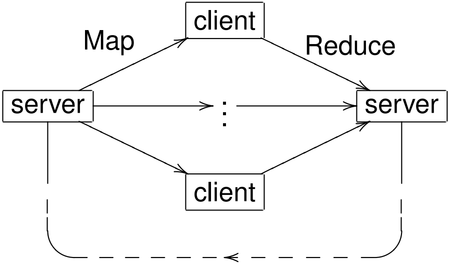
Dynamic Multirole Session Types

Pierre-Malo Deniélou   Nobuko Yoshida

Imperial College London



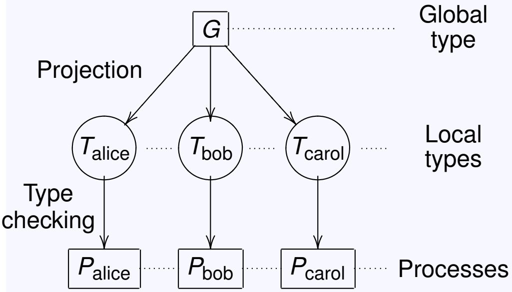
Multiparty session types (MPST)

- Today's distributed applications involve more and more agents that interact through complex communication patterns.
- Multiparty sessions types can describe these interactions and statically ensure type and communication safety and fidelity to a stipulated protocol.

Multiparty session types (MPST)

- Today's distributed applications involve more and more agents that interact through complex communication patterns.
- Multiparty sessions types can describe these interactions and statically ensure type and communication safety and fidelity to a stipulated protocol.

Multiparty session types in a nutshell



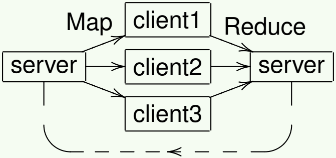
$G = \text{alice}{\to}\text{bob}\langle\text{nat}\rangle;$
$\quad \text{bob}{\to}\text{carol}\langle\text{nat}\rangle; \text{end}$

$T_{\text{bob}} = ?\langle\text{alice}, \text{nat}\rangle.$
$\quad !\,\langle\text{carol}, \text{nat}\rangle.\text{end}$

$P_{\text{bob}} = ?\langle\text{alice}\rangle(x).$
$\quad !\,\langle\text{carol}, x+1\rangle.\mathbf{0}$

Multiparty session example

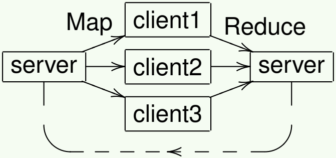
Map-Reduce in MPST



$$G_{org} = \mu \mathbf{x}.(\text{server} \to \text{client1}\langle \text{Map}\rangle; \\ \text{server} \to \text{client2}\langle \text{Map}\rangle; \\ \text{server} \to \text{client3}\langle \text{Map}\rangle; \\ \text{client1} \to \text{server}\langle \text{Reduce}\rangle; \\ \text{client2} \to \text{server}\langle \text{Reduce}\rangle; \\ \text{client3} \to \text{server}\langle \text{Reduce}\rangle); \mathbf{x}$$

Multiparty session example

Map-Reduce in MPST



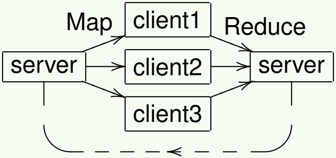
$$G_{org} = \mu\mathbf{x}.(\text{server}\to\text{client1}\langle\text{Map}\rangle; \text{server}\to\text{client2}\langle\text{Map}\rangle; \text{server}\to\text{client3}\langle\text{Map}\rangle; \text{client1}\to\text{server}\langle\text{Reduce}\rangle; \text{client2}\to\text{server}\langle\text{Reduce}\rangle; \text{client3}\to\text{server}\langle\text{Reduce}\rangle); \mathbf{x}$$

Main characteristics and features

Initial synchronisation
Fixed number of participants
Asynchronous semantics
Communication safety
Progress

Multiparty session example

Map-Reduce in MPST

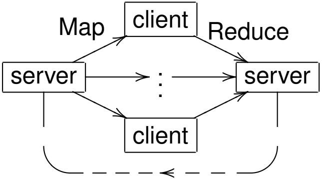


$$G_{org} = \mu \mathbf{x}.(\text{server}{\to}\text{client1}\langle\text{Map}\rangle; \text{server}{\to}\text{client2}\langle\text{Map}\rangle; \text{server}{\to}\text{client3}\langle\text{Map}\rangle; \text{client1}{\to}\text{server}\langle\text{Reduce}\rangle; \text{client2}{\to}\text{server}\langle\text{Reduce}\rangle; \text{client3}{\to}\text{server}\langle\text{Reduce}\rangle); \mathbf{x}$$

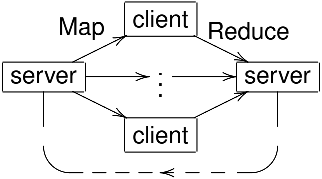
Main characteristics and features

	$\to$	
Initial synchronisation		Periodic synchronisation
Fixed number of participants		Variable number of participants
Asynchronous semantics	$\to$	Explicit parallel composition
Communication safety		Communication safety
Progress		Progress

Map-Reduce with dynamic multirole sessions



Map-Reduce with dynamic multirole sessions



$$G = \mu\mathbf{x}.\ \forall x : \text{client}.\{\text{server}\to x\langle\text{Map}\rangle;\ x\to\text{server}\langle\text{Reduce}\rangle\};\ \mathbf{x}$$

Roles

Two roles (server and client) who each correspond to a communication pattern. Multiple participants can instantiate roles.

Universal quantification

$\forall x : r.G'$ polls the current participants $\mathrm{p}_1, ..., \mathrm{p}_n$ of role r and, in parallel processes, binds x to each in the subsequent interaction, as in

$$\forall x : r.G' \quad \equiv \quad G'\{\mathrm{p}_1/x\} \mid ... \mid G'\{\mathrm{p}_n/x\}$$

Outline

Global Types

Global types follow standard Multiparty Session Type syntax, with the addition of universal quantification and explicit parallel composition.

$G ::=$		Global types
$\mid$	$p \to p' \{ l_i \langle \vec{p}_i \rangle \langle U_i \rangle . G_i \}_{i \in I}$	Labelled messages
$\mid$	$\forall x : r \setminus \vec{p}. G$	Universal quantification
$\mid$	$G \mid G'$	Parallel composition
$\mid$	$G ; G'$	Sequential composition
$\mid$	$\mu \mathbf{x}. G$	Recursion
$\mid$	$\mathbf{x}$	Recursion variable
$\mid$	ε	Inaction
$\mid$	end	End

Global Types

Global types follow standard Multiparty Session Type syntax, with the addition of universal quantification and explicit parallel composition.

$G ::=$		Global types
$\mid$	$p \to p' \{ l_i \langle \vec{p}_i \rangle \langle U_i \rangle . G_i \}_{i \in I}$	Labelled messages
$\mid$	$\forall x : r \setminus \vec{p}.G$	Universal quantification
$\mid$	$G \mid G'$	Parallel composition
$\mid$	$G ; G'$	Sequential composition
$\mid$	$\mu \mathbf{x}.G$	Recursion
$\mid$	$\mathbf{x}$	Recursion variable
$\mid$	ε	Inaction
$\mid$	end	End

Example (Semantical differences)

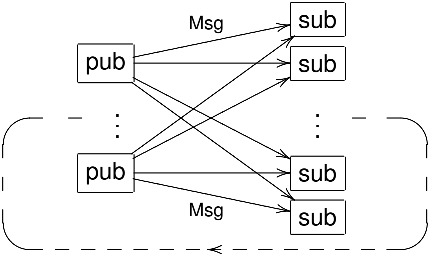
$$G_1 = \mu \mathbf{x}.\forall x : \text{client}.\{x \to \text{server}\langle \text{Msg} \rangle . \forall y : \text{client} \setminus x.\{\text{server} \to y \langle \text{Spread} \rangle\}\}; \mathbf{x}$$

$$G_2 = \mu \mathbf{x}.\forall x : \text{client}.\{x \to \text{server}\langle \text{Msg} \rangle\}; \forall y : \text{client}.\{\text{server} \to y \langle \text{Digest} \rangle\}; \mathbf{x}$$

$$G_3 = \mu \mathbf{x}.\forall x : \text{client}.\{x \to \text{server}\langle \text{Msg} \rangle ; \text{server} \to x \langle \text{Answer} \rangle\}; \mathbf{x}$$

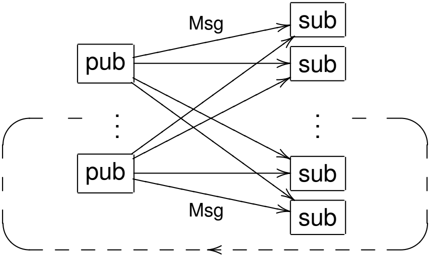
Publisher-Subscriber example

A set of publishers repeatedly broadcast their messages to a set of subscribers.



Publisher-Subscriber example

A set of publishers repeatedly broadcast their messages to a set of subscribers.



Global type for Pub-Sub

We write the global type using the universal quantifier for both the pub and the sub roles. The global type is the following:

$$\mu \mathbf{x}.(\forall x : \text{pub}.\forall y : \text{sub}.x \rightarrow y \langle \text{Msg} \rangle); \mathbf{x}$$

Processes

$$\begin{array}{llll} u & ::= & x \mid a \mid b \mid \ldots & \text{Shared channel} \\ p & ::= & \mathtt{p}:r \mid x:r & \text{Participant with role} \\ \vec{p} & ::= & \mathtt{p}::\vec{p} \mid x::\vec{p} \mid \varepsilon & \text{Participant list} \\ c & ::= & s[p] \mid y & \text{Session channel} \end{array}$$

$P ::=$ Processes

$\mid$	$u\langle G\rangle$	Session Init		
$\mid$	$u[p](y).P$	Join	$\mid$ if e then P else P	Conditional
$\mid$	$\mathtt{quit}\langle c\rangle$	Quit	$\mid$ $\mu X.P \mid X \mid \mathbf{0}$	Recursion
$\mid$	$c!\langle p, l\langle\vec{p}\rangle(e)\rangle$	Send	$\mid$ $(\nu\ a:G)P$	Restriction
$\mid$	$c?\langle p, \{l_i\langle\vec{p}_i\rangle(x_i).P_i\}_{i\in I}\rangle$	Receive	$\mid$ $(\nu\ s)P$	Session restriction
$\mid$	$c\forall(x:r\setminus\vec{p}).\{P\}$	Poll	$\mid$ $s:h$	Message buffer
$\mid$	$P \mid P$	Parallel	$\mid$ $a\langle s\rangle[\mathbb{R}]$	Session registry
$\mid$	$P;P$	Sequential		

Processes for Pub-Sub

$$P(z:\text{pub},m) = a[z:\text{pub}](s).\mu X.(s\forall(y:\text{sub}).\{s!\langle y,\text{Msg}\langle m\rangle\rangle\});X$$
$$P(z:\text{sub}) = a[z:\text{sub}](s).\mu X.(s\forall(x:\text{pub}).\{s?\langle x,\text{Msg}(w)\rangle\});X$$

Operational semantics

$a\langle s\rangle[\mathrm{R}]$ keeps the current list of participants in R.

$$a\langle G\rangle \;\rightarrow\; (\nu\ s)(a\langle s\rangle[\mathrm{R}] \mid s:\varepsilon) \qquad (\forall r_i \in G, \mathrm{R}(r_i) = \varnothing) \quad \lfloor\text{INIT}\rfloor$$

$$a[\mathrm{p}:r](y).P \mid a\langle s\rangle[\mathrm{R}\cdot r:\mathrm{P}] \;\rightarrow\; P\{s[\mathrm{p}:r]/y\} \mid a\langle s\rangle[\mathrm{R}\cdot r:\mathrm{P} \uplus \{\mathrm{p}\}] \quad \lfloor\text{JOIN}\rfloor$$

$$\mathtt{quit}\langle s[\mathrm{p}:r]\rangle \mid a\langle s\rangle[\mathrm{R}\cdot r:\mathrm{P}] \;\rightarrow\; a\langle s\rangle[\mathrm{R}\cdot r:\mathrm{P}\setminus \mathrm{p}] \quad \lfloor\text{QUIT}\rfloor$$

$$s[\mathrm{p}:r]!\langle \mathrm{p}':r', l\langle\vec{\mathrm{p}}\rangle\langle v\rangle\rangle \mid a\langle s\rangle[\mathrm{R}] \mid s:h \;\rightarrow\; a\langle s\rangle[\mathrm{R}] \mid s:h\cdot(\mathrm{p}:r,\ \mathrm{p}':r',\ l\langle\vec{\mathrm{p}}\rangle\langle v\rangle)$$
$$(\mathrm{p}\in\mathrm{R}(r) \wedge \mathrm{p}'\in\mathrm{R}(r')) \quad \lfloor\text{SEND}\rfloor$$

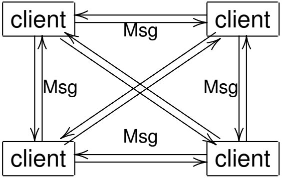
$$s[\mathrm{p}:r]?\langle \mathrm{p}':r', \{l_i\langle\vec{\mathrm{p}}_i\rangle(x_i).P_i\}_{i\in I}\rangle \mid a\langle s\rangle[\mathrm{R}]$$
$$\mid s:(\mathrm{p}':r',\ \mathrm{p}:r,\ l_k\langle\vec{\mathrm{p}}_k\rangle\langle v\rangle)\cdot h \rightarrow P_k\{v/x_k\} \mid a\langle s\rangle[\mathrm{R}] \mid s:h$$
$$(\mathrm{p}\in\mathrm{R}(r) \wedge k\in I) \quad \lfloor\text{RECV}\rfloor$$

$$s[\mathrm{p}:r']\forall(x:r\setminus\vec{\mathrm{p}}).\{P\} \mid a\langle s\rangle[\mathrm{R}] \;\rightarrow\; P\{\mathrm{p}_1/x\} \mid \ldots \mid P\{\mathrm{p}_k/x\} \mid a\langle s\rangle[\mathrm{R}]$$
$$(\mathrm{R}(r)\setminus\vec{\mathrm{p}} = \{\mathrm{p}_1,..,\mathrm{p}_k\} \wedge \mathrm{p}\in\mathrm{R}(r')) \quad \lfloor\text{POLL}\rfloor$$

Another example: peer-to-peer chat

At every step, each client sends a message to every other client.

$$G = \mu \mathbf{x}.(\forall x : \text{client}.\forall y : \text{client} \setminus x.\{x \to y\, \text{Msg}\langle \text{string} \rangle\}); \mathbf{x}$$



Local Type

$$T_{\text{client}}(z) = \mu \mathbf{x}.(\forall y : \text{client} \setminus z.\{!\langle y, \text{Msg}\langle \text{string} \rangle \rangle\} \mid \forall x : \text{client} \setminus z.\{?\langle x, \text{Msg}\langle \text{string} \rangle \rangle\}); \mathbf{x}$$

How do we go from the global type to the local type?

Projection: $G \uparrow z : r$

Intuition

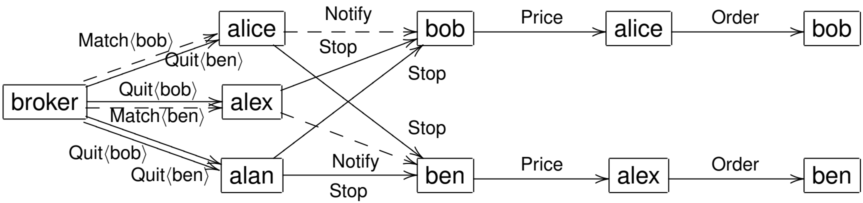
$$
\begin{array}{ll}
(\forall x : r.G) & \uparrow \mathtt{p}_i : r \\
(G\{\mathtt{p}_1/x\} \mid ... \mid G\{\mathtt{p}_k/x\}) & \uparrow \mathtt{p}_i : r \\
(G\{\mathtt{p}_1/x\} \uparrow \mathtt{p}_i : r) \mid ... \mid (G\{\mathtt{p}_k/x\} \uparrow \mathtt{p}_i : r) & \\
(G\{\mathtt{p}_i/x\} \uparrow \mathtt{p}_i : r) \mid \forall x : r \setminus \mathtt{p}_i.(G \uparrow \mathtt{p}_i : r) &
\end{array}
$$

Main rules

$$
\begin{array}{rcl}
p \to p' \{l_i\langle \vec{p}_i\rangle\langle U_i\rangle : G_i\}_{i\in I} \uparrow p & = & !\langle p', \{l_i\langle \vec{p}_i\rangle\langle U_i\rangle.G_i \uparrow p\}_{i\in I}\rangle \\
p' \to p\{l_i\langle \vec{p}_i\rangle\langle U_i\rangle : G_i\}_{i\in I} \uparrow p & = & ?\langle p', \{l_i\langle \vec{p}_i\rangle\langle U_i\rangle.G_i \uparrow p\}_{i\in I}\rangle \\
p \to p\{l_i\langle \vec{p}_i\rangle\langle U_i\rangle : G_i\}_{i\in I} \uparrow p & = & !\langle p, \{l_i\langle \vec{p}_i\rangle\langle U_i\rangle.?\langle p, l_i\langle \vec{p}_i\rangle\langle U_i\rangle.G_i \uparrow p\rangle\}_{i\in I}\rangle \\
p' \to p'' \{l_i\langle \vec{p}_i\rangle\langle U_i\rangle.G_i\}_{i\in I} \uparrow p & = & \bigsqcup_{i\in I}\{G_i \uparrow p\} \\
(\forall x : r \setminus \vec{p}.G) \uparrow z : r & = & G\{z/x\} \uparrow z : r \mid \forall x : r \setminus z :: \vec{p}.(G \uparrow z : r) \ (z \notin \vec{p}) \\
(\forall x : r \setminus \vec{p}.G) \uparrow p & = & \forall x : r \setminus \vec{p}.(G \uparrow p) \qquad \text{(otherwise)}
\end{array}
$$

Auction example, disambiguation of parallel branches

A single broker forms pairs of buyers and sellers.



Global type for Auction

$$G = \forall x : \text{buyer}. \forall y : \text{seller}. \text{broker} \to x \{ \text{Match}\langle y \rangle . x \to y \langle \text{Notify} \rangle . y \to x \langle \text{Price} \rangle . x \to y \langle \text{Order} \rangle , \text{Quit}\langle y \rangle . \ x \to y \ \langle \text{Stop} \rangle \} ; \text{end}$$

Well-formedness

- Syntax correctness

$$\begin{array}{ll} \times & G_1 = \mu\mathbf{x}.(\text{server}{\to}\text{client}\langle\text{Msg}\rangle;\mathbf{x} \mid \text{server}{\to}\text{broker}\langle\text{Notify}\rangle;\mathbf{x}) \\ \surd & G_2 = \mu\mathbf{x}.(\text{server}{\to}\text{client}\langle\text{Msg}\rangle \mid \text{server}{\to}\text{broker}\langle\text{Notify}\rangle);\mathbf{x} \\ \surd & G_3 = \mu\mathbf{x}.\text{server}{\to}\text{client}\langle\text{Msg}\rangle;\mathbf{x} \mid \mu\mathbf{y}.\text{server}{\to}\text{broker}\langle\text{Notify}\rangle;\mathbf{y} \end{array}$$

- Projectability (projection always returns)

$$\begin{array}{l} \times\ G_4 = \text{broker}{\to}\text{buyer}\{\text{Notify.buyer}{\to}\text{seller}\langle\text{Msg}\rangle;\text{seller}{\to}\text{buyer}\langle\text{Pay}\rangle, \\ \qquad\qquad\qquad\qquad \text{Quit.buyer}{\to}\text{seller}\langle\text{Msg}\rangle\} \\ \surd\ G_5 = \text{broker}{\to}\text{buyer}\{\text{Notify.buyer}{\to}\text{seller}\langle\text{Price}\rangle;\text{seller}{\to}\text{buyer}\langle\text{Pay}\rangle, \\ \qquad\qquad\qquad\qquad \text{Quit.buyer}{\to}\text{seller}\langle\text{Stop}\rangle\} \end{array}$$

- Linearity (no possible confusion between parallel branches)

$$\begin{array}{ll} \times & G_6 = \forall x:\text{buyer}.\forall y:\text{seller}.\{\text{broker}{\to}x\langle\text{Msg}\rangle.x{\to}y\langle\text{Notify}\rangle\} \\ \surd & G_7 = \forall x:\text{buyer}.\forall y:\text{seller}.\{\text{broker}{\to}x\langle\text{Msg}\langle y\rangle\rangle.x{\to}y\langle\text{Notify}\rangle\} \end{array}$$

Typing system

We show only a selection of rules.

$$\frac{\Gamma \vdash u : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \uparrow p}{\Gamma \vdash u[p](y).P \triangleright \Delta}\,[\text{JOIN}] \qquad \frac{\Gamma \vdash P \triangleright \Delta, c : \mathsf{end}}{\Gamma \vdash \mathtt{quit}\langle c \rangle; P \triangleright \Delta, c : \mathsf{end}}\,[\text{LEAVE}]$$

$$\frac{\Gamma, x : r \vdash P \triangleright c : T \quad \Gamma \vdash \vec{p}}{\Gamma \vdash c\forall(x : r \setminus \vec{p}).\{P\} \triangleright c : \forall x : r \setminus \vec{p}.T}\,[\text{POLLING}]$$

$$\frac{\Gamma \vdash a : \langle G \rangle \quad \{r_i\}_{i \in I} = dom(\mathrm{R}) \quad G \uparrow x_i : r_i = T_i}{\Gamma \vdash_{\varnothing} a\langle s \rangle[\mathrm{R}] \triangleright \{s[\mathrm{p}_{ji} : r_i] : T_i\{\mathrm{p}_{ji}/x_i\}\}_{i \in I, \mathrm{p}_{ji} \notin \mathrm{R}(r_i)}}\,[\text{RGST}] \qquad \frac{\Gamma \vdash_{\Sigma_i} P_i \triangleright \Delta_i \ (i = 1, 2)}{\Gamma \vdash_{\Sigma_1 \uplus \Sigma_2} P_1 \mid P_2 \triangleright \Delta_1 * \Delta_2}\,[\text{GPAR}]$$

Theorem (Type safety)

Suppose $\Gamma \vdash P \triangleright \Delta$. *For any* P' *such that* $P \rightarrow^* P'$, P' *has no type error.*

Limitations

The semantics and type system are not constrained enough ...

Leaving a session

The typing rule [LEAVE] only allows a participant to leave when its local type is end. It means that if G is of the form $\mu\mathbf{x}.G_0;\mathbf{x};\mathsf{end}$, no one can leave ...

$$\mu\mathbf{x}.\forall x:\text{client}.\forall y:\text{client}\setminus x.\{x\to y\,\text{Msg}\langle\text{string}\rangle\};\mathbf{x}$$

Polling consistency for communication safety

$$a[z:\text{client}](s).\mu X.(s\forall(y:\text{client}\setminus z).\{s!\langle y,\text{Msg}\langle m\rangle\rangle\} \mid s\forall(x:\text{client}\setminus z).\{s?\langle x,\text{Msg}(w)\rangle\});X$$

All local polling operations should give the same list, otherwise messages are unexpected or absent.

Multiparty locking

We need to temporarily *block* late participants from joining in the middle of a session execution in order to prevent any interference with polling: we automatically introduce a locking mechanism $\mathtt{lock}\{G\}$.

$$\mu\mathbf{x}.\mathtt{lock}\{\forall x:\text{client}.\forall y:\text{client}\setminus x.\{x\to y\,\text{Msg}\langle\text{string}\rangle\}\};\mathbf{x}$$

Syntax and semantics

$$P ::= \ldots \mid c\ \mathtt{lock} \mid c\ \mathtt{unlock} \mid a^{\circ}[\mathrm{R},\Lambda] \mid a^{\bullet}[\mathrm{R},\Lambda]$$

$$\Lambda ::= \varnothing \mid \Lambda\cup\{\mathtt{p}:r\}$$

$$s[\mathtt{p}:r]\mathtt{lock} \mid a\langle s\rangle[\mathrm{R}] \to a^{\circ}\langle s\rangle[\mathrm{R},\{\mathtt{p}:r\}] \qquad \lfloor\text{LOCK}\rfloor$$

$$s[\mathtt{p}:r]\mathtt{lock} \mid a^{\circ}\langle s\rangle[\mathrm{R},\Lambda] \to \begin{cases} a^{\circ}\langle s\rangle[\mathrm{R},\Lambda\uplus\{\mathtt{p}:r\}] & (\mathrm{R}\not\approx\Lambda\uplus\{\mathtt{p}:r\}) \quad \lfloor\text{UP}\rfloor \\ a^{\bullet}\langle s\rangle[\mathrm{R},\Lambda\uplus\{\mathtt{p}:r\}] & (\mathrm{R}\approx\Lambda\uplus\{\mathtt{p}:r\}) \quad \lfloor\text{TOP}\rfloor \end{cases}$$

$$s[\mathtt{p}:r]\mathtt{unlock} \mid a^{\bullet}\langle s\rangle[\mathrm{R},\Lambda\uplus\{\mathtt{p}:r\}] \to \begin{cases} a^{\bullet}\langle s\rangle[\mathrm{R},\Lambda] & (\Lambda\neq\varnothing) \quad \lfloor\text{DOWN}\rfloor \\ a\langle s\rangle[\mathrm{R}] & (\Lambda=\varnothing) \quad \lfloor\text{UNLOCK}\rfloor \end{cases}$$

$$s[\mathtt{p}:r]!\langle\mathtt{p}':r',l\langle\vec{\mathtt{p}}\rangle\langle v\rangle\rangle \mid a^{\bullet}\langle s\rangle[\mathrm{R},\Lambda] \mid s:h \to a^{\bullet}\langle s\rangle[\mathrm{R},\Lambda] \mid s:h\cdot(\mathtt{p}:r,\ \mathtt{p}':r',\ l\langle\vec{\mathtt{p}}\rangle\langle v\rangle)$$

$$\ldots \qquad \lfloor\text{SEND}\rfloor$$

Locking

Typing locks

$$G ::= \ldots \mid \texttt{lock}\{G\} \qquad T ::= \ldots \mid \texttt{lock} \mid \texttt{unlock}$$

Well-locked global types are of the form $\texttt{lock}\{G_0\};\mathsf{end}$.
Persistently well-locked global types are of the form $\mu\mathbf{x}.\texttt{lock}\{G_0\};\mathbf{x};\mathsf{end}$

$$\texttt{lock}\{G\}\uparrow z{:}r \quad = \quad \texttt{lock};(G\uparrow z{:}r);\texttt{unlock}$$

$$\frac{\Gamma \vdash \mathsf{Env}}{\Gamma \vdash c\ \texttt{lock} \triangleright c{:}\texttt{lock}} \qquad \frac{\Gamma \vdash \mathsf{Env}}{\Gamma \vdash c\ \texttt{unlock} \triangleright c{:}\texttt{unlock}} \qquad \frac{\Gamma \vdash P \triangleright \Delta, c{:}\mathsf{end} \quad \Gamma \vdash u{:}\langle G\rangle}{\Gamma \vdash \texttt{quit}\langle c\rangle;P \triangleright \Delta, c{:}G\uparrow p}$$

Single iteration chat client

$$P_{\mathsf{client}}(\mathtt{p}) = a[\mathtt{p}{:}\mathsf{client}](s).(s\,\texttt{lock};s\forall(y{:}\mathsf{client}\setminus z).\{s!\langle y,\mathsf{Msg}\langle m\rangle\rangle\} \mid s\forall(x{:}\mathsf{client}\setminus z).\{s?\langle x,\mathsf{Msg}(w)\rangle\});s\,\texttt{unlock};\texttt{quit}\langle s\rangle$$

Theorems

Theorem (Communication Safety)

Every sent message is expected by a receiver. Every receiver will receive a message.

Theorem (Progress)

Well-locked and well-typed processes do not reach a deadlock state.

Theorem (Join progress)

Persistantly well-locked and well-typed processes can progress and integrate new joiners.

Implementation

- An extension to OCaml
- The compiler generates from the global type a taylored runtime
- The runtime deals with transport (UDP, TCP, AMQP) and registry
- A continuation-based programming interface

Conclusion and future work

Dynamic multirole session types

- A conservative extension of multiparty session types
- A new universal quantification to ease programming and typing
- Strong safety and progress guarantees at the price of synchronisation

Ongoing work

- Automatically distribute the registry
- Give a structure (topology) to role participants
- Getting rid of some aspects of the synchronisation

Conclusion and future work

Dynamic multirole session types

- A conservative extension of multiparty session types
- A new universal quantification to ease programming and typing
- Strong safety and progress guarantees at the price of synchronisation

Ongoing work

- Automatically distribute the registry
- Give a structure (topology) to role participants
- Getting rid of some aspects of the synchronisation

Thanks