deadlock-free processes by encodability [DGS12]

Our contributions

Deadlock-free session processes

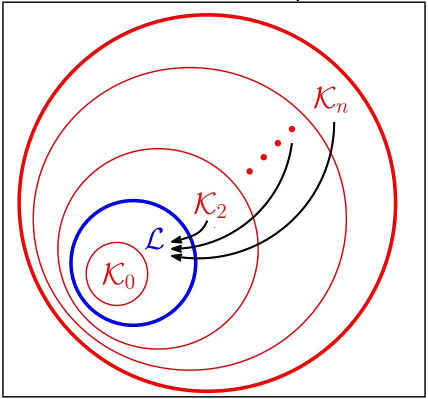


$\mathcal{K}_n$ deadlock-free processes by encodability [DGS12]

$\mathcal{L}$ deadlock-free processes based on linear logic [CP10, Caires14]

Our contributions

Deadlock-free session processes



$\mathcal{K}_n$	deadlock-free processes by encodability [DGS12]
$\mathcal{L}$	deadlock-free processes based on linear logic [CP10, Caires14]
$\longrightarrow$	type-preserving rewriting procedure

Outline

Technical Ingredients

Type systems:

$\Gamma \vdash_{\mathtt{ST}} P$	Simple session types
$P \vdash_{\mathtt{CH}} \Delta$	Linear logic propositions as session types [CP10]
$\Gamma \vdash_{\mathtt{KB}}^{n} P$	Usage type system [Kob02] w/ degree of sharing $n \geq 0$

Encodings/transformations:

$[\![\cdot]\!]_f$	Session π-processes into standard π-processes
$[\![\cdot]\!]$	Session types into usage types (defined in [DGS12])
$[\![\cdot]\!]_{\mathsf{c}}$	Session types into linear logic propositions
$\{\!\{\cdot\}\!\}$	Handling syntactic differences in processes (e.g. restriction)

The two deadlock-free session languages, formally

$$\mathcal{L} = \{P \mid \exists \Gamma. \ (\Gamma \vdash_{\mathtt{ST}} P \ \wedge \ \{\!\{P\}\!\} \vdash_{\mathtt{CH}} [\![\Gamma]\!]_{\mathsf{c}})\}$$

$$\mathcal{K}_n = \{P \mid \exists \Gamma, f. \ (\Gamma \vdash_{\mathtt{ST}} P \ \wedge \ [\![\Gamma]\!]_f \vdash_{\mathtt{KB}}^{n} [\![P]\!]_f)\} \quad (n \geq 0)$$

Session π-calculus

Syntax

$$
\begin{array}{rlll}
P, Q ::= & \overline{x}\langle v\rangle.P & & \text{(output)} \\
& x(y).P & & \text{(input)} \\
& x \triangleleft l_j.P & & \text{(selection)} \\
& x \triangleright \{l_i : P_i\}_{i\in I} & & \text{(branching)} \\
& (\boldsymbol{\nu} xy)P & & \text{(session restriction)} \\
& P \mid Q & & \text{(composition)} \\
& \mathbf{0} & & \text{(inaction)}
\end{array}
$$

Write **n** to denote a channel that cannot be used.

Reduction semantics

$$
(\boldsymbol{\nu} xy)(\overline{x}\langle v\rangle.P \mid y(z).Q) \to (\boldsymbol{\nu} xy)(P \mid Q[{}^{v}/{}_{z}])
$$

$$
(\boldsymbol{\nu} xy)(x \triangleleft l_j.P \mid y \triangleright \{l_i : P_i\}_{i\in I}) \to (\boldsymbol{\nu} xy)(P \mid P_j) \quad j \in I
$$

Session Types

$$
\begin{array}{rll}
T, S ::= & \textbf{end} & \text{(terminated protocol)} \\
& ?T.S & \text{(input value of type } T\text{, continue as } S\text{)} \\
& !T.S & \text{(output value of type } T\text{, continue as } S\text{)} \\
& \&\{l_i : S_i\}_{i \in I} & \text{(external choice)} \\
& \oplus\{l_i : S_i\}_{i \in I} & \text{(internal choice)}
\end{array}
$$

The typing system $\vdash_{\mathrm{ST}}$

- Ensures communication safety and session fidelity, but it does not exclude deadlocked processes.
- Example: process $(\boldsymbol{\nu} x_1 x_2)(\boldsymbol{\nu} y_1 y_2)(\overline{x}_1\langle \mathbf{n} \rangle.\overline{y}_1\langle \mathbf{n} \rangle \mid y_2(t).x_2(s))$ is typable in $\vdash_{\mathrm{ST}}$ but deadlocked. In contrast, the typable process

$$(\boldsymbol{\nu} x_1 x_2)(\boldsymbol{\nu} y_1 y_2)(\overline{x}_1\langle \mathbf{n} \rangle.\overline{y}_1\langle \mathbf{n} \rangle \mid x_2(s).y_2(t)) \text{ is deadlock-free}$$

Linear Logic Session Types [Caires14]

Syntax

- As before, but with standard restriction operator $(\boldsymbol{\nu} x)P$ and the **forwarding process** $[x \leftrightarrow y]$, which "fuses" names x and y
- We have the reduction rule: $(\boldsymbol{\nu} x)([x \leftrightarrow y] \mid P) \rightarrow P[y/x]$

Types (= linear logic propositions)

$$A, B ::= \underbrace{\perp \mid \mathbf{1}}_{\text{Terminated}} \mid \underbrace{A \otimes B}_{\text{Output}} \mid \underbrace{A \parr B}_{\text{Input}} \mid \underbrace{\oplus\{l_i : A_i\}_{i \in I}}_{\text{Int. Choice}} \mid \underbrace{\&\{l_i : A_i\}_{i \in I}}_{\text{Ext. Choice}}$$

Linear Logic Session Types [Caires14]

Some Typing Rules for $\vdash_{\mathtt{CH}}$

$$\text{(T-cut)}\ \frac{P \vdash_{\mathtt{CH}} \Delta, x{:}A \qquad Q \vdash_{\mathtt{CH}} \Delta', x{:}\overline{A}}{(\nu x)(P \mid Q) \vdash_{\mathtt{CH}} \Delta, \Delta'} \qquad \text{(T-mix)}\ \frac{P \vdash_{\mathtt{CH}} \Delta \qquad Q \vdash_{\mathtt{CH}} \Delta'}{P \mid Q \vdash_{\mathtt{CH}} \Delta, \Delta'}$$

Notable points:

- **Composition plus hiding** thanks to rule (T-cut)
- Rule (T-mix) types independent parallel composition
- **Properties**: Type preservation (subject reduction) and progress

Usage Types [Kob02]

Syntax

- **Polyadic communication** and standard restriction
- **Case construct case** v **of** $\{l_i\_x_i \triangleright P_i\}_{i \in I}$ with **variant value** l_j_v

Usage Types

$$
\begin{array}{rlll}
U ::= & ?^o_\kappa.U & \text{(used in input)} & \emptyset & \text{(not usable)} \\
 & !^o_\kappa.U & \text{(used in output)} & (U_1 \mid U_2) & \text{(used in parallel)} \\
T ::= & U[\widetilde{T}] & \text{(channel types)} & \langle l_i : T_i \rangle_{i \in I} & \text{(variant type)}
\end{array}
$$

Obligations o and **capabilities** κ describe channel dependencies.

- An obligation of level n must be fulfilled by using only capabilities of level **less than** n
- For an action with capability of level n, there must exist a co-action with obligation of level **less than or equal to** n

A usage U that satisfies these conditions is **reliable**, noted $\mathrm{rel}(U)$

Usage Types [Kob02]

Some Typing Rules for $\vdash^n_{\mathtt{KB}}$

$$\frac{\Gamma_1 \vdash^n_{\mathtt{KB}} P \quad \Gamma_2 \vdash^n_{\mathtt{KB}} Q \quad |\Gamma_1 \cap \Gamma_1| \leq n}{\Gamma_1 \mid \Gamma_2 \vdash^n_{\mathtt{KB}} P \mid Q}\ (\mathrm{T}\pi\text{-}\mathrm{PAR}_n)$$

$$\frac{\Gamma, x : U[\widetilde{T}] \vdash^n_{\mathtt{KB}} P \quad \mathsf{rel}(U)}{\Gamma \vdash^n_{\mathtt{KB}} (\boldsymbol{\nu} x)P}\ (\mathrm{T}\pi\text{-}\mathrm{RES})$$

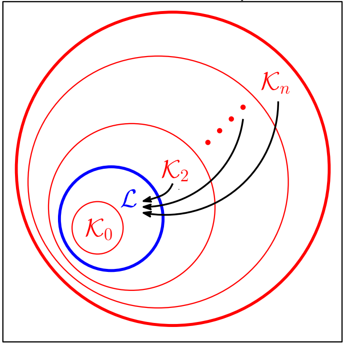
Notable points:

- Separate typing rules for restriction and parallel
- Novelty wrt [Kob02]: rule $(\mathrm{T}\pi\text{-}\mathrm{PAR}_n)$ defines degree of sharing
- Rule $(\mathrm{T}\pi\text{-}\mathrm{RES})$ checks usage reliability
- **Properties**: Type preservation and deadlock-freedom

Outline

Main results

Deadlock-free session processes



1. The inclusion between the constituent classes of $\mathcal{K}$ is **strict**

   We have: $\mathcal{K}_0 \subset \mathcal{K}_1 \subset \cdots \subset \mathcal{K}_{n-1} \subset \mathcal{K}_n$.

2. $\mathcal{L}$ and $\mathcal{K}_1$ **coincide**

   Logical foundations of session types induce the most basic form of concurrent cooperation: sharing exactly one session.

3. **Rewriting** of processes in $\mathcal{K}$ into $\mathcal{L}$

   Sequential prefixes replaced with representative parallel components.
   It enjoys type-preservation, operational correspondence, compositionality.

Inclusion: $\mathcal{K}_n \subset \mathcal{K}_{n+1}$

$\mathcal{K}_2$ contains (deadlock-free) session processes not captured in $\mathcal{K}_1$.

A representative example is:

$$P_2 = (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\, \overline{a_2}\langle x\rangle \mid \overline{b_1}\langle \mathbf{n}\rangle.\, b_2(z))$$

P_2 is typable in $\vdash^n_{\mathtt{KB}}$ (with $n \geq 2$) but not in $\vdash^1_{\mathtt{KB}}$, because it involves the composition of two processes which share two sessions

Inclusion: $\mathcal{K}_n \subset \mathcal{K}_{n+1}$

$\mathcal{K}_2$ contains (deadlock-free) session processes not captured in $\mathcal{K}_1$.

A representative example is:

$$P_2 = (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\, \overline{a_2}\langle x\rangle \mid \overline{b_1}\langle \mathbf{n}\rangle.\, b_2(z))$$

P_2 is typable in $\vdash^{n}_{\mathtt{KB}}$ (with $n \geq 2$) but not in $\vdash^{1}_{\mathtt{KB}}$, because it involves the composition of two processes which share two sessions

The idea generalises to show $\mathcal{K}_n \subset \mathcal{K}_{n+1}$, for all $n \geq 0$.

Coincidence: $\mathcal{K}_1 = \mathcal{L}$

Key aspects in the proof:

- Session types always result in "sequential" usages.
- Usages induced by LL session types are always reliable: they respect obligations/capabilities
- Preservation of duality through encodings of types
- Presence of both rules (T-mix) and (T-cut) for composition in $\vdash_{\mathtt{CH}}$

Rewriting $\mathcal{K}_n$ into $\mathcal{L}$

- If $P \in \mathcal{K}_{n+1}$ but $P \notin \mathcal{K}_n$ (with $n \geq 1$) then there is a subprocess of P that needs to be "adjusted" in order to "fit in" $\mathcal{K}_n$
- Such a subprocess of P must become more "parallel" in order to be typable under the smaller degree of sharing n

Rewriting $\mathcal{K}_n$ into $\mathcal{L}$: Intuition

- We propose a **rewriting procedure** that converts processes in $\mathcal{K}_n$ into processes in $\mathcal{K}_1$ (that is, $\mathcal{L}$)
- **Idea**: given a parallel process as input, return as output a process in which one component is kept unchanged, but the other is **replaced** by **parallel representatives** of the sessions in it.
- Using types, such representatives are defined as **characteristic processes** and **catalyzers**
- The procedure is type preserving and satisfies compositionality and operational correspondence

Rewriting $\mathcal{K}_n$ into $\mathcal{L}$: Example

Consider the $\mathcal{K}_2$ process

$$P_2 = (\boldsymbol{\nu} a_1 a_2)(\boldsymbol{\nu} b_1 b_2)(a_1(x).\ \overline{b_1}\langle x \rangle \mid \overline{a_2}\langle \mathbf{n} \rangle.\ b_2(z))$$

Omitting some details, the procedure rewrites P into either:

1. $(\boldsymbol{\nu} a)(\boldsymbol{\nu} b)(a(x).\ \overline{b}(w).([w \leftrightarrow x] \mid \mathbf{0}) \mid \overline{a}\langle \mathbf{n} \rangle.\mathbf{0} \mid\ b(z).\mathbf{0})$
2. $(\boldsymbol{\nu} a)(\boldsymbol{\nu} b)(a(x).\mathbf{0} \mid\ \overline{b}\langle x \rangle.\mathbf{0} \mid \overline{a}\langle v \rangle.([v \leftrightarrow \mathbf{n}] \mid b(z).\mathbf{0}))$

Outline

Concluding Remarks

- The first **formal comparison** between two behavioural type systems that enforce deadlock freedom for π-calculus processes
- We study $\mathcal{L}$ and $\mathcal{K}$, the deadlock-free languages induced by
  - Session types based on linear logic [Caires, Pfenning et al; Wadler]
  - Usage types [Kobayashi et al]
- We identify the **degree of sharing** as a subtle issue that determines new hierarchies in deadlock-free, session processes

Concluding Remarks

- The first **formal comparison** between two behavioural type systems that enforce deadlock freedom for π-calculus processes
- We study $\mathcal{L}$ and $\mathcal{K}$, the deadlock-free languages induced by
  - Session types based on linear logic [Caires, Pfenning et al; Wadler]
  - Usage types [Kobayashi et al]
- We identify the **degree of sharing** as a subtle issue that determines new hierarchies in deadlock-free, session processes

Future Work

- Processes with infinite behaviour
- Semantic characterizations of degree of sharing (e.g., preorders)
- Comparisons/integration with very recent work on static deadlock detection and resolution [Giunti, Francalanza & Ravara, WWV 2015]

Comparing Deadlock-Free Session Typed Processes

Ornela Dardha (University of Glasgow)
Jorge A. Pérez (University of Groningen)

September 17, 2015

Encodings of Processes and Types [DGS12]

Processes. The structure of a session-typed process is mimicked by sending its continuation as a payload over a channel. E.g.:

$$\begin{array}{rcl} [\![\overline{x}\langle v\rangle.P]\!]_f & \triangleq & (\boldsymbol{\nu} c)\overline{f_x}\langle v, c\rangle.[\![P]\!]_{f,\{x\mapsto c\}} \\ [\![x(y).P]\!]_f & \triangleq & f_x(y,c).[\![P]\!]_{f,\{x\mapsto c\}} \\ [\![(\boldsymbol{\nu} xy)P]\!]_f & \triangleq & (\boldsymbol{\nu} c)[\![P]\!]_{f,\{x,y\mapsto c\}} \end{array}$$

Types. $\bullet$ denotes $\mathbf{1}$ or $\perp$. Let $[\![\Gamma, x : T]\!]_f \triangleq [\![\Gamma]\!]_f, f_x : [\![T]\!]_{\mathsf{su}}$ and

$$\begin{array}{rcl} [\![\mathbf{end}]\!]_{\mathsf{su}} & = & \emptyset[] \\ [\![?T.S]\!]_{\mathsf{su}} & = & ?^o_\kappa[[\![T]\!]_{\mathsf{su}}, [\![S]\!]_{\mathsf{su}}] \\ [\![!T.S]\!]_{\mathsf{su}} & = & !^o_\kappa[[\![T]\!]_{\mathsf{su}}, [\![\overline{S}]\!]_{\mathsf{su}}] \end{array} \qquad \begin{array}{rcl} [\![\mathbf{end}]\!]_{\mathsf{c}} & = & \bullet \\ [\![?T.S]\!]_{\mathsf{c}} & = & [\![T]\!]_{\mathsf{c}} \invamp [\![S]\!]_{\mathsf{c}} \\ [\![!T.S]\!]_{\mathsf{c}} & = & [\![\overline{T}]\!]_{\mathsf{c}} \otimes [\![S]\!]_{\mathsf{c}} \end{array}$$

Lemma. Duality and encoding of session types

(i) $\overline{T} = S$ iff $\overline{[\![T]\!]_{\mathsf{c}}} = [\![S]\!]_{\mathsf{c}}$; (ii) $\overline{T} = S$ iff $\overline{[\![T]\!]_{\mathsf{su}}} = [\![S]\!]_{\mathsf{su}}$.