Multiparty session typed message passing, as a computational effect

Dylan McDermott

dylan@dylanm.org

Message-passing with MPST as a computational effect

- ***Computational effect*** $\approx$ something observable a computation can do, beyond producing a result

  Examples: mutable state, exceptions, nondeterminism, ***message-passing***

- ***(Effect) grading***: static analysis of computational effects, especially in languages with eager evaluation

  Example: a ***session type*** describes the protocol followed by a message-passing computation

This talk

`SafeMP`: a call-by-value (eager) calculus with

- Asynchronous message-passing (buffer messages into queues)
- A graded type system: grades are multiparty session types, with asynchronous subtyping
- Communication safety, deadlock-freedom, and liveness properties, provided by the type system

Example: mutable state via message-passing

Implementation of state:

let rec $f x =$
  recv client $\{$
    $\texttt{get}\langle z : \textbf{unit}\rangle$. **send** $(\texttt{st}, x)$ **to** client **then** $f\ x$,
    $\texttt{put}\langle y : \textbf{int}\rangle$. $f\ y$,
    $\texttt{done}\langle z : \textbf{unit}\rangle$. **return** $\star$
  $\}$
in $f\ 0$

Increment:

send $(\texttt{get}, \star)$ **to** state **then**
recv $\texttt{state}\{(\texttt{st}, x)$.
  let $y = x + 1$ **in**
  send $(\texttt{put}, y)$ **to** state **then**
  return $y\}$

Reset t_{reset}:

send $(\texttt{put}, 0)$ **to** state **then return** $\star$

Get and reset:

send $(\texttt{get}, \star)$ **to** state **then**
let $\_ = t_{\text{reset}}$ **in**
recv $\texttt{state}\{(\texttt{st}, x)$. **return** $x\}$

Fine-grain call-by-value (Levy, Power, Thielecke 2003)

A value ***is***:

$$v, w ::= x \mid \star \mid n \mid \mathbf{true} \mid \mathbf{false} \mid (v, w) \mid \lambda x : \mathsf{b}.\, t$$

A computation ***does***:

$$\begin{aligned} t, u ::= {} & v + w \mid v < w \mid \mathbf{if}\, v\, \mathbf{then}\, t_1\, \mathbf{else}\, t_2 \\ & \mid \mathbf{match}\, v\, \mathbf{with}\, (x, y).\, t \mid v\, w \\ & \mid \mathbf{return}\, v \mid \mathbf{let}\, x = t\, \mathbf{in}\, u \qquad \text{(returning and sequencing)} \end{aligned}$$

Types

$$\mathsf{b} ::= \mathbf{unit} \mid \mathbf{int} \mid \mathbf{bool} \mid \mathsf{b} \times \mathsf{b}' \mid \mathsf{b} \to \mathsf{b}'$$

Syntax of SafeMP

Values:

$$v, w ::= x \mid \star \mid n \mid \textbf{true} \mid \textbf{false}$$

Types b $::=$ **unit** | **int** | **bool**

Computations:

$$t, u ::= v + w \mid v < w \mid \textbf{if } v \textbf{ then } t_1 \textbf{ else } t_2$$

$$\mid \textbf{let rec } f(x_1, \ldots, x_n) = t \textbf{ in } u \quad \text{(recursive function definition)}$$

$$\mid f(v_1, \ldots, v_n) \quad \text{(function application)}$$

$$\mid \textbf{return } v \mid \textbf{let } x = t \textbf{ in } u \quad \text{(returning and sequencing)}$$

Syntax of SafeMP

Values:

$$v, w ::= x \mid \star \mid n \mid \textbf{true} \mid \textbf{false}$$

Types $\mathsf{b} ::= \textbf{unit} \mid \textbf{int} \mid \textbf{bool}$

Computations:

$$t, u ::= v + w \mid v < w \mid \textbf{if } v \textbf{ then } t_1 \textbf{ else } t_2$$

$$\mid \textbf{let rec } f(x_1, \ldots, x_n) = t \textbf{ in } u \quad \text{(recursive function definition)}$$

$$\mid f(v_1, \ldots, v_n) \quad \text{(function application)}$$

$$\mid \textbf{return } v \mid \textbf{let } x = t \textbf{ in } u \quad \text{(returning and sequencing)}$$

$$\mid \textbf{send } (\ell, v) \textbf{ to } \mathsf{p} \textbf{ then } t \quad \text{(sending a message to p)}$$

$$\mid \textbf{recv } \mathsf{p} \{\ell_i\langle x_i : \mathsf{b}_i\rangle . \, t_i\}_{i \in I} \quad \text{(receiving a message from p; permitted messages are } (\ell_i, v) \text{ with } v : \mathsf{b}_i)$$

Synchronous reduction of computations

$$
\begin{array}{rll}
\text{Actions } \alpha ::= & \tau & \text{(internal action)} \\
& \mid \mathsf{p}!(\ell, v) & \text{(send message } (\ell, v) \text{ to } \mathsf{p}) \\
& \mid \mathsf{p}?(\ell, v) & \text{(receive message } (\ell, v) \text{ from } \mathsf{p})
\end{array}
$$

Reduction $\boxed{t \overset{\alpha}{\rightsquigarrow} t'}$ (selected rules)

$$\textbf{if true then } t_1 \textbf{ else } t_2 \overset{\tau}{\rightsquigarrow} t_1$$

$$\textbf{let } x = \textbf{return } v \textbf{ in } u \overset{\tau}{\rightsquigarrow} u[x \mapsto v]$$

$$\textbf{send } (\ell, v) \textbf{ to } \mathsf{p} \textbf{ then } t \overset{\mathsf{p}!(\ell, v)}{\rightsquigarrow} t$$

$$\textbf{recv } \mathsf{p} \{\ell_i \langle x_i : \mathsf{b}_i \rangle . \, t_i\}_{i \in I} \overset{\mathsf{p}?(\ell_i, v)}{\rightsquigarrow} t_i[x_i \mapsto v] \quad \text{if } v : \mathsf{b}_i$$

$$\frac{t \overset{\alpha}{\rightsquigarrow} t'}{\textbf{let } x = t \textbf{ in } u \overset{\alpha}{\rightsquigarrow} \textbf{let } x = t' \textbf{ in } u}$$

Queues and configurations

A (message) ***queue*** σ maps each p to a list of messages, and has finitely many messages in total

- $\emptyset$ is the empty queue
- $(p \lhd (\ell, v)) :: \sigma$ is σ, with (ℓ, v) added to the front of p's list
- $\sigma :: (p \lhd (\ell, v))$ is σ, with (ℓ, v) added to the back of p's list

A ***configuration*** is a closed computation with two queues:

$$(\rho, t, \sigma)$$

receive queue — computation being run — send queue

SafeMP operational semantics

Asynchronous reduction relation $(\rho, t, \sigma) \overset{\alpha}{\rightsquigarrow} (\rho', t', \sigma')$

$$(\rho, t, \sigma :: (\mathsf{p} \lhd (\ell, v))) \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} (\rho, t, \sigma)$$

$$(\rho, t, \sigma) \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} ((\mathsf{p} \lhd (\ell, v)) :: \rho, t, \sigma)$$

$$\frac{t \overset{\tau}{\rightsquigarrow} u}{(\rho, t, \sigma) \overset{\tau}{\rightsquigarrow} (\rho, u, \sigma)} \qquad \frac{t \overset{\mathsf{p}!m}{\rightsquigarrow} u}{(\rho, t, \sigma) \overset{\tau}{\rightsquigarrow} (\rho, u, (\mathsf{p} \lhd m) :: \sigma)}$$

$$\frac{t \overset{\mathsf{p}?m}{\rightsquigarrow} u}{(\rho :: (\mathsf{p} \lhd m), t, \sigma) \overset{\tau}{\rightsquigarrow} (\rho, u, \sigma)}$$

SafeMP operational semantics

$$(\emptyset, t, \emptyset) \xrightarrow{\text{state!}(\texttt{get},\star)}{}^{*} \xrightarrow{\text{state?}(\texttt{st},5)} \xrightarrow{\text{state!}(\texttt{put},0)}{}^{*} (\emptyset, \textbf{return}\ 5, \emptyset)$$

where $t =$ **send** $(\texttt{get}, \star)$ **to** state **then**
let $\_ = t_{\text{reset}}$ **in**
recv $\texttt{state}\{(\texttt{st}, x).\ \textbf{return}\ x\}$

Session types

$$\mathbb{T} ::= X \mid \mu X.\, \mathbb{T} \qquad \text{type variables and guarded recursion}$$

$$\mid \text{end} \qquad \text{inaction}$$

$$\mid \mathsf{p} \oplus \begin{cases} \ell_1\langle \mathsf{b}_1 \rangle.\, \mathbb{T}_1 \\ \vdots \\ \ell_n\langle \mathsf{b}_n \rangle.\, \mathbb{T}_n \end{cases} \qquad \text{internal choice, sending a message to } \mathsf{p}$$

$$\mid \mathsf{p}\& \begin{cases} \ell_1\langle \mathsf{b}_1 \rangle.\, \mathbb{T}_1 \\ \vdots \\ \ell_n\langle \mathsf{b}_n \rangle.\, \mathbb{T}_n \end{cases} \qquad \text{external choice, receiving a message from } \mathsf{p}$$

Example: mutable state via message-passing

Implementation of state:

let rec $f x =$
  recv client $\{$
    $\mathtt{get}\langle z : \mathbf{unit}\rangle$. **send** $(\mathtt{st}, x)$ **to** client **then** $f\ x$,
    $\mathtt{put}\langle y : \mathbf{int}\rangle$. $f\ y$,
    $\mathtt{done}\langle z : \mathbf{unit}\rangle$. **return** $\star$
  $\}$
in $f\ 0$

Session type:

$$\mu X.\ \text{client}\&\begin{cases}\mathtt{get}\langle\mathbf{unit}\rangle.\ \text{client} \oplus \mathtt{st}\langle\mathbf{int}\rangle.\ X\\ \mathtt{put}\langle\mathbf{int}\rangle.\ X\\ \mathtt{done}\langle\mathbf{unit}\rangle.\ \text{end}\end{cases}$$

Reset t_{reset}:

send $(\mathtt{put}, 0)$ **to** state **then return** $\star$

Session type: state $\oplus\ \mathtt{put}\langle\mathbf{int}\rangle$. end

Get and reset:

send $(\mathtt{get}, \star)$ **to** state **then**
let $\_ = t_{\text{reset}}$ **in**
recv $\mathtt{state}\{(\mathtt{st}, x).$ **return** $x\}$

Session type:

state $\oplus\ \mathtt{get}\langle\mathbf{unit}\rangle$.
state $\oplus\ \mathtt{put}\langle\mathbf{int}\rangle$.
state $\&\ \mathtt{st}\langle\mathbf{int}\rangle$. end

Grading, in some generality

Grades $\mathbb{T} \in \mathcal{E}$ are elements of an ordered monoid

$$(\mathcal{E}, <:, \mathbb{1}, \cdot)$$

Assign a grade to each computation:

$$\frac{t \text{ has grade } \mathbb{T} \qquad \mathbb{T} <: \mathbb{U}}{t \text{ has grade } \mathbb{U}} \qquad \frac{}{(\textbf{return } v) \text{ has grade } \mathbb{1}} \qquad \frac{t \text{ has grade } \mathbb{T} \qquad u \text{ has grade } \mathbb{U}}{(\textbf{let } x = t \textbf{ in } u) \text{ has grade } (\mathbb{T} \cdot \mathbb{U})}$$

Grading, in some generality

Grades $\mathbb{T} \in \mathcal{E}$ are elements of an ordered monoid

$$(\mathcal{E}, <:, \mathbb{1}, \cdot)$$

set of session types, over some set Θ of type variables

asynchronous subtyping

end

$$\begin{gathered}
\mathsf{end} \cdot \mathbb{U} = \mathbb{U} \\
\mathsf{X} \cdot \mathbb{U} = \mathsf{X} \\
(\mu \mathsf{X}.\, \mathbb{T}) \cdot \mathbb{U} = \mu \mathsf{X}.\, (\mathbb{T} \cdot \mathbb{U}) \\
(\mathsf{p} \oplus_{i \in I} \ell_i \langle \mathsf{b}_i \rangle.\, \mathbb{T}_i) \cdot \mathbb{U} = \mathsf{p} \oplus_{i \in I} \ell_i \langle \mathsf{b}_i \rangle.\, (\mathbb{T}_i \cdot \mathbb{U}) \\
(\mathsf{p} \& _{i \in I} \ell_i \langle \mathsf{b}_i \rangle.\, \mathbb{T}_i) \cdot \mathbb{U} = \mathsf{p} \& _{i \in I} \ell_i \langle \mathsf{b}_i \rangle.\, (\mathbb{T}_i \cdot \mathbb{U})
\end{gathered}$$

Assign a grade to each computation:

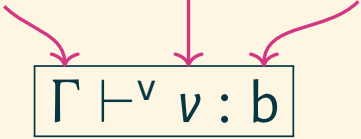
$$\frac{t \text{ has grade } \mathbb{T} \qquad \mathbb{T} <: \mathbb{U}}{t \text{ has grade } \mathbb{U}} \qquad \frac{}{(\mathbf{return}\ v) \text{ has grade } \mathbb{1}} \qquad \frac{t \text{ has grade } \mathbb{T} \qquad u \text{ has grade } \mathbb{U}}{(\mathbf{let}\ x = t\ \mathbf{in}\ u) \text{ has grade } (\mathbb{T} \cdot \mathbb{U})}$$

Typing SafeMP: values

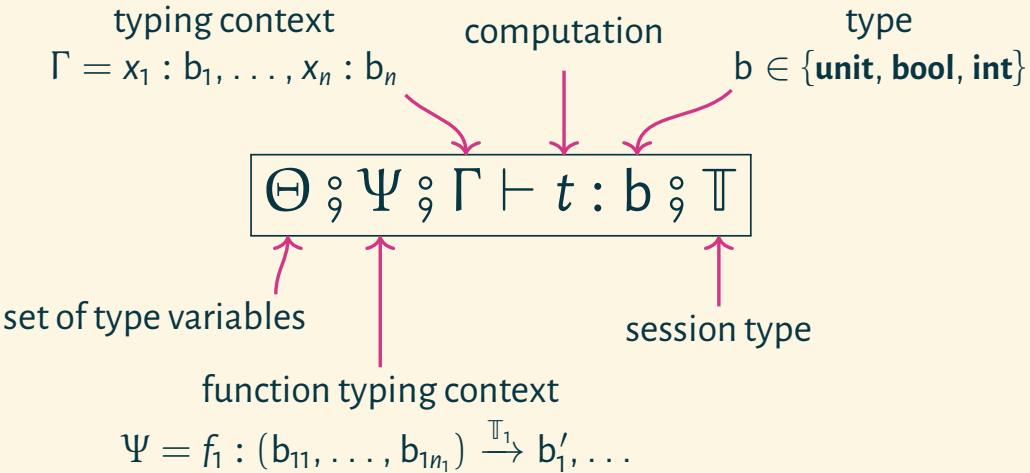
typing context
$\Gamma = x_1 : b_1, \ldots, x_n : b_n$

value

type
$b \in \{\textbf{unit}, \textbf{bool}, \textbf{int}\}$



Typing SafeMP: computations



Typing SafeMP: computations

Computation typing $\boxed{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash t : \mathsf{b} \fatsemi \mathbb{T}}$ (selected rules):

$$\frac{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash t : \mathsf{b} \fatsemi \mathbb{T} \quad \mathbb{T} <: \mathbb{U}}{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash t : \mathsf{b} \fatsemi \mathbb{U}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} v : \mathsf{b}}{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash \mathbf{return}\, v : \mathsf{b} \fatsemi \mathsf{end}}$$

$$\frac{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash t : \mathsf{b} \fatsemi \mathbb{T} \quad \Theta \fatsemi \Psi \fatsemi \Gamma, x : \mathsf{b} \vdash u : \mathsf{b}' \fatsemi \mathbb{U}}{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash \mathbf{let}\, x = t\, \mathbf{in}\, u : \mathsf{b}' \fatsemi \mathbb{T} \cdot \mathbb{U}}$$

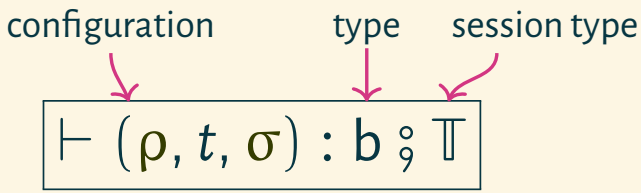
Typing SafeMP: computations

Computation typing $\boxed{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash t : \mathsf{b} \fatsemi \mathbb{T}}$ (selected rules):

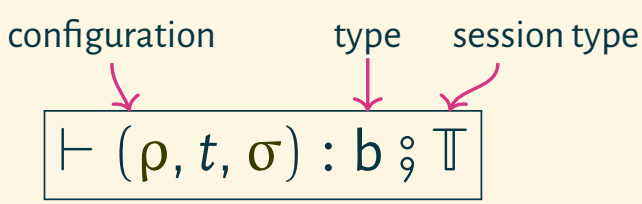
$$\frac{\Gamma \vdash^{\mathsf{v}} v : \mathsf{b} \quad \Theta \fatsemi \Psi \fatsemi \Gamma \vdash t : \mathsf{b}' \fatsemi \mathbb{T}}{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash \mathbf{send}\,(\ell, v)\,\mathbf{to}\,\mathsf{p}\,\mathbf{then}\,t : \mathsf{b}' \fatsemi (\mathsf{p} \oplus \ell\langle \mathsf{b} \rangle.\, \mathbb{T})}$$

$$\frac{\Theta \fatsemi \Psi \fatsemi \Gamma, x_i : \mathsf{b}_i \vdash t_i : \mathsf{b}' \fatsemi \mathbb{T}_i \text{ for each } i \in I}{\Theta \fatsemi \Psi \fatsemi \Gamma \vdash \mathbf{recv}\,\mathsf{p}\,\{\ell_i\langle x_i \rangle.\, t_i\}_{i \in I} : \mathsf{b}' \fatsemi (\mathsf{p} \mathbin{\&}_{i \in I} \ell_i\langle \mathsf{b}_i \rangle.\, \mathbb{T}_i)}$$

Typing SafeMP: configurations

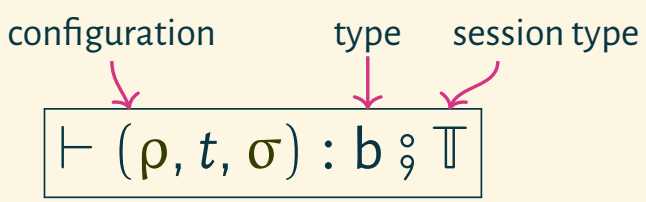


Typing SafeMP: configurations



$$\frac{\cdot \fatsemi \cdot \fatsemi \cdot \vdash t : \mathsf{b} \fatsemi \mathbb{T}}{\vdash (\emptyset, t, \emptyset) : \mathsf{b} \fatsemi \mathbb{T}}$$

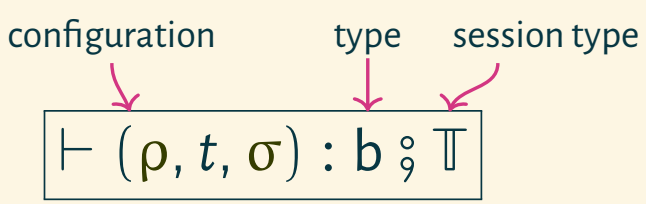
Typing SafeMP: configurations



$$\frac{v : \mathsf{b}' \quad \mathbb{T} \xhookrightarrow{\mathsf{p}?\ell\langle \mathsf{b}' \rangle} \mathbb{U} \quad \vdash (\rho, t, \sigma) : \mathsf{b} \fatsemi \mathbb{T}}{\vdash ((\mathsf{p} \triangleleft (\ell, v)) :: \rho, t, \sigma) : \mathsf{b} \fatsemi \mathbb{U}}$$

(ρ, t, σ) is *required* to accept message (ℓ, v) from p, then continue as $\mathbb{U}$, by $\boxed{\mathbb{T} \xhookrightarrow{\mathsf{p}?\ell\langle \mathsf{b}' \rangle} \mathbb{U}}$ (recall the reduction $(\rho, t, \sigma) \overset{\mathsf{p}?m}{\rightsquigarrow} ((\mathsf{p} \triangleleft m) :: \rho, t, \sigma)$)

Typing SafeMP: configurations



$$\frac{v : \mathsf{b}' \quad \mathbb{U} \xhookrightarrow{\mathsf{p}!\ell\langle \mathsf{b}' \rangle} \mathbb{T} \quad \vdash (\rho, t, \sigma) : \mathsf{b} \fatsemi \mathbb{T}}{\vdash (\rho, t, \sigma :: (\mathsf{p} \lhd (\ell, v))) : \mathsf{b} \fatsemi \mathbb{U}}$$

$(\rho, t, \sigma :: (\mathsf{p} \lhd (\ell, v)))$ is *permitted* to send (ℓ, v) to p, then continue as $\mathbb{T}$, by $\boxed{\mathbb{U} \xhookrightarrow{\mathsf{p}!\ell\langle \mathsf{b}' \rangle} \mathbb{T}}$ (recall the reduction $(\rho, t, \sigma :: (\mathsf{p} \lhd m)) \overset{\mathsf{p}!m}{\rightsquigarrow} (\rho, t, \sigma)$)

Subject reduction

Assume $\vdash (\rho, t, \sigma) : \mathsf{b} \fatsemi \mathbb{T}$, and $(\rho, t, \sigma) \overset{\alpha}{\rightsquigarrow} (\rho', t', \sigma')$.

1. If $\alpha = \tau$, then $\vdash (\rho', t', \sigma') : \mathsf{b} \fatsemi \mathbb{T}$.
2. If $\alpha = \mathsf{p}!(\ell, v)$ with $v : \mathsf{b}'$, then $\vdash (\rho', t', \sigma') : \mathsf{b} \fatsemi \mathbb{U}$ for some $\mathbb{T} \xhookrightarrow{\mathsf{p}!\ell\langle \mathsf{b}' \rangle} \mathbb{U}$.
3. If $\alpha = \mathsf{p}?(\ell, v)$ with $v : \mathsf{b}'$, then $\vdash (\rho', t', \sigma') : \mathsf{b} \fatsemi \mathbb{U}$ for every $\mathbb{T} \xhookrightarrow{\mathsf{p}?\ell\langle \mathsf{b}' \rangle} \mathbb{U}$.

Subject reduction

$$(\emptyset, t, \emptyset) \xrightarrow{\text{state!}(\texttt{get},\star)}{}^{*} \xrightarrow{\text{state?}(\texttt{st},5)} \xrightarrow{\text{state!}(\texttt{put},0)}{}^{*} (\emptyset, \textbf{return}\ 5, \emptyset)$$

$$\begin{array}{l}\text{state} \oplus \texttt{get}\langle\textbf{unit}\rangle.\\ \text{state} \oplus \texttt{put}\langle\textbf{int}\rangle.\\ \text{state} \,\&\, \texttt{st}\langle\textbf{int}\rangle.\ \text{end}\end{array} \xrightarrow{\text{state!}\texttt{get}\langle\textbf{unit}\rangle} \begin{array}{l}\text{state} \oplus \texttt{put}\langle\textbf{int}\rangle.\\ \text{state} \,\&\, \texttt{st}\langle\textbf{int}\rangle.\ \text{end}\end{array} \xrightarrow{\text{state?}\texttt{st}\langle\textbf{int}\rangle} \begin{array}{l}\text{state} \oplus \texttt{put}\langle\textbf{int}\rangle.\\ \text{end}\end{array} \xrightarrow{\text{state!}\texttt{put}\langle\textbf{int}\rangle} \text{end}$$

$$\begin{array}{rl}\text{where } t = & \textbf{send}\ (\texttt{get}, \star)\ \textbf{to}\ \text{state}\ \textbf{then}\\ & \textbf{let}\ \_ = t_{\text{reset}}\ \textbf{in}\\ & \textbf{recv}\ \texttt{state}\{(\texttt{st}, x).\ \textbf{return}\ x\}\end{array}$$

Sessions

Configurations running in parallel:

$$(\mathsf{r}_1 \lhd (\rho_1, t_1, \sigma_1), \ldots, \mathsf{r}_n \lhd (\rho_n, t_n, \sigma_n))$$

Sessions

Configurations running in parallel:

$$(\mathsf{r}_1 \lhd (\rho_1, t_1, \sigma_1), \ldots, \mathsf{r}_n \lhd (\rho_n, t_n, \sigma_n))$$

Reduction

$(\mathsf{r}_1 \lhd (\rho_1, t_1, \sigma_1), \ldots, \mathsf{r}_n \lhd (\rho_n, t_n, \sigma_n)) \rightsquigarrow (\mathsf{r}_1 \lhd (\rho_1', t_1', \sigma_1'), \ldots, \mathsf{r}_n \lhd (\rho_n', t_n', \sigma_n'))$
is one of

- internal: $(\rho_j, t_j, \sigma_j) \overset{\tau}{\rightsquigarrow} (\rho_j', t_j', \sigma_j')$ and $(\rho_i, t_i, \sigma_i) = (\rho_i', t_i', \sigma_i')$ for all $i \neq j$.
- communication: $(\rho_k, t_k, \sigma_k) \overset{\mathsf{r}_j!(\ell,v)}{\rightsquigarrow} (\rho_k', t_k', \sigma_k')$ and $(\rho_j, t_j, \sigma_j) \overset{\mathsf{r}_k?(\ell,v)}{\rightsquigarrow} (\rho_j', t_j', \sigma_j')$ and $(\rho_i, t_i, \sigma_i) = (\rho_i', t_i', \sigma_i')$ for all $i \neq j$.

Sessions

If $\mathbb{G}$ is a global type, and

$$\vdash (\rho_i, t_i, \sigma_i) : \mathsf{b} \parr (\mathbb{G} \upharpoonright \mathsf{r}_i) \quad \text{for each } i$$

then

$$(\mathsf{r}_1 \lhd (\rho_1, t_1, \sigma_1), \ldots, \mathsf{r}_n \lhd (\rho_n, t_n, \sigma_n))$$

is deadlock-free and live.