# Denotational reasoning for asynchronous multiparty session types

**Dylan McDermott** Nobuko Yoshida

University of Oxford

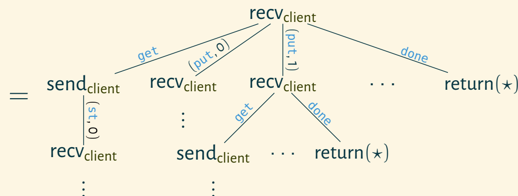
Funded by Project TaRDIS (Horizon Europe), and UKRI

dylan@dylanm.org

# What this work is about

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

$$\left[\!\left[\begin{array}{l}\textbf{let rec}\ f x = \\ \quad \textbf{recv}\ \text{client}\,\{ \\ \quad\quad \texttt{get}\langle _ : \textbf{unit}\rangle.\ \textbf{send}\ (\texttt{st}, x)\ \textbf{to}\ \text{client}\ \textbf{then}\ f\ x, \\ \quad\quad \texttt{put}\langle y : \textbf{int}\rangle.\ f\ y, \\ \quad\quad \texttt{done}\langle _ : \textbf{unit}\rangle.\ \textbf{return}\ \star \\ \quad \} \\ \textbf{in}\ f\ 0 \end{array}\right]\!\right] =$$



# Approach: message-passing is 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): ***session types*** as grades, used to enforce safety and liveness properties

# Approach: message-passing is 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): ***session types***

  and liveness properties

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

# SafeMP: a call-by-value message-passing calculus

- Syntax based on ***fine-grain call-by-value*** (Levy, Power, Thielecke 2003)
- Message-passing is asynchronous (buffer messages into queues)

Two example computations:

state implements some memory, acting as a server for client

$t_{\text{state}}$:

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

$t_{\text{client}}$:

$$
\begin{array}{l}
\textbf{send } (\texttt{get}, \star) \textbf{ to } \text{state} \textbf{ then} \\
\textbf{send } (\texttt{put}, 0) \textbf{ to } \text{state} \textbf{ then} \\
\textbf{recv } \texttt{state} \{ (\texttt{st}, x) .\ \textbf{return } x \}
\end{array}
$$

# `SafeMP`: a call-by-value message-passing calculus

- Syntax based on ***fine-grain call-by-value*** (Levy, Power, Thielecke 2003)
- Message-passing is asynchronous (buffer messages into queues)
- Has a graded type system: grades are multiparty session types, with asynchronous subtyping
- Has an extensional denotational semantics, constructed using algebraic techniques
- Deadlock-freedom and liveness properties, provided by the type system, proved using the denotations

# Multiparty session types (Honda et al.)

Describe protocols followed by message-passing computations:

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

$$\mid \text{end} \mid p \oplus \begin{cases} \ell_1\langle b_1\rangle.\,\mathbb{T}_1 \\ \vdots \\ \ell_n\langle b_n\rangle.\,\mathbb{T}_n \end{cases} \mid p\& \begin{cases} \ell_1\langle b_1\rangle.\,\mathbb{T}_1 \\ \vdots \\ \ell_n\langle b_n\rangle.\,\mathbb{T}_n \end{cases}$$

inaction — internal choice, sending a message to p — external choice, receiving a message from p

$\mathbb{T}$ : session type
b : (value) type
$\ell$ : message label
p : participant (e.g. state)

# Example: mutable state via message-passing

SafeMP computation $t_{\text{state}}$:

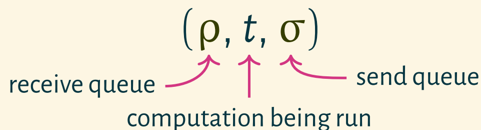
$$\begin{array}{l}\textbf{let rec } f x = \\ \quad \textbf{recv } \text{client} \{ \\ \quad\quad \texttt{get}\langle _ : \textbf{unit} \rangle .\ \textbf{send } (\texttt{st}, x) \textbf{ to } \text{client } \textbf{then } f\ x, \\ \quad\quad \texttt{put}\langle y : \textbf{int} \rangle .\ f\ y, \\ \quad\quad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

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

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

# Typing SafeMP: configurations

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




Typing relation $\boxed{\vdash (\rho, t, \sigma) : \mathsf{b} \mathbin{;} \mathbb{T}}$:

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

$$\frac{\vdash (\rho, t, \sigma) : \mathsf{b} \mathbin{;} \mathbb{T} \qquad \mathbb{T} <: \mathbb{U}}{\vdash (\rho, t, \sigma) : \mathsf{b} \mathbin{;} \mathbb{U}}$$

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

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

# Denotational semantics

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

| | |
|---|---|
| $\boxed{\mathcal{N}(X)_{\mathbb{T}}}$ | set containing interpretations of computations that return elements of $X$, and have grade $\mathbb{T}$ |
| $\mathrm{return}$ | 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

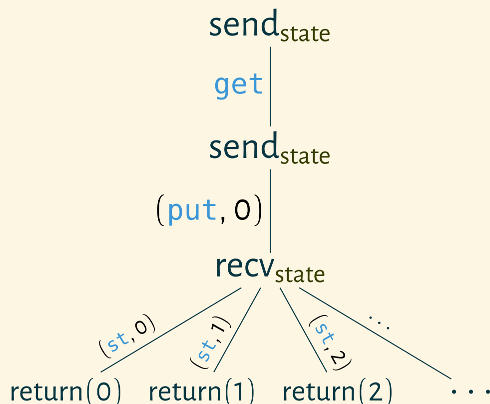
Generated ***coinductively*** by

$$t ::= \mathtt{return}(x) \mid \mathtt{send}_{\mathsf{p},m}(t) \mid \mathtt{recv}_{\mathsf{p}}(t_m)_{m \in M}$$

$m$: message $(\ell, v)$
$M$: set of messages
$\mathsf{p}$: participant
$x$: result; element of a given set $X$

(this is a free algebra on $X$, in $\omega$**Cpo**)

Computation tree $[\![t_{\mathsf{client}}]\!]$:



# Relating computation trees to session types

Two key coinductively defined relations:

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

  "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 $\boxed{t \sim_{\mathbb{T}} u}$:

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

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

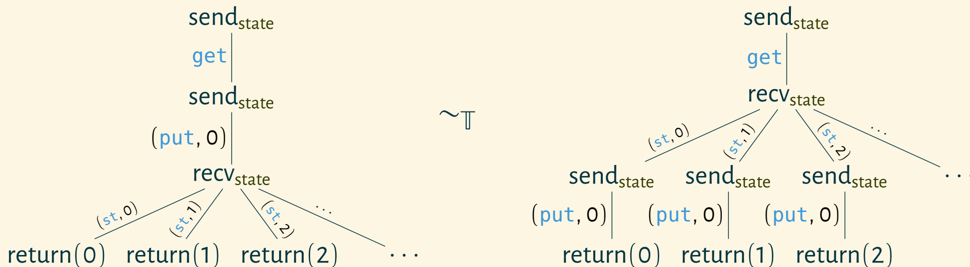
  Respects asynchronous subtyping: if $\mathbb{T} <: \mathbb{U}$, then $t \sim_{\mathbb{T}} u$ implies $t \sim_{\mathbb{U}} u$

# 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 $\mathbb{T} = \text{state} \oplus \texttt{get}\langle \textbf{unit} \rangle.\, \text{state} \,\&\, \texttt{st}\langle \textbf{int} \rangle.\, \text{state} \oplus \texttt{put}\langle \textbf{int} \rangle.\, \text{end}$:



# 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 $\vdash t : \mathrm{b} \mathbin{;} \mathbb{T}$ as an computation tree $[\![t]\!] \in \mathcal{N}(X)_{\mathbb{T}}$
- interpret each well-typed configuration $\vdash (\rho, t, \sigma) : \mathrm{b} \mathbin{;} \mathbb{T}$ as a computation tree $[\![(\rho, t, \sigma)]\!] \in \mathcal{N}[\![\mathrm{b}]\!]\mathbb{T}$

# Main theorem: adequacy

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

$$(\rho, t, \sigma) \sim_{\mathbb{T}} (\rho', t', \sigma') \quad \Leftrightarrow \quad [\![(\rho, t, \sigma)]\!] = [\![(\rho', t', \sigma')]\!]$$

The interpretation $[\![-]\!]$ 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 $\mathcal{N}$)

# Contributions

- 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

# Related work

- Dominic Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects.

  Encoding of session-typed $\pi$-calculus in effect-typed $\lambda$-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

# 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 $\mathbb{T} \xhookrightarrow{\mathsf{p}?\ell\langle \mathsf{b}' \rangle} \mathbb{U}$.

# Syntax of SafeMP

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

$v$: value
b: type
$\ell$: message label
p: participant

Computations:

$$
\begin{aligned}
t, u \;::=\;& v + w \mid v < w \mid \textbf{if } v \textbf{ then } t_1 \textbf{ else } t_2 \\
&\mid \textbf{let rec } f(x_1, \ldots, x_n) = t \textbf{ in } u && \text{(recursive function definition)} \\
&\mid f(v_1, \ldots, v_n) && \text{(function application)} \\
&\mid \textbf{return } v \mid \textbf{let } x = t \textbf{ in } u && \text{(returning and sequencing)} \\
&\mid \textbf{send } (\ell, v) \textbf{ to p then } t && \text{(sending a message to p)} \\
&\mid \textbf{recv p } \{\ell_i \langle x_i : \mathrm{b}_i \rangle .\, t_i\}_{i \in I} && \text{(receiving a message from p;} \\
& && \text{permitted messages are } (\ell_i, v) \text{ with } v : \mathrm{b}_i)
\end{aligned}
$$

# Sessions

Configurations running in parallel:

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

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

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

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