On Progress for Structured Communications

Mariangiola Dezani-Ciancaglini[†]
Ugo de'Liguoro[†]
Nobuko Yoshida[‡]

[†]Università di Torino, [‡]Imperial College London

October 6, 2007

Contents

Deadlock Freeness and Structured Communication

Deadlock Freeness and Structured Communication

Deadlock Freeness and Structured Communication

We investigate type systems for concurrent languages, using (a dialect of) the Pi-Calculus:

Deadlock Freeness and Structured Communication

We investigate type systems for concurrent languages, using (a dialect of) the Pi-Calculus:

- We consider communication structured in sessions;

Deadlock Freeness and Structured Communication

We investigate type systems for concurrent languages, using (a dialect of) the Pi-Calculus:

- We consider communication structured in sessions;
- Safeness is ensured by using a private channel;

Deadlock Freeness and Structured Communication

We investigate type systems for concurrent languages, using (a dialect of) the Pi-Calculus:

- We consider communication structured in sessions;
- Safeness is ensured by using a private channel;
- Known type systems ensure error freeness;

Deadlock Freeness and Structured Communication

We investigate type systems for concurrent languages, using (a dialect of) the Pi-Calculus:

- We consider communication structured in sessions;
- Safeness is ensured by using a private channel;
- Known type systems ensure error freeness;
- A desirable property is that communication within a session does not block;

Deadlock Freeness and Structured Communication

We investigate type systems for concurrent languages, using (a dialect of) the Pi-Calculus:

- We consider communication structured in sessions;
- Safeness is ensured by using a private channel;
- Known type systems ensure error freeness;
- A desirable property is that communication within a session does not block;
- We study how to enforce the latter property via a type system, building over session types [HVK 98] and type systems for partial deadlock freeness [Koba 98].

Progress Problem for Multiple Sessions

The notion of session

A session is an abstraction of a series of communications through a private channel between two parties.

Over a port name a (we call *shared channel*) a connection is established:

$$\text{accept } a(x) \text{ in } \ldots x \ldots \mid \text{request } a(y) \text{ in } \ldots y \ldots$$

which produces:

$$(\nu k)(\ldots k^{+} \ldots \mid \ldots k^{-} \ldots)$$

where k is a private channel name (we say a *live channel*); the "conversation" takes place over k in this session.

Multiple sessions

Sessions should complete: if we allow multiple sessions to be opened, they can block each other, although safe in isolation:

$$\text{accept } a(x) \text{ in accept } b(y) \text{ in } y!\langle 3\rangle.x?(z)\ldots \mid$$

$$\text{request } a(u) \text{ in request } b(v) \text{ in } u!\langle \text{“Hallo”}\rangle.v?(w)\ldots$$

which leads to the blocked situation:

$$(\nu k_a, k_b)(k_b^+!\langle 3\rangle.k_a^+?(z)\ldots \mid k_a^-!\langle \text{“Hallo”}\rangle.k_b^-?(w)\ldots)$$

Completion here is a liveness property, rather a form of (strong) normalization.

Syntax and Operational Semantics

Process syntax

Prefixes

$$
\begin{array}{rcll}
T & ::= & a(x).P & \text{accept} \\
 & \| & \star a(x).P & \text{permanent accept} \\
 & \| & \overline{a}(x).P & \text{request} \\
 & \| & \kappa!\langle e\rangle & \text{data send} \\
 & \| & \kappa?(x).P & \text{data receive} \\
 & \| & \kappa \triangleleft l.P & \text{selection} \\
 & \| & \kappa \triangleright \{l_1 : P_1 [\!] \ldots [\!] l_n : P_n\} & \text{branching} \\
 & \| & \kappa!\langle\!\langle \kappa' \rangle\!\rangle & \text{session send} \\
 & \| & \kappa?(\!(x)\!).P & \text{session receive} \\
 & \| & \text{if } e \text{ then } P \text{ else } Q & \text{if-then-else}
\end{array}
$$

where a is a shared channel, κ a live channel or a variable.

Process syntax

Processes

P	$::=$	$\mathbf{0}$	inaction
	$\|$	T	prefixed process
	$\|$	$P \,;\, Q$	sequencing
	$\|$	$P \,\|\, Q$	parallel
	$\|$	$(\boldsymbol{\nu} a)P$	shared channel hiding
	$\|$	$(\boldsymbol{\nu} k)P$	live channel hiding

Operational semantics

Main rules are:

$$\text{[Con]} \quad a(x).P|\overline{a}(y).Q \to (\nu k)(P\{k^+/x\}|Q\{k^-/y\})$$

$$\text{[ConR]} \quad \star a(x).P|\overline{a}(y).Q \to (\nu k)(P\{k^+/x\}|\star a(x).P|Q\{k^-/y\})$$

$$\text{[ComV]} \quad k^p!\langle e\rangle; P|k^{\overline{p}}?(x).Q \to P|Q\{v/x\}$$

$$\text{[Label]} \quad k^p \triangleleft l_i.P|k^{\overline{p}} \triangleright \{l_1 : Q_1 [\!] \ldots [\!] l_n : Q_n\} \to P|Q_i$$

$$\text{[ComS]} \quad (k^p!\langle\!\langle k_1^q\rangle\!\rangle; P)|(k^{\overline{p}}?(\!(x)\!).Q; R) \to Q\{k_1^q/x\}|P|R$$

where k is fresh in [Con] and [ConR], $e \downarrow v$ in [ComV], $1 \leq i \leq n$ in [Label], $k_1 \notin \mathsf{bn}(P)$ and $\mathsf{bn}(R) \cap \mathsf{bn}(Q) = \emptyset$ in [ComS].

Operational semantics

Main rules are:

$$\text{[Con]} \quad a(x).P|\overline{a}(y).Q \to (\nu k)(P\{k^+/x\}|Q\{k^-/y\})$$

$$\text{[ConR]} \quad \star a(x).P|\overline{a}(y).Q \to (\nu k)(P\{k^+/x\}|\star a(x).P|Q\{k^-/y\})$$

$$\text{[ComV]} \quad k^p!\langle e\rangle; P|k^{\overline{p}}?(x).Q \to P|Q\{v/x\}$$

$$\text{[Label]} \quad k^p \triangleleft l_i.P|k^{\overline{p}} \triangleright \{l_1 : Q_1 [\!] \ldots [\!] l_n : Q_n\} \to P|Q_i$$

$$\text{[ComS]} \quad (k^p!\langle\!\langle k_1^q\rangle\!\rangle; P)|(k^{\overline{p}}?(\!(x)\!).Q; R) \to Q\{k_1^q/x\}|P|R$$

where k is fresh in [Con] and [ConR], $e \downarrow v$ in [ComV], $1 \leq i \leq n$ in [Label], $k_1 \notin \mathsf{bn}(P)$ and $\mathsf{bn}(R) \cap \mathsf{bn}(Q) = \emptyset$ in [ComS].

The sequencing constructor

The sequencing constructor

Writing [ComS] as [ComV] yields:

$$(k^p!\langle\!\langle k_1^q \rangle\!\rangle; P) | k^{\overline{p}}?(x).Q \to P | Q\{k_1^q/x\} \quad k_1^q \notin \mathsf{bn}(P)$$

The sequencing constructor

Writing [ComS] as [ComV] yields:

$$(k^p!\langle\!\langle k_1^q \rangle\!\rangle; P) | k^{\overline{p}}?(\!(x)\!).Q \to P | Q\{k_1^q/x\} \quad k_1^q \notin \mathsf{bn}(P)$$

By using this last rule we have that e.g.

$$\overline{a}(x).\overline{b}(y).(y?(\!(z)\!).z!\langle 5\rangle); x?(t).\mathbf{0} \,|\, a(x').b(y').y'!\langle\!\langle x' \rangle\!\rangle$$

reduces to $(\boldsymbol{\nu} k)(k^-!\langle 5\rangle; k^+?(t).\mathbf{0})$ which is stuck.

The sequencing constructor

Writing [ComS] as [ComV] yields:

$$(k^p!\langle\!\langle k_1^q \rangle\!\rangle; P) | k^{\overline{p}}?(\!(x)\!).Q \to P | Q\{k_1^q/x\} \quad k_1^q \notin \mathsf{bn}(P)$$

By using this last rule we have that e.g.

$$\overline{a}(x).\overline{b}(y).(y?(\!(z)\!).z!\langle 5\rangle); x?(t).\mathbf{0} \mid a(x').b(y').y'!\langle\!\langle x' \rangle\!\rangle$$

reduces to $(\boldsymbol{\nu} k)(k^-!\langle 5\rangle; k^+?(t).\mathbf{0})$ which is stuck.

By means of our [ComS] we get instead $y?(\!(z)\!).z!\langle 5\rangle$.

The Notion of Progress

Progress property in general

Let M be a term of a calculus and Val $\subseteq$ NF the set of *values*: a type system satisfies the progress property if:

Progress $\quad \Gamma \vdash M : A \Rightarrow M \in \text{Val or } M \longrightarrow$

Combined with

Subject Reduction $\quad \Gamma \vdash M : A \;\&\; M \longrightarrow N \Rightarrow \Gamma \vdash N : A$

it gives error freeness of typeable terms.

Progress property for structured communications

Progress property for structured communications

A process P has the *progress property* if $P \rightarrow^* P'$ implies that:

Progress property for structured communications

A process P has the *progress property* if $P \rightarrow^* P'$ implies that:

1. P' does not contain live channels or

Progress property for structured communications

A process P has the *progress property* if $P \rightarrow^* P'$ implies that:

1. P' does not contain live channels or

2. $\exists Q, R.\ Q \not\rightarrow\ \&\ P' \,|\, Q \rightarrow R$ via some live channel k.

Progress property for structured communications

A process P has the *progress property* if $P \rightarrow^* P'$ implies that:

1. P' does not contain live channels or

2. $\exists Q, R.\ Q \not\rightarrow\ \&\ P' \,|\, Q \rightarrow R$ via some live channel k.

In words: P has the progress property if P itself and all its reducts either do not contain any open session, or they are capable to interact in a session with other parties.

Progress property for structured communications

A process P has the *progress property* if $P \rightarrow^* P'$ implies that:

1. P' does not contain live channels or

2. $\exists Q, R.\ Q \not\rightarrow\ \&\ P' \,|\, Q \rightarrow R$ via some live channel k.

In words: P has the progress property if P itself and all its reducts either do not contain any open session, or they are capable to interact in a session with other parties.

If P has the progress property then no session will get stuck.

Progress versus Error Freeness

Adapting from [HVK 98], P is error free if whenever

$$P \equiv T; Q | T'; Q' | R$$

with T, T' prefixed processes with the same live channel as subject, T and T' are dual actions:

- $\kappa!\langle e \rangle$ is dual of $\kappa?(x).P$
- $\kappa \triangleleft l.P$ is dual of $\kappa \triangleright \{l_1 : P_1 [\!] \ldots [\!] l_n : P_n\}$
- $\kappa!\langle\!\langle \kappa' \rangle\!\rangle$ is dual of $\kappa?(\!(x)\!).P$

Or if there are more than two prefixes in parallel with the same subject κ.

Progress versus Error Freeness

Adapting from [HVK 98], P is error free if whenever

$$P \equiv T; Q | T'; Q' | R$$

with T, T' prefixed processes with the same live channel as subject, T and T' are dual actions:

- $\kappa!\langle e \rangle$ is dual of $\kappa?(x).P$
- $\kappa \triangleleft l.P$ is dual of $\kappa \triangleright \{l_1 : P_1 [\!] \ldots [\!] l_n : P_n\}$
- $\kappa!\langle\!\langle \kappa' \rangle\!\rangle$ is dual of $\kappa?(\!(x)\!).P$

Or if there are more than two prefixes in parallel with the same subject κ.

Session type systems ensure error freeness, but not progress!

Example: circularity of channels

Consider:

$$\begin{aligned} P_2 &= a(x).b(y).(x!\langle 3\rangle; x?(z).y!\langle \text{“}Hallo''\rangle; P_2') \\ Q_2 &= \overline{a}(x).\overline{b}(y).(y?(z').x?(z).x!\langle 5\rangle; Q_2') \end{aligned}$$

Example: circularity of channels

Consider:

$$\begin{aligned} P_2 &= a(x).b(y).(x!\langle 3\rangle; x?(z).y!\langle \text{“}Hallo''\rangle; P_2') \\ Q_2 &= \overline{a}(x).\overline{b}(y).(y?(z').x?(z).x!\langle 5\rangle; Q_2') \end{aligned}$$

Then $P_2|Q_2$ reduces to a deadlock [Koba 98]:

$$\begin{aligned} (\nu k_1, k_2) \quad (\quad & k_1^+!\langle 3\rangle; k_1^+?(z).k_2^+!\langle \text{“}Hallo''\rangle; P_2' \quad | \\ & k_2^-?(z').k_1^-?(z).k_1^-!\langle 5\rangle; Q_2' \\ ) & \end{aligned}$$

This does not reduce nor it is able to interact along k_1, k_2 which are restricted: it does not satisfy the progress property.

Example: circularity of channels

Consider:

$$\begin{aligned} P_2 &= a(x).b(y).(x!\langle 3\rangle; x?(z).y!\langle \text{“}Hallo''\rangle; P_2') \\ Q_2 &= \overline{a}(x).\overline{b}(y).(y?(z').x?(z).x!\langle 5\rangle; Q_2') \end{aligned}$$

Then $P_2|Q_2$ reduces to a deadlock [Koba 98]:

$$\begin{aligned} (\nu k_1, k_2) \quad (\quad & k_1^+!\langle 3\rangle; k_1^+?(z).k_2^+!\langle \text{“}Hallo''\rangle; P_2' \quad | \\ & k_2^-?(z').k_1^-?(z).k_1^-!\langle 5\rangle; Q_2' \\ ) & \end{aligned}$$

This does not reduce nor it is able to interact along k_1, k_2 which are restricted: it does not satisfy the progress property.

Nor it is an error: the leading prefixes have different subjects.

Type System

Session Types

Type syntax is as follows:

direction	$\dagger$	$::=$	$! \mid ?$
partial session	σ	$::=$	$\varepsilon \mid \dagger\,\mathsf{t} \mid \dagger\,\mathsf{s} \mid \sigma.\sigma \mid \oplus_{1\leq i\leq n}\{l_i : \sigma_i\} \mid \&_{1\leq i\leq n}\{l_i : \sigma_i\}$
ended session	s	$::=$	$\sigma.\,\mathsf{end} \mid \oplus_{1\leq i\leq n}\{l_i : \mathsf{s}_i\} \mid \&_{1\leq i\leq n}\{l_i : \mathsf{s}_i\}$
standard type	t	$::=$	$[\mathsf{s}] \mid \mathsf{bool} \mid \mathsf{int} \mid \ldots$
running session	τ	$::=$	$\sigma \mid \mathsf{s}$

We say that $\overline{\tau}$ is the dual of τ where:

- $\overline{!} = ?$ and $\overline{?} = !$
- $\overline{\oplus_{1\leq i\leq n}\{l_i : \sigma_i\}} = \&_{1\leq i\leq n}\{l_i : \overline{\sigma}_i\}$, $\overline{\&_{1\leq i\leq n}\{l_i : \sigma_i\}} = \oplus_{1\leq i\leq n}\{l_i : \overline{\sigma}_i\}$
- $\overline{\varepsilon} = \varepsilon$, $\overline{\dagger\mathsf{s}} = \overline{\dagger}\mathsf{s}$, $\overline{\dagger\mathsf{t}} = \overline{\dagger}\mathsf{t}$, $\overline{\sigma.\mathsf{end}} = \overline{\sigma}.\mathsf{end}$, $\overline{\sigma_1.\sigma_2} = \overline{\sigma}_1.\overline{\sigma_2}$

Session Types

Typing judgments have the shape:

$$\Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta \parallel \mathcal{C}$$

where:

- Γ := environment for shared channels and variables;
- $\mathcal{S}$:= sendable shared channel names;
- $\mathcal{B}$:= bound shared channel names;
- Δ := environment for live channels and variables;
- $\mathcal{C}$:= channel relation.

Session Types

A channel relation is a directed graph, written (λ any name):

$$\mathcal{C} ::= \emptyset \parallel \mathcal{C}, \lambda \parallel \mathcal{C}, \lambda \prec \lambda'$$

A relation is well-formed if it has no cycle of length > 1 and it does not relate shared channels with themselves.

An implicit side condition to typing rules is that relations in the conclusions are well-formed.

Basic Typing Rules

First a shared channel name has a type which is dual on the opposite extremes:

$$\frac{}{\Gamma, a:[\mathsf{s}] \vdash a:[\mathsf{s}']}\ SChan$$

$$\frac{\Gamma \vdash a:[\mathsf{s}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta, x:\mathsf{s} \parallel \mathcal{C} \quad a \notin \mathcal{S} \cup \mathcal{B}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash a(x).P : \Delta \parallel \mathcal{C}\{a/x\}}\ Acc$$

$$\frac{\Gamma \vdash a:[\overline{\mathsf{s}}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta, x:\mathsf{s} \parallel \mathcal{C} \quad a \notin \mathcal{S} \cup \mathcal{B}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \overline{a}(x).P : \Delta \parallel \mathcal{C}\{a/x\}}\ Req$$

Basic Typing Rules

First a shared channel name has a type which is dual on the opposite extremes:

$$\frac{}{\Gamma, a:[\mathsf{s}] \vdash a:[\mathsf{s}']}\ SChan$$

$$\frac{\Gamma \vdash a:[\mathsf{s}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta, x:\mathsf{s} \parallel \mathcal{C} \quad a \notin \mathcal{S} \cup \mathcal{B}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash a(x).P : \Delta \parallel \mathcal{C}\{a/x\}}\ Acc$$

$$\frac{\Gamma \vdash a:[\overline{\mathsf{s}}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta, x:\mathsf{s} \parallel \mathcal{C} \quad a \notin \mathcal{S} \cup \mathcal{B}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \overline{a}(x).P : \Delta \parallel \mathcal{C}\{a/x\}}\ Req$$

dual session types

Basic Typing Rules

First a shared channel name has a type which is dual on the opposite extremes:

$$\frac{}{\Gamma, a:[\mathsf{s}] \vdash a:[\mathsf{s}']}\ SChan$$

$$\frac{\Gamma \vdash a:[\mathsf{s}] \quad \Gamma;\mathcal{S};\mathcal{B} \vdash P : \Delta, x:\mathsf{s} \,\|\, \mathcal{C} \quad a \notin \mathcal{S} \cup \mathcal{B}}{\Gamma;\mathcal{S};\mathcal{B} \vdash a(x).P : \Delta \,\|\, \mathcal{C}\{a/x\}}\ Acc$$

$$\frac{\Gamma \vdash a:[\overline{\mathsf{s}}] \quad \Gamma;\mathcal{S};\mathcal{B} \vdash P : \Delta, x:\mathsf{s} \,\|\, \mathcal{C} \quad a \notin \mathcal{S} \cup \mathcal{B}}{\Gamma;\mathcal{S};\mathcal{B} \vdash \overline{a}(x).P : \Delta \,\|\, \mathcal{C}\{a/x\}}\ Req$$

renaming of x by a

Basic Typing Rules

Rules for communicating along a live channel:

$$\frac{\Gamma \vdash e : \mathsf{t} \quad \text{if } e = a \text{ then } a \in \mathcal{S}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \kappa!\langle e \rangle : \{\kappa : !\mathsf{t}\} \parallel \{\ell(\kappa)\}} \; Snd$$

$$\frac{\Gamma, x : \mathsf{t}; \mathcal{S}; \mathcal{B} \vdash P : \Delta \parallel \mathcal{C}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \kappa?(x).P : \{\kappa : ?\mathsf{t}\} \cdot \Delta \parallel \mathsf{pre}(\{\ell(\kappa)\}, \mathcal{C})} \; Rcv$$

Rule for parallel composition:

$$\frac{\Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta \parallel \mathcal{C} \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash Q : \Delta' \parallel \mathcal{C}'}{\Gamma; \mathcal{S}; \mathcal{B} \vdash P \mid Q : \Delta \cup \Delta' \parallel \mathcal{C} \cup \mathcal{C}'} \; Par$$

where $\Delta \cup \Delta'$ is defined only if

$$k^+ \in \mathsf{dom}(\Delta) \text{ and } k^- \in \mathsf{dom}(\Delta') \Rightarrow \Delta(k^+) = \overline{\Delta'(k^-)}.$$

Example: circularity of channels revised

$$P_2 = a(x).b(y).(x!\langle 3\rangle; x?(z).y!\langle \text{“}Hallo''\rangle; P_2')$$
$$Q_2 = \overline{a}(x).\overline{b}(y).(y?(z').x?(z).x!\langle 5\rangle; Q_2')$$

Then we have, by (Snd), (Rcv):

$$\Gamma; \mathcal{S}; \mathcal{B} \vdash x!\langle 3\rangle; x?(z).y!\langle \text{“}Hallo''\rangle; P_2' : \Delta' \parallel \{x \prec y\}$$

$$\Gamma; \mathcal{S}; \mathcal{B} \vdash y?(z').x?(z).x!\langle 5\rangle; Q_2' : \Delta'' \parallel \{y \prec x\}$$

where $\Delta' = \Delta, x$:!int.?int, y :!string and $\Delta'' = \Delta, x$:?int.!int, y :?string. Hence, by (Acc) and (Req):

$$\Gamma; \mathcal{S}; \mathcal{B} \vdash P_2 : \Delta \parallel \{a \prec b\} \qquad \Gamma; \mathcal{S}; \mathcal{B} \vdash Q_2 : \Delta \parallel \{b \prec a\}$$

but $\{a \prec b, b \prec a\}$ is not well-formed, so that (Par) does not apply.

Sequencing and live channels

Consider:

$$P_6 \equiv \star a(x).k^+!\langle 3\rangle \qquad P_7 \equiv \overline{a}(y).\mathbf{0} \,|\, \overline{a}(z).\mathbf{0}$$

Then

$$P_6|P_7 \stackrel{*}{\longrightarrow} P_6 \,|\, k^+!\langle 3\rangle \,|\, k^+!\langle 3\rangle$$

which is non linear in k^+; on the other hand with

$$P_8 \equiv \star a(x).\mathbf{0}; k^+!\langle 3\rangle$$

we have

$$P_7|P_8 \stackrel{*}{\longrightarrow} P_8$$

which does not destroy linearity but progress, since k^+ is definitely blocked.

Sequencing and live channels

$$\frac{\Gamma \vdash a : [\mathsf{s}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \{x : \mathsf{s}\} \parallel \mathcal{C} \quad a \notin \mathcal{S}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \star a(x).P : \{\diamond\} \parallel \mathcal{C}\{a/x\}}\ Acc^{\star}$$

$$\frac{\Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta \parallel \mathcal{C} \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash Q : \Delta' \parallel \mathcal{C}'}{\Gamma; \mathcal{S}; \mathcal{B} \vdash P; Q : \Delta \cdot \Delta' \parallel \mathsf{pre}(\mathcal{C}, \mathcal{C}')}\ Seq$$

where $(\Delta \cdot \Delta')(\kappa) = \Delta(\kappa).\Delta'(\kappa)$ if $\kappa \in \mathsf{dom}(\Delta) \cap \mathsf{dom}(\Delta')$ and

$$\diamond \in \Delta \Rightarrow \Delta' \subseteq \{\diamond\},$$

$\Delta \cdot \Delta'$ is undefined otherwise.

Sequencing and live channels

Recall $P_8 \equiv \star a(x).\mathbf{0}; k^+!\langle 3\rangle$. Then

$$\Gamma; \mathcal{S}; \mathcal{B} \vdash \star a(x).\mathbf{0} : \{\diamond\} \,\|\, \mathcal{C}\{a/x\}$$

and

$$\Gamma; \mathcal{S}; \mathcal{B} \vdash k^+!\langle 3\rangle : \Delta', k : !\mathsf{int} \,\|\, \mathcal{C}'$$

But $\Delta', k : !\mathsf{int} \not\subseteq \{\diamond\}$ hence $\Delta \cdot (\Delta', k : !\mathsf{int})$ is undefined and P_8 is not typeable, nor it is $P_7 | P_8$.

Bound shared channels

A bound shared channel which does not have a dual to start a session can block the communication on live channels:

$$P_9 \equiv (\boldsymbol{\nu} a)(\overline{a}(x).k^{+}!\langle 3\rangle) \mid k^{-}?(y).\mathbf{0}.$$

The problem does not arise if the shared channel a is free, since we can always compose with dual processes, e.g.:

$$P_{10} \equiv \overline{a}(x).k^{+}!\langle 3\rangle \mid k^{-}?(y).\mathbf{0} \mid a(y).\mathbf{0}.$$

This is why we do not consider $\overline{a}(x).k^{+}!\langle 3\rangle$ as deadlocked, differently than [Koba 98].

Bound shared channels

$$\frac{\Gamma \vdash a : [\overline{\mathsf{s}}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \{x : \mathsf{s}\} \parallel \mathcal{C} \quad a \notin \mathcal{S}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \overline{a}(x).P : \{\diamond\} \parallel \mathcal{C}\{a/x\}}\ ReqB$$

$$\frac{\Gamma \vdash a : [\overline{\mathsf{s}}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \{x : \mathsf{s}\} \parallel \mathcal{C}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \overline{a}(x).P : \{\diamond\} \parallel \mathcal{C} \backslash\backslash x}\ ReqBS$$

$$\frac{\Gamma \vdash a : [\overline{\mathsf{s}}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta, x : \mathsf{s} \parallel \mathcal{C} \quad a \notin \mathcal{B}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \overline{a}(x).P : \Delta \parallel \mathcal{C} \backslash\backslash x}\ ReqS$$

$$\frac{\Gamma, a : [\mathsf{s}]; \mathcal{S}; \mathcal{B} \vdash P : \Delta \parallel \mathcal{C} \quad a \in \mathcal{B}}{\Gamma; \mathcal{S} \setminus a; \mathcal{B} \setminus a \vdash (\boldsymbol{\nu} a)P : \Delta \parallel \mathcal{C} \setminus a}\ HidingS$$

where $\mathcal{C} \backslash\backslash x = \mathcal{C} \setminus x$ if x is minimal w.r.t. $\mathcal{C}$.

Bound shared channels

To derive:

$$\Gamma; \mathcal{S}; \mathcal{B} \vdash (\boldsymbol{\nu} a)(\overline{a}(x).k^{+}!\langle 3\rangle) : \Delta \,\|\, \mathcal{C}$$

we have to use (HidingS), with a premise

$$\Gamma; \mathcal{S}'; \mathcal{B} \cup \{a\} \vdash \overline{a}(x).k^{+}!\langle 3\rangle : \Delta \,\|\, \mathcal{C}'$$

where $\Delta = \{\diamond\}$; but then the only applicable rules to type $k^{+}!\langle 3\rangle$ are (ReqB) and (ReqBs); they all need a premise

$$\Gamma; \mathcal{S}'; \mathcal{B} \cup \{a\} \vdash k^{+}!\langle 3\rangle : \{x : !\mathsf{int}\} \,\|\, \mathcal{C}''$$

which is not deducible since $k \notin \mathsf{dom}(\{x : !\mathsf{int}\})$.

Sending shared channels

Shared channels can be sent only if their dual processes can communicate without waiting other communications to succeed.

Consider

$$P_{11} \equiv \overline{a}(t).t!\langle b\rangle \mid a(x).\overline{c}(y).x?(z).\overline{z}(q).q?(w).y?(w').\mathbf{0}$$

$$P_{12} \equiv c(s).b(r).(s!\langle 3\rangle; r!\langle 4\rangle)$$

Then

$$P_{11} \mid P_{12} \xrightarrow{*} (\boldsymbol{\nu} k_b k_c)(k_b^+?(w).k_c^+?(t).\mathbf{0} \mid k_c^-!\langle 3\rangle; k_b^-!\langle 4\rangle)$$

which is a deadlock.
P_{11} can be safely put in parallel with

$$P'_{12} \equiv c(s).b(r).(r!\langle 4\rangle; s!\langle 3\rangle).$$

Sending shared channels

$P_{12} \equiv c(s).b(r).(s!\langle 3\rangle; r!\langle 4\rangle)$ cannot be typed by using rules (ReqS) and (ReqBS), since r is not minimum in its channel relation $\{s \prec r\}$.

$$\frac{\Gamma \vdash a : [\overline{\mathsf{s}}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta, x : \mathsf{s} \parallel \mathcal{C} \quad a \notin \mathcal{B}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \overline{a}(x).P : \Delta \parallel \mathcal{C} \backslash\backslash x} \; ReqS$$

$$\frac{\Gamma \vdash c : [\overline{\mathsf{s}}] \quad \Gamma; \mathcal{S}; \mathcal{B} \vdash P : \{x : \mathsf{s}\} \parallel \mathcal{C}}{\Gamma; \mathcal{S}; \mathcal{B} \vdash \overline{c}(x).P : \{\diamond\} \parallel \mathcal{C} \backslash\backslash x} \; ReqBS$$

Instead the process $P'_{12} \equiv c(s).b(r).(r!\langle 4\rangle; s!\langle 3\rangle)$ is typable using rules (ReqS) and (ReqBS).

Results

The Progress Theorem

Δ is balanced if $k^p : \tau$ and $k^{\overline{p}} : \tau' \in \Delta$ imply $\tau' = \overline{\tau}$.

Subject Reduction

1. If $\Gamma \vdash e : \mathsf{t}$ and $e \downarrow v$, then $\Gamma \vdash v : \mathsf{t}$.
2. If $\Gamma; \mathcal{S}; \mathcal{B} \vdash P : \Delta \parallel \mathcal{C}$, where Δ is balanced, and $P \to Q$, then $\Gamma; \mathcal{S}; \mathcal{B} \vdash Q : \Delta' \parallel \mathcal{C}'$, where Δ' is balanced, $\Delta' \sqsubseteq \Delta$ and $\mathcal{C}' \Subset \mathcal{C}$.

A process P is initial if it does not contain live channels nor any hiding of live channels.

Progress Theorem

All initial processes have the progress property.

Concluding Remarks

Achievements

Achievements

- A session typing system allowing interleaved sessions, which are not necessarily nested (improvement w.r.t. [CDY 07]);

Achievements

- A session typing system allowing interleaved sessions, which are not necessarily nested (improvement w.r.t. [CDY 07]);
- The typing system is based on the intuitive idea of channel causality without additional information nor type decoration;

Achievements

- A session typing system allowing interleaved sessions, which are not necessarily nested (improvement w.r.t. [CDY 07]);
- The typing system is based on the intuitive idea of channel causality without additional information nor type decoration;
- A progress theorem for session types.

Limitations

Limitations

- The treatment of recursion (in the form of repeated accepts) is restrictive;

Limitations

- The treatment of recursion (in the form of repeated accepts) is restrictive;
- We do not include definitions in the calculus, i.e. we do not have full recursion;

Limitations

- The treatment of recursion (in the form of repeated accepts) is restrictive;
- We do not include definitions in the calculus, i.e. we do not have full recursion;
- Session can interleave freely, but each one is still between two parties only (a feature of session types in general).

Future work

Future work

- To face the problem of treating full recursion and in a less restrictive way;

Future work

- To face the problem of treating full recursion and in a less restrictive way;
- To consider extensions of the system with bounded polymorphism and correspondence assertions;

Future work

- To face the problem of treating full recursion and in a less restrictive way;
- To consider extensions of the system with bounded polymorphism and correspondence assertions;
- To establish progress results that guarantee for choreographic (global) communication dependencies.

References

[CDY 07] M. Coppo, M. Dezani, N. Yoshida, "Asynchronous Session Types and Progress for Object-Oriented Languages", *LNCS* 4468 (2007), 1-31.

[HVK 98] K. Honda, V.T. Vasconcelos, M. Kubo, "Language Primitives and Type Disciplines for Structured Communication-based Programming", *LNCS* 1381 (1998), 22-138.

[Koba 98] N. Kobayashi "A Type System for Lock-Free Processes", Info. Comp. 177 (1998), 436-482.