

Multiparty session-typed message passing, as a computational effect

Dylan McDermott

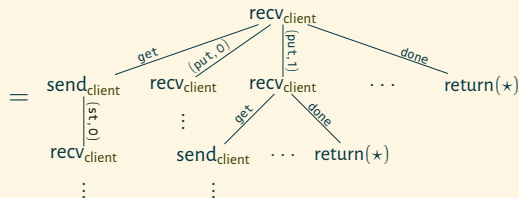
Based on joint work with Nobuko Yoshida

✉ dylan@dylanm.org

What this work is about

Extensional denotational semantics of *asynchronous message passing* with *multiparty session types*

```
let rec fx =  
  recvclient {  
    get⟨_ : unit⟩. send(st, x) to client then fx,  
    put⟨y : int⟩. fy,  
    done⟨_ : unit⟩. return ★  
  }  
in f 0
```



Approach: message-passing as a computational effect

Computational effect

e.g. {
mutable state
exceptions
nondeterminism
message-passing

$\approx$

something observable a computation can do, beyond producing a result

We have a big computational effects toolbox, including:

- Algebraic techniques for denotational semantics
- **(Effect) grading**: static analysis of computational effects

Example (new here!): multiparty **session types** as grades, used to enforce safety and liveness properties

Approach: message-passing as a computational effect

Computational effect

e.g. {
mutable state
exceptions
nondeterminism
message-passing

$\approx$

something observable a computation can do, beyond producing a result

We have a big computational effects toolbox, including:

- Algebraic techniques for denotational semantics

- **(Effect) grading**: static analysis of c

Example (new here!): multiparty s
enforce safety and liveness proper

Receive-liveness: if someone is waiting to receive a message, then it eventually will receive that message (assuming fair scheduling)

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
- An extensional denotational semantics, constructed using algebraic techniques

Example: mutable state via message-passing

Implementation of **state**:

```
let rec fx =  
  recv client {  
    get⟨z : unit⟩. send (st, x) to client then f x,  
    put⟨y : int⟩. f y,  
    done⟨z : unit⟩. return ★  
  }  
in f 0
```

Increment:

```
send (get, ★) to state then  
recv state {(st, x).  
  let y = x + 1 in  
  send (put, y) to state then  
  return y}
```

Reset t_{reset} :

```
send (put, 0) to state then return ★
```

Get and reset:

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

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

A value *is*:

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

A computation *does*:

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

(Value) types

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

Syntax of SafeMP

Values:

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

(Value) types $b ::= \mathbf{unit} \mid \mathbf{int} \mid \mathbf{bool}$

Computations:

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

$\mid \mathbf{let } \mathbf{rec } f(x_1, \dots, x_n) = t \mathbf{ in } u$ (recursive function definition)

$\mid f(v_1, \dots, v_n)$ (function application)

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

Syntax of SafeMP

Values:

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

(Value) types $b ::= \mathbf{unit} \mid \mathbf{int} \mid \mathbf{bool}$

Computations:

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

$\mid \mathbf{let } \mathbf{rec } f(x_1, \dots, x_n) = t \mathbf{ in } u$ (recursive function definition)

$\mid f(v_1, \dots, v_n)$ (function application)

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

$\mid \mathbf{send } (\ell, v) \mathbf{ to } p \mathbf{ then } t$ (sending a message to p)

$\mid \mathbf{recv } p \{ \ell_i \langle x_i : b_i \rangle . t_i \}_{i \in I}$ (receiving a message from p ;
permitted messages are (ℓ_i, v) with $v : b_i$)

Example: mutable state via message-passing

Implementation of **state**:

```
let rec fx =  
  recv client {  
    get⟨z : unit⟩. send (st, x) to client then f x,  
    put⟨y : int⟩. f y,  
    done⟨z : unit⟩. return ★  
  }  
in f 0
```

Increment:

```
send (get, ★) to state then  
recv state {(st, x).  
  let y = x + 1 in  
  send (put, y) to state then  
  return y}
```

Reset t_{reset} :

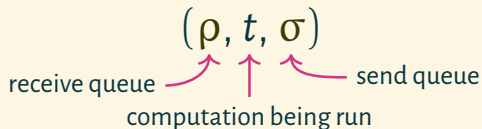
```
send (put, 0) to state then return ★
```

Get and reset:

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

Configurations and operational semantics

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



Asynchronous reduction relation

$$(\rho, t, \sigma) \xrightarrow{\alpha} (\rho', t', \sigma')$$

labelled by **actions**

$\alpha ::= \tau$	(internal action)
$ \mathbf{p}!(\ell, v)$	(send message (ℓ, v) to $\mathbf{p}$)
$ \mathbf{p}?(\ell, v)$	(receive message (ℓ, v) from $\mathbf{p}$)

Typing SafeMP: multiparty session types

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

$\mid \text{end} \mid \mathbf{p} \oplus \begin{cases} \ell_1 \langle \mathbf{b}_1 \rangle. \mathbb{T}_1 \\ \vdots \\ \ell_n \langle \mathbf{b}_n \rangle. \mathbb{T}_n \end{cases} \mid \mathbf{p} \& \begin{cases} \ell_1 \langle \mathbf{b}_1 \rangle. \mathbb{T}_1 \\ \vdots \\ \ell_n \langle \mathbf{b}_n \rangle. \mathbb{T}_n \end{cases}$

inaction

internal choice,
sending a message to $\mathbf{p}$

external choice,
receiving a message from $\mathbf{p}$

Example: mutable state via message-passing

SafeMP computation t_{state} :

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

Session type $\mathbb{T}_{\text{state}}$:

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

Typing SafeMP: configurations

Typing relation

$$\vdash (\rho, t, \sigma) : b \circ \mathbb{T}$$

- Defined using a *graded* computation typing relation: session types are grades
- Has an admissible subtyping rule

$$\frac{\vdash (\rho, t, \sigma) : b \circ \mathbb{T} \quad \mathbb{T} <: \mathbb{U}}{\vdash (\rho, t, \sigma) : b \circ \mathbb{U}}$$

reformulation of Ghilezan et al.'s sound and complete *asynchronous* subtyping

- Satisfies a subject reduction theorem for $(\rho, t, \sigma) \xrightarrow{\alpha} (\rho', t', \sigma')$

Typing SafeMP: grading

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

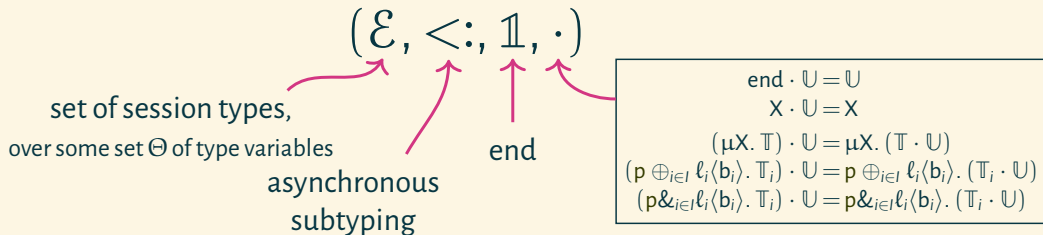
$$(\mathcal{E}, <:, \mathbb{1}, \cdot)$$

Assign a grade to each computation:

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

Typing SafeMP: grading

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



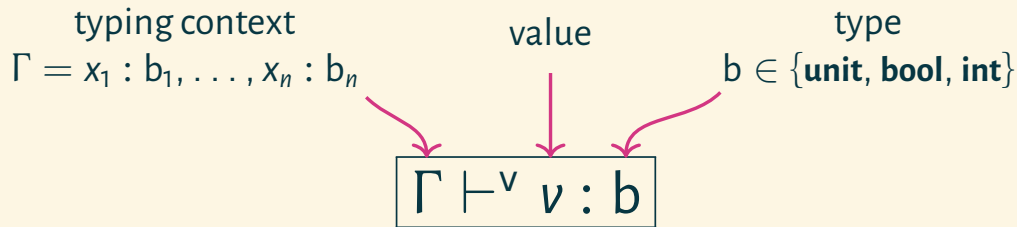
Assign a grade to each computation:

$$\frac{t \text{ has grade } \mathbb{T} \quad \mathbb{T} <: \mathbb{U}}{t \text{ has grade } \mathbb{U}}$$

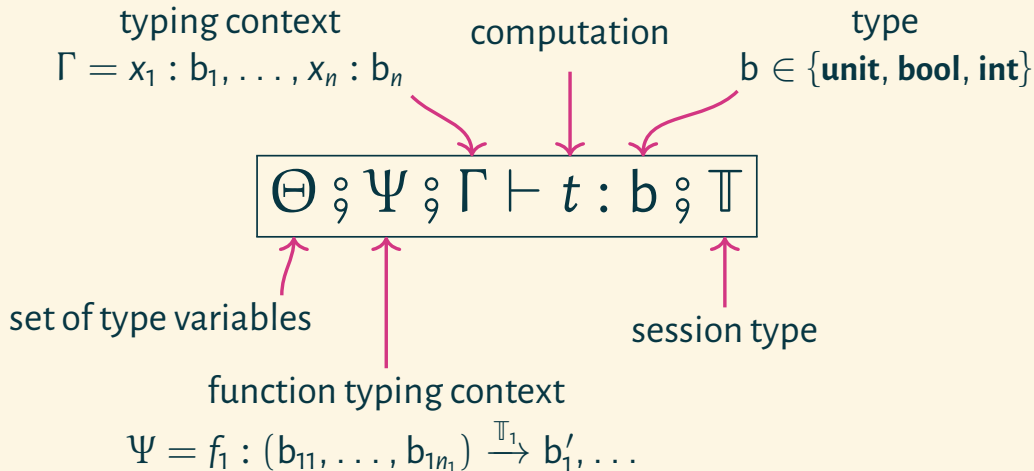
$$\frac{}{(\mathbf{return} \ v) \text{ has grade } \mathbb{1}}$$

$$\frac{t \text{ has grade } \mathbb{T} \quad u \text{ has grade } \mathbb{U}}{(\mathbf{let} \ x = t \ \mathbf{in} \ u) \text{ has grade } (\mathbb{T} \cdot \mathbb{U})}$$

Typing SafeMP: values



Typing SafeMP: computations



Typing SafeMP: computations

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

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

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

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

Typing SafeMP: computations

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

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

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

Example: mutable state via message-passing

SafeMP computation t_{state} :

```
let rec f x =  
  recv client {  
    get ⟨_ : unit⟩. send (st, x) to client then f x,  
    put ⟨y : int⟩. f y,  
    done ⟨_ : unit⟩. return ★  
  }  
in f 0
```

Session type $\mathbb{T}_{\text{state}}$:

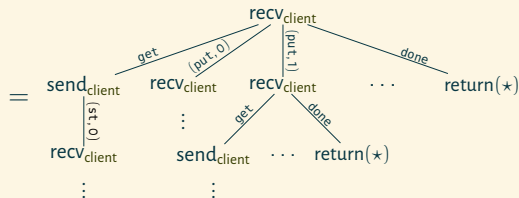
$$\mu X. \text{client} \& \begin{cases} \text{get} \langle \text{unit} \rangle. \text{client} \oplus \text{st} \langle \text{int} \rangle. X \\ \text{put} \langle \text{int} \rangle. X \\ \text{done} \langle \text{unit} \rangle. \text{end} \end{cases}$$

$$\vdash (\emptyset, t_{\text{state}}, \emptyset) : \text{unit} \circ \mathbb{T}_{\text{state}}$$

What this work is about

Extensional denotational semantics of *asynchronous message passing* with *multiparty session types*

```
let rec fx =  
  recvclient {  
    get⟨_ : unit⟩. send(st, x) to client then fx,  
    put⟨y : int⟩. fy,  
    done⟨_ : unit⟩. return ★  
  }  
in f 0
```



Denotational semantics

The usual structure (since (Katsumata 2012)) for modelling graded effects is a *graded monad* $\mathcal{N}$:

$\mathcal{N}(X)_{\mathbb{T}}$	set containing interpretations of computations that return elements of X , and have grade $\mathbb{T}$
return	unit functions; to interpret return
$\gg=$	bind functions; to interpret sequencing let $x = t$ in u
$\mathcal{N}(X)_{\mathbb{T} <: \mathbb{U}}$	interpretation of subgrading (asynchronous subtyping)

We construct a session-type-graded monad that models message-passing

Computation trees

Generated *coinductively* by

$$\begin{array}{l} t ::= \text{return}(x) \\ \quad | \text{send}_{p,m}(t) \\ \quad | \text{recv}_p(t_m)_{m \in M} \end{array}$$

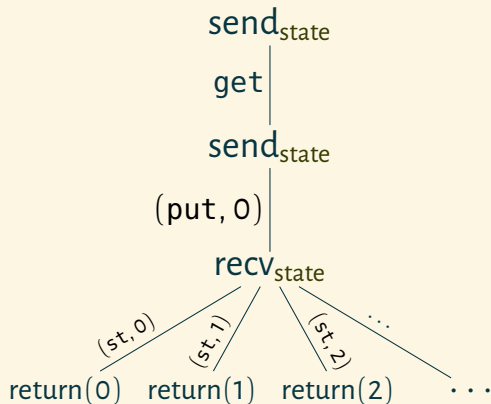
m : message (ℓ, v)

M : set of messages

p : participant

x : result; element of a given set X

Computation tree $\llbracket t_{\text{client}} \rrbracket$:



Relating computation trees to session types

Two key coinductively defined relations:

- **Typing** relation $t \blacktriangleleft \mathbb{T}$:

“tree t implements protocol $\mathbb{T}$ ”

Respects asynchronous subtyping: if $\mathbb{T} <: \mathbb{U}$, then $t \blacktriangleleft \mathbb{T}$ implies $t \blacktriangleleft \mathbb{U}$

- **Typed bisimulation** relation $t \sim_{\mathbb{T}} u$:

“trees t and u are equivalent implementations of the protocol $\mathbb{T}$ ”

Implies $t, u \blacktriangleleft \mathbb{T}$

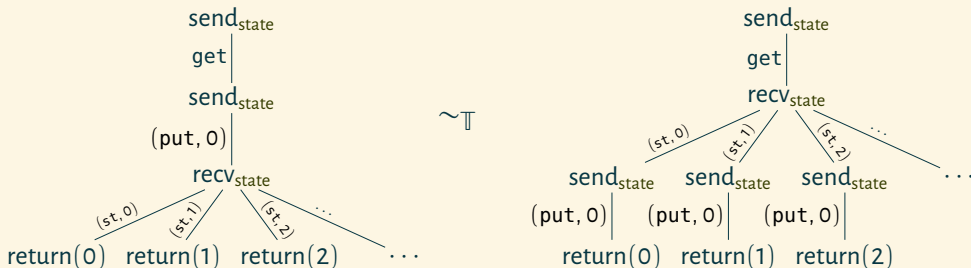
Respects asynchronous subtyping: if $\mathbb{T} <: \mathbb{U}$, then $t \sim_{\mathbb{T}} u$ implies $t \sim_{\mathbb{U}} u$

Relating computation trees to session types

Two key coinductively defined relations:

- **Typing** relation $t \blacktriangleleft \mathbb{T}$
- **Typed bisimulation** relation $t \sim_{\mathbb{T}} u$

Example with $\mathbb{T} = \text{state} \oplus \text{get} \langle \text{unit} \rangle . \text{state} \& \text{st} \langle \text{int} \rangle . \text{state} \oplus \text{put} \langle \text{int} \rangle . \text{end} :$



Session-type graded monad for asynchronous message-passing

$\mathcal{N}(X)_{\mathbb{T}}$ = set of $\sim_{\mathbb{T}}$ -*normal forms* of computation trees $t \blacktriangleleft \mathbb{T}$

By more-or-less standard definitions:

- interpret each well-typed closed SafeMP computation $\vdash t : b \circ \mathbb{T}$ as a computation tree $\llbracket t \rrbracket \in \mathcal{N}(X)_{\mathbb{T}}$
- interpret each well-typed configuration $\vdash (\rho, t, \sigma) : b \circ \mathbb{T}$ as a computation tree $\llbracket (\rho, t, \sigma) \rrbracket \in \mathcal{N}[\![b]\!]_{\mathbb{T}}$

Main theorem: adequacy

If $\vdash (\rho, t, \sigma), (\rho', t', \sigma') : b \circ \mathbb{T}$ then

$$(\rho, t, \sigma) \sim_{\mathbb{T}} (\rho', t', \sigma') \quad \Leftrightarrow \quad \llbracket (\rho, t, \sigma) \rrbracket = \llbracket (\rho', t', \sigma') \rrbracket$$

The interpretation $\llbracket - \rrbracket$ captures exactly the observable (up to $\sim_{\mathbb{T}}$) behaviour of SafeMP

Use this to:

- test whether two SafeMP implementations are equivalent;
- verify **deadlock-freedom** and **liveness** properties for SafeMP (or any other calculus with an adequate interpretation using $\mathcal{N}$)

Denotational reasoning for asynchronous multiparty session types

Dylan McDermott  and Nobuko Yoshida 

- Multiparty session-typed message-passing as a graded computational effect,
- enabling us to construct an extensional denotational semantics,
- which we can use to reason about programs with asynchronous message-passing

Subject reduction

Assume $\vdash (\rho, t, \sigma) : b \circledast \mathbb{T}$, and $(\rho, t, \sigma) \xrightarrow{\alpha} (\rho', t', \sigma')$.

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

Syntax of SafeMP

Syntax based on ***fine-grain call-by-value*** (Levy, Power, Thielecke 2003)

Computations:

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

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

$\mid f(v_1, \dots, v_n)$ (function application)

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

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

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

v : value
 b : type
 ℓ : message label
 p : participant

Sessions

Configurations running in parallel:

$$(r_1 \triangleleft (\rho_1, t_1, \sigma_1), \dots, r_n \triangleleft (\rho_n, t_n, \sigma_n))$$

If $\mathbb{G} \upharpoonright r_i$ are projections from a global type, and

$$\vdash (\rho_i, t_i, \sigma_i) : b \circ (\mathbb{G} \upharpoonright r_i) \quad \text{for each } i$$

then we have **safety**, **deadlock-freedom** and **liveness** under fair scheduling.

Synchronous reduction of computations

Actions $\alpha ::= \tau$ (internal action)
 $\mid \mathbf{p}!(\ell, v)$ (send message (ℓ, v) to $\mathbf{p}$)
 $\mid \mathbf{p}?(\ell, v)$ (receive message (ℓ, v) from $\mathbf{p}$)

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

if true then t_1 else t_2	$\xrightarrow{\tau}$	t_1	
let $x = \mathbf{return } v$ in u	$\xrightarrow{\tau}$	$u[x \mapsto v]$	
send (ℓ, v) to $\mathbf{p}$ then t	$\xrightarrow{\mathbf{p}!(\ell, v)}$	t	
recv $\mathbf{p} \{ \ell_i \langle x_i : b_i \rangle . t_i \}_{i \in I}$	$\xrightarrow{\mathbf{p}?(\ell_i, v)}$	$t_i[x_i \mapsto v]$	if $v : b_i$

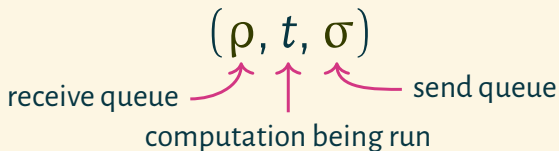
	$\xrightarrow{\alpha}$	t'	
let $x = t$ in u	$\xrightarrow{\alpha}$	let $x = t'$ in u	

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

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



SafeMP operational semantics

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

$$(\rho, t, \sigma :: (\mathbf{p} \triangleleft (\ell, \nu))) \xrightarrow{\mathbf{p}!(\ell, \nu)} (\rho, t, \sigma)$$

$$(\rho, t, \sigma) \xrightarrow{\mathbf{p}?(\ell, \nu)} ((\mathbf{p} \triangleleft (\ell, \nu)) :: \rho, t, \sigma)$$

$$\frac{t \xrightarrow{\tau} u}{(\rho, t, \sigma) \xrightarrow{\tau} (\rho, u, \sigma)} \quad \frac{t \xrightarrow{\mathbf{p}!m} u}{(\rho, t, \sigma) \xrightarrow{\tau} (\rho, u, (\mathbf{p} \triangleleft m) :: \sigma)}$$

$$\frac{t \xrightarrow{\mathbf{p}?m} u}{(\rho :: (\mathbf{p} \triangleleft m), t, \sigma) \xrightarrow{\tau} (\rho, u, \sigma)}$$

SafeMP operational semantics

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

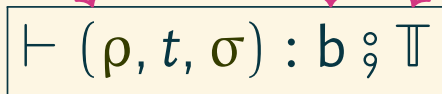
where $t =$ **send** (get, $\star$) **to** state **then**
 let $_ = t_{\text{reset}}$ **in**
 recv state $\{(\text{st}, x). \text{return } x\}$

Typing SafeMP: configurations

configuration

type

session type

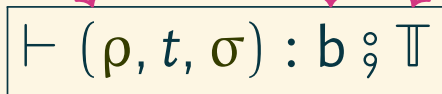

$$\vdash (\rho, t, \sigma) : b \circ \mathbb{T}$$

Typing SafeMP: configurations

configuration

type

session type


$$\vdash (\rho, t, \sigma) : b \mathbin{\text{\textcircled{;}}} \mathbb{T}$$

$$\frac{\cdot \mathbin{\text{\textcircled{;}}} \cdot \mathbin{\text{\textcircled{;}}} \cdot \vdash t : b \mathbin{\text{\textcircled{;}}} \mathbb{T}}{\vdash (\emptyset, t, \emptyset) : b \mathbin{\text{\textcircled{;}}} \mathbb{T}}$$

Typing SafeMP: configurations

configuration type session type

$$\boxed{\vdash (\rho, t, \sigma) : b \circ \mathbb{T}}$$

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

(ρ, t, σ) is *required* to accept message (ℓ, v) from p , then continue as $\mathbb{U}$,

by $\boxed{\mathbb{T} \xrightarrow{p?l\langle b' \rangle} \mathbb{U}}$ (recall the reduction $(\rho, t, \sigma) \xrightarrow{p?m} ((p \triangleleft m) :: \rho, t, \sigma))$

Typing SafeMP: configurations

configuration type session type

$$\boxed{\vdash (\rho, t, \sigma) : b \mathbin{\text{\texttt{;}}} \mathbb{T}}$$

$$\frac{v : b' \quad \mathbb{U} \xrightarrow{p! \ell \langle b' \rangle} \mathbb{T} \quad \vdash (\rho, t, \sigma) : b \mathbin{\text{\texttt{;}}} \mathbb{T}}{\vdash (\rho, t, \sigma :: (p \triangleleft (\ell, v))) : b \mathbin{\text{\texttt{;}}} \mathbb{U}}$$

$(\rho, t, \sigma :: (p \triangleleft (\ell, v)))$ is *permitted* to send (ℓ, v) to p , then continue as $\mathbb{T}$,

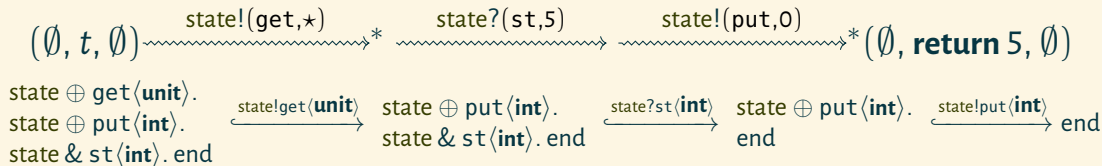
by $\boxed{\mathbb{U} \xrightarrow{p! \ell \langle b' \rangle} \mathbb{T}}$ (recall the reduction $(\rho, t, \sigma :: (p \triangleleft m)) \xrightarrow{p!m} (\rho, t, \sigma)$)

Subject reduction

Assume $\vdash (\rho, t, \sigma) : b \circledast \mathbb{T}$, and $(\rho, t, \sigma) \xrightarrow{\alpha} (\rho', t', \sigma')$.

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

Subject reduction



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

Sessions

Configurations running in parallel:

$$(r_1 \triangleleft (\rho_1, t_1, \sigma_1), \dots, r_n \triangleleft (\rho_n, t_n, \sigma_n))$$

Sessions

Configurations running in parallel:

$$(r_1 \triangleleft (\rho_1, t_1, \sigma_1), \dots, r_n \triangleleft (\rho_n, t_n, \sigma_n))$$

Reduction

$(r_1 \triangleleft (\rho_1, t_1, \sigma_1), \dots, r_n \triangleleft (\rho_n, t_n, \sigma_n)) \rightsquigarrow (r_1 \triangleleft (\rho'_1, t'_1, \sigma'_1), \dots, r_n \triangleleft (\rho'_n, t'_n, \sigma'_n))$
is one of

- internal: $(\rho_j, t_j, \sigma_j) \xrightarrow{\tau} (\rho'_j, t'_j, \sigma'_j)$ and $(\rho_i, t_i, \sigma_i) = (\rho'_i, t'_i, \sigma'_i)$ for all $i \neq j$.
- communication: $(\rho_k, t_k, \sigma_k) \xrightarrow{r_j!(\ell, \nu)} (\rho'_k, t'_k, \sigma'_k)$ and $(\rho_j, t_j, \sigma_j) \xrightarrow{r_k?(\ell, \nu)} (\rho'_j, t'_j, \sigma'_j)$ and $(\rho_i, t_i, \sigma_i) = (\rho'_i, t'_i, \sigma'_i)$ for all $i \notin j, k$.

Sessions

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

$$\vdash (\rho_i, t_i, \sigma_i) : \mathbf{b} \circ (\mathbb{G} \upharpoonright r_i) \quad \text{for each } i$$

then

$$(r_1 \triangleleft (\rho_1, t_1, \sigma_1), \dots, r_n \triangleleft (\rho_n, t_n, \sigma_n))$$

is deadlock-free and live.

Related work

- Dominic Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects.
Encoding of session-typed π -calculus in effect-typed λ -calculus
- Castellani et al. Global types and event structure semantics for asynchronous multiparty sessions.
Intensional semantics of session-types, using event structures
- Castellan and Yoshida. Two Sides of the Same Coin: Session Types and Game Semantics
Strategies as a model of session calculus
- Dimitrios Kouzapas and Nobuko Yoshida. Globally governed session semantics.
Bisimulations controlled by global types