University of Glasgow | School of Computing Science

Resource Sharing via Capability-Based Multiparty Session Types

A. Laura Voinea, Ornela Dardha, Simon J. Gay

School of Computing Science
University of Glasgow, UK

19 Dec 2019

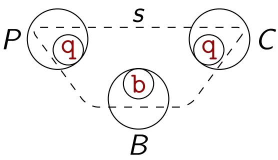
Supported by the UK EPSRC grant EP/K034413/1, "From Data Types to Session Types: A Basis for Concurrency and Distribution (ABCD)", by the EU HORIZON 2020 MSCA RISE project 778233 "BehAPI: Behavioural Application Program Interfaces", and by an EPSRC PhD studentship

Session Types and Linearity

- Session types must control aliasing to avoid races
- Most session type systems use strict linear, and some affine typing to guarantee unique ownership of channel endpoints
- Why is linearity a problem?
  - Inflexible programming
  - Excludes scenarios that make use of shared channels

Motivating Example: Producer-Consumer

- Producer (P) and Consumer (C) synchronise by communicating via a **shared channel** $s[q]$ with Buffer (B)
- Producer sends data to Buffer
- Consumer requests data from Buffer, to which Buffer responds with a send message
- Linear typing of the channel does not allow this

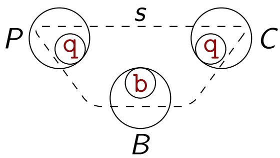


Global type:

$$G_0 = \; q \to b\colon \mathrm{add}(\mathtt{Int}).q \to b\colon \mathrm{request}().$$
$$b \to q\colon \mathrm{send}(\mathtt{Int}).G_0$$

Motivating Example: Producer-Consumer

- Producer (P) and Consumer (C) synchronise by communicating via a **shared channel** $\mathtt{s}[\mathtt{q}]$ with Buffer (B)
- Producer sends data to Buffer
- Consumer requests data from Buffer, to which Buffer responds with a send message
- Linear typing of the channel does not allow this

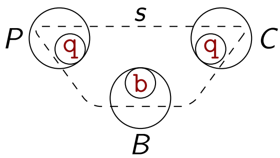


Global type:

$$G_0 = \mathtt{q}\to\mathtt{b}\text{: add}(\mathtt{Int}).\ \mathtt{q}\to\mathtt{b}\text{: request}().$$
$$\mathtt{b}\to\mathtt{q}\text{: send}(\mathtt{Int}).G_0$$

Motivating Example: Producer-Consumer

- Producer (P) and Consumer (C) synchronise by communicating via a **shared channel** s[q] with Buffer (B)
- Producer sends data to Buffer
- Consumer requests data from Buffer, to which Buffer responds with a send message
- Linear typing of the channel does not allow this

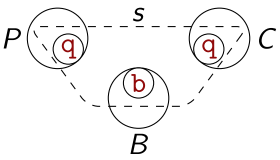


Global type:

$$G_0 = \ \mathtt{q}\to\mathtt{b}\colon \mathrm{add}(\mathtt{Int}).\ \mathtt{q}\to\mathtt{b}\colon \mathrm{request}().$$
$$\mathtt{b}\to\mathtt{q}\colon \mathrm{send}(\mathtt{Int}).G_0$$

Motivating Example: Producer-Consumer

- Producer (P) and Consumer (C) synchronise by communicating via a **shared channel** $s[q]$ with Buffer (B)
- Producer sends data to Buffer
- Consumer requests data from Buffer, to which Buffer responds with a send message
- Linear typing of the channel does not allow this



Global type:

$$G_0 = q \to b: \mathrm{add}(\mathtt{Int}).q \to b: \mathrm{request}().$$
$$b \to q: \mathrm{send}(\mathtt{Int}).\ G_0$$

Our Work

We combine MPST with Capabilities (Capability Calculus) [Crary et al., 1999, Walker and Morrisett, 2000]

Split the channel into two first class entities:

- the channel itself
- the capability of using it

Linearity is moved to the use of capabilities

Channels can be shared and aliased

Capabilities can be transferred

Our Work

A channel has a tracked type $c : \mathtt{tr}(\rho)$

The capability ρ is separately associated with the session type $\{\rho \mapsto S\}$

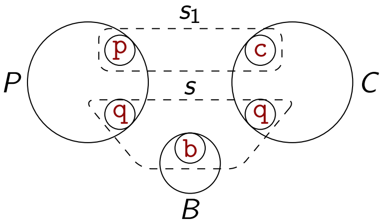
Each process has a list of capabilities for the channels it can use, $C = \{\rho_1 \mapsto S_1\} \otimes \ldots \otimes \{\rho_n \mapsto S_n\}$

The type system maintains the invariant that $\rho_1 \ldots \rho_n$ are distinct

Our Work

The global type G_1 is the producer-consumer protocol with 4 roles: p, c, q, b:

$$G_1 \quad = \quad \begin{array}{l} \mathtt{q}{\to}\mathtt{b}\text{: add}(\mathtt{Int}). \\ \mathtt{p}{\to}\mathtt{c}\text{: turn}(\mathtt{tr}(\rho_{\mathtt{q}})). \\ \mathtt{q}{\to}\mathtt{b}\text{: request}(). \\ \mathtt{b}{\to}\mathtt{q}\text{: send}(\mathtt{Int}). \\ \mathtt{c}{\to}\mathtt{p}\text{: turn}(\mathtt{tr}(\rho_{\mathtt{q}})).\ G_1 \end{array}$$



$\rho_{\mathtt{q}}$ is usually too specific because it refers to a particular channel's capability.

Channel delegation

To abstract from specific channels we use existential types:
$\exists[\rho|\{\rho \mapsto S\}].\mathtt{tr}(\rho)$

- channel
- abstracted capability

This enables a channel to be delegated, with the information that it is linked to some capability, which will be transmitted in a separate message.

Global Types

G	$::=$	$\texttt{end}$	termination
	$\mid$	$\mathtt{p}\to\mathtt{q}\colon \{l_i(U_i).G_i\}_{i\in I}$	interaction
	$\mid$	$\mathtt{p}\to\mathtt{q}\colon \{l_i(\exists[\rho_i \mid \{\rho_i \mapsto U_i\}].tr(\rho_i)).G_i\}_{i\in I}$	pack interaction
	$\mid$	$\mu\,\mathtt{t}.G$	recursive type
	$\mid$	$\mathtt{t}$	type variable

Session Types

$$
\begin{array}{rlll}
S & ::= & \mathtt{end} & \text{terminated session} \\
& \mid & \mathtt{p} \oplus_{i \in I} ! l_i(U_i).S_i & \text{select towards role } \mathtt{p} \\
& \mid & \mathtt{p} \&_{i \in I} ? l_i\,(U_i)\,.S_i & \text{branching from role } \mathtt{p} \\
& \mid & \mathtt{p} \oplus_{i \in I} ! l_i(\exists[\rho_i | \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i & \text{pack select towards role } \mathtt{p} \\
& \mid & \mathtt{p} \&_{i \in I} ? l_i(\exists[\rho_i | \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i & \text{pack branch from role } \mathtt{p} \\
& \mid & \mathtt{t} & \text{type variable} \\
& \mid & \mu\,\mathtt{t}.S & \text{recursive session type}
\end{array}
$$

Message payload types

$$
\begin{array}{rlll}
U & ::= & B \mid S \text{ (closed)} & \text{ground type, session type} \\
& \mid & \mathtt{tr}(\rho) \mid \{\rho \mapsto S\} & \text{tracked type, capability type} \\
B & ::= & \mathtt{Int} \mid \mathtt{Bool} & \textit{ground type}
\end{array}
$$

Session Types

$S ::= \mathtt{end}$	terminated session
$\mid\ \mathtt{p} \oplus_{i \in I} !l_i(U_i).S_i$	select towards role p
$\mid\ \mathtt{p} \&_{i \in I} ?l_i\,(U_i)\,.S_i$	branching from role p
$\mid\ \mathtt{p} \oplus_{i \in I} !l_i(\exists[\rho_i \mid \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i$	pack select towards role p
$\mid\ \mathtt{p} \&_{i \in I} ?l_i(\exists[\rho_i \mid \{\rho_i \mapsto U_i\}].\mathtt{t}\ldots(\rho_i)).S_i$	pack branch from role p
$\mid\ \mathtt{t}$	...le
$\mid\ \mu\,\mathtt{t}.S$	...ession type

Choose a label l_i and send it to p with a payload of type U_i, and bind the capability ρ_i for the continuation type S_i

Message payload types

$U ::= B \mid S$ (closed)	ground type, session type
$\mid\ \mathtt{tr}(\rho) \mid \{\rho \mapsto S\}$	tracked type, capability type
$B ::= \mathtt{Int} \mid \mathtt{Bool}$	*ground type*

Session Types

Receive a label l_i and a channel reference of type U_i from role $\mathtt{p}$, and bind the capability ρ_i for the continuation type S_i

$S \; ::= \; \mathtt{end}$	...d session
$\mid \; \mathtt{p} \oplus_{i \in I} ! l_i(U_i).S_i$	...vards role $\mathtt{p}$
$\mid \; \mathtt{p} \,\&_{i \in I} ? l_i\,(U_i)\,.S_i$	...g from role $\mathtt{p}$
$\mid \; \mathtt{p} \oplus_{i \in I} ! l_i(\exists[\rho_i \vert \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i$	pack select towards role $\mathtt{p}$
$\mid \; \mathtt{p} \,\&_{i \in I} ? l_i(\exists[\rho_i \vert \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i$	pack branch from role $\mathtt{p}$
$\mid \; \mathtt{t}$	type variable
$\mid \; \mu\, \mathtt{t}.S$	recursive session type

Message payload types

$U \; ::= \; B \mid S$ (closed)	ground type, session type
$\mid \; \mathtt{tr}(\rho) \mid \{\rho \mapsto S\}$	tracked type, capability type
$B \; ::= \; \mathtt{Int} \mid \mathtt{Bool}$	*ground type*

Producer-Consumer Global Type

The global type G is the producer-consumer protocol with 4 roles: $\mathtt{p}$ for the producer, $\mathtt{c}$ for the consumer, $\mathtt{q}$ shared between producer and consumer, and $\mathtt{b}$ for buffer:

$$\begin{aligned} G_2 = \quad & \mathtt{p}{\rightarrow}\mathtt{c}\colon \mathsf{buffer}(\exists[\rho_{\mathtt{q}} | \{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}].\mathtt{tr}(\rho_{\mathtt{q}})).\mu\,\mathtt{t}.\mathtt{q}{\rightarrow}\mathtt{b}\colon \mathsf{add}(\mathtt{Int}). \\ & \mathtt{p}{\rightarrow}\mathtt{c}\colon \mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}).\mathtt{q}{\rightarrow}\mathtt{b}\colon \mathsf{request}().\mathtt{b}{\rightarrow}\mathtt{q}\colon \mathsf{send}(\mathtt{Int}). \\ & \mathtt{c}{\rightarrow}\mathtt{p}\colon \mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S_{\mathtt{q}}\}).\mathtt{t} \end{aligned}$$

The projection $G_2 \upharpoonright \mathtt{q}$ yields the (local) session type describing how a communication channel should be used as $\mathtt{q}$:

$$\begin{aligned} S_{\mathtt{q}} = G_2 \upharpoonright \mathtt{q} \quad &= \quad \mu\,\mathtt{t}.\mathtt{b}{\oplus}!\mathsf{add}(\mathtt{Int}).\mathtt{b}{\oplus}!\mathsf{request}(\mathtt{Str}).\mathtt{b}\&?\mathsf{send}(\mathtt{Int}).t \\ S'_{\mathtt{q}} \quad &= \quad \mathtt{b}{\oplus}!\mathsf{request}(\mathtt{Str}).\mathtt{b}\&?\mathsf{send}(\mathtt{Int}).S_{\mathtt{q}} \end{aligned}$$

Producer-Consumer Local Types

Session type for role $\mathtt{p}$:

$$S_{\mathtt{p}} = G_2 \upharpoonright \mathtt{p} = \begin{array}{l} \mathtt{c}\oplus!\mathsf{buffer}(\exists[\rho_{\mathtt{q}}|\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}].\mathtt{tr}(\rho_{\mathtt{q}})).\mu\,\mathtt{t}. \\ \mathtt{c}\oplus!\mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}).\mathtt{c}\&?\mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S_{\mathtt{q}}\}).t \end{array}$$

Session type for role $\mathtt{c}$:

$$S_{\mathtt{c}} = G_2 \upharpoonright \mathtt{c} = \begin{array}{l} \mathtt{p}\&?\mathsf{buffer}(\exists[\rho_{\mathtt{q}}|\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}].\mathtt{tr}(\rho_{\mathtt{q}})).\mu\,\mathtt{t}. \\ \mathtt{p}\&?\mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}).\mathtt{p}\oplus!\mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S_{\mathtt{q}}\}).t \end{array}$$

Session type for role $\mathtt{b}$:

$$S_{\mathtt{b}} = G \upharpoonright \mathtt{b} = \mu\,\mathtt{t}.\mathtt{q}\&?\mathsf{add}(\mathtt{Int}).\mathtt{q}\&?\mathsf{request}(\mathtt{Str}).\mathtt{q}\oplus!\mathsf{send}(\mathtt{Int}).t$$

Process syntax

$$
\begin{array}{llll}
P & ::= & \mathbf{0} \;\mid\; P \mid Q \;\mid\; (\nu\, s)P & \text{inaction, parallel, restriction} \\
& \mid & \mathsf{c}[\mathtt{p}] \oplus \langle l(v) \rangle . P \;\mid\; \mathsf{c}[\mathtt{p}] \&_{i \in I} \{ l_i(x_i).\, P_i \} & \text{select, branch} \\
& \mid & \mathsf{c}[\mathtt{p}] \oplus \langle l(\mathsf{pack}(\rho, \mathtt{s}[\mathtt{q}])) \rangle . P & \text{select pack} \\
& \mid & \mathsf{c}[\mathtt{p}] \&_{i \in I} \{ l_i(\mathsf{pack}(\rho_i, \mathtt{s_i}[\mathtt{q}])).\, P_i \} & \text{branch pack} \\
& \mid & \mathsf{def}\; D \;\mathsf{in}\; P \;\mid\; X\langle \tilde{x} \rangle & \text{recursion, process call} \\
\\
D & ::= & X\langle \tilde{x} \rangle = P & \text{process declaration} \\
\\
\mathsf{c} & ::= & x \;\mid\; \mathtt{s}[\mathtt{p}] & \text{variable, channel with role } \mathtt{p} \\
v & ::= & \mathsf{c} \;\mid\; \rho & \text{channel, capability} \\
& \mid & \mathsf{true} \;\mid\; \mathsf{false} \;\mid\; 0 \;\mid\; 1 \;\mid\; \ldots & \text{base value}
\end{array}
$$

Process syntax

Channel capability is sent, and the execution continues as process P

P	$::= \ \mathbf{0} \ \mid \ P \mid Q \ \mid \ (\nu\, s)P$	inaction, parallel, restriction
	$\mid \ \mathrm{c}[\mathtt{p}] \oplus \langle l(v) \rangle.P \ \mid \ \mathrm{c}[\mathtt{p}] \,\&_{i \in I}\{l_i(x_i).\, P_i\}$	select, branch
	$\mid \ \mathrm{c}[\mathtt{p}] \oplus \langle l(\mathrm{pack}(\rho, \mathtt{s}[\mathtt{q}])) \rangle.P$	select pack
	$\mid \ \mathrm{c}[\mathtt{p}] \,\&_{i \in I}\{l_i(\mathrm{pack}(\rho_i, \mathtt{s_i}[\mathtt{q}])).\, P_i\}$	branch pack
	$\mid \ \mathrm{def}\, D \,\mathrm{in}\, P \ \mid \ X\langle \tilde{x} \rangle$	recursion, process call
D	$::= \ X\langle \tilde{x} \rangle = P$	process declaration
c	$::= \ x \ \mid \ \mathtt{s}[\mathtt{p}]$	variable, channel with role $\mathtt{p}$
v	$::= \ \mathrm{c} \ \mid \ \rho$	channel, capability
	$\mid \ \mathrm{true} \ \mid \ \mathrm{false} \ \mid \ 0 \ \mid \ 1 \ \mid \ \ldots$	base value

Process syntax

Receives labelled channel capability $l_i(\mathsf{pack}(\rho_i, \mathtt{s_i}[\mathtt{q}]))$, then the execution continues as P_i

P	$::=$	$\mathbf{0} \mid P \mid Q \mid (\nu\, s)P$	restriction
	$\mid$	$\mathsf{c}[\mathtt{p}] \oplus \langle l(v) \rangle.P \mid \mathsf{c}[\mathtt{p}] \&_{i\in I}\{l_i(x_i).P_i\}$	select, branch
	$\mid$	$\mathsf{c}[\mathtt{p}] \oplus \langle l(\mathsf{pack}(\rho, \mathtt{s}[\mathtt{q}])) \rangle.P$	select pack
	$\mid$	$\mathsf{c}[\mathtt{p}] \&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{s_i}[\mathtt{q}])).\, P_i\}$	branch pack
	$\mid$	$\mathsf{def}\, D \,\mathsf{in}\, P \mid X\langle\tilde{x}\rangle$	recursion, process call
D	$::=$	$X\langle\tilde{x}\rangle = P$	process declaration
c	$::=$	$x \mid \mathtt{s}[\mathtt{p}]$	variable, channel with role $\mathtt{p}$
v	$::=$	$\mathsf{c} \mid \rho$	channel, capability
	$\mid$	$\mathsf{true} \mid \mathsf{false} \mid 0 \mid 1 \mid \ldots$	base value

Process Representation – Producer

$$
\begin{aligned}
S_{\mathtt{p}} =\ & \mathtt{c}\oplus!\mathsf{buffer}(\exists[\rho_{\mathtt{q}}|\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}].\mathtt{tr}(\rho_{\mathtt{q}})).\mu\, \mathtt{t}.\mathtt{c}\oplus!\mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}).\\
& \mathtt{c}\&?\mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S_{\mathtt{q}}\}).t\\
S_{\mathtt{q}} =\ & \mu\, \mathtt{t}.\mathtt{b}\oplus!\mathsf{add}(\mathtt{Int}).\mathtt{b}\oplus!\mathsf{request}(\mathtt{Str}).\mathtt{b}\&?\mathsf{send}(\mathtt{Int}).t\\
S'_{\mathtt{q}} =\ & \mathtt{b}\oplus!\mathsf{request}(\mathtt{Str}).\mathtt{b}\&?\mathsf{send}(\mathtt{Int}).S_{\mathtt{q}}
\end{aligned}
$$

Producer Process:

$$
\begin{aligned}
&\mathsf{Produce}\langle \mathtt{x} : \mathtt{tr}(\rho_{\mathtt{x}}), \mathtt{y} : \mathtt{tr}(\rho_{\mathtt{y}}), \mathsf{i} : \mathtt{Int}, \rho_{\mathtt{x}} : \{\rho_{\mathtt{x}} \mapsto S'_{\mathtt{p}}\}, \rho_{\mathtt{y}} : \{\rho_{\mathtt{y}} \mapsto S_{\mathtt{q}}\}\rangle\\
&\qquad = \mathtt{y}[\mathtt{b}] \oplus \mathsf{add}(\mathsf{i}).\ \mathtt{x}[\mathtt{c}] \oplus \mathsf{turn}(\rho_{\mathtt{y}}).\ \mathtt{x}[\mathtt{c}]\ \&\ \mathsf{turn}(\rho_{\mathtt{y}}).\\
&\qquad\quad \mathsf{Produce}\langle \mathtt{x}, \mathtt{y}, \mathsf{i}+1, \rho_{\mathtt{x}}, \rho_{\mathtt{y}}\rangle;\ \{\rho_{\mathtt{x}} \mapsto S'_{\mathtt{p}}\} \otimes \{\rho_{\mathtt{y}} \mapsto S_{\mathtt{q}}\}\\
&\mathsf{P}\langle \mathtt{x} : \mathtt{tr}(\rho_{\mathtt{x}}), \mathtt{y} : \mathtt{tr}(\rho_{\mathtt{y}}), \rho_{\mathtt{x}} : \{\rho_{\mathtt{x}} \mapsto S_{\mathtt{p}}\}, \rho_{\mathtt{y}} : \{\rho_{\mathtt{y}} \mapsto S_{\mathtt{q}}\}\rangle\\
&\qquad = \mathtt{x}[\mathtt{c}] \oplus \mathsf{buffer}(\mathsf{pack}(\rho_{\mathtt{y}}, \mathtt{y}[\mathtt{b}])).\\
&\qquad\quad \mathsf{Produce}\langle \mathtt{x}, \mathtt{y}, 0, \rho_{\mathtt{x}}, \rho_{\mathtt{y}}\rangle; \{\rho_{\mathtt{x}} \mapsto S_{\mathtt{p}}\} \otimes \{\rho_{\mathtt{y}} \mapsto S_{\mathtt{q}}\}
\end{aligned}
$$

Process Representation – Consumer

$$S_{\mathtt{c}} = \mathtt{p}\&?\mathsf{buffer}(\exists[\rho_{\mathtt{q}}|\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}].\mathtt{tr}(\rho_{\mathtt{q}})).\mu\, \mathtt{t}.\mathtt{p}\&?\mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S'_{\mathtt{q}}\}).$$
$$\mathtt{p}\oplus!\mathsf{turn}(\{\rho_{\mathtt{q}} \mapsto S_{\mathtt{q}}\}).t$$
$$S'{}_{\mathtt{q}} = \mathtt{b}\oplus!\mathsf{request}(\mathtt{Str}).\mathtt{b}\&?\mathsf{send}(\mathtt{Int}).S_{\mathtt{q}}$$

Consumer Process:

$$\mathsf{Consume}\langle \mathrm{x} : \mathtt{tr}(\rho_{\mathrm{x}}), \mathrm{y} : \mathtt{tr}(\rho_{\mathrm{y}}), \rho_{\mathrm{x}} : \{\rho_{\mathrm{x}} \mapsto S'_{\mathtt{c}}\}\rangle =$$
$$\mathrm{x}[\mathtt{p}]\,\&\,\mathsf{turn}(\rho_{\mathrm{y}}).\mathrm{y}[\mathtt{b}] \oplus \mathsf{request}(r).\ \mathrm{y}[\mathtt{b}]\,\&\,\mathsf{send}(\mathsf{i}).\ \mathrm{x}[\mathtt{p}] \oplus \mathsf{turn}(\rho_{\mathrm{y}}).$$
$$\mathsf{Consume}\langle \mathrm{x}, \mathrm{y}, \rho_{\mathrm{x}}\rangle; \{\rho_{\mathrm{x}} \mapsto S'_{\mathtt{c}}\}$$

$$\mathsf{C}\langle \mathrm{x} : \mathtt{tr}(\rho_{\mathrm{x}}), \rho_{\mathrm{x}} : \{\rho_{\mathrm{x}} \mapsto S_{\mathtt{c}}\}\rangle =$$
$$\mathrm{x}[\mathtt{p}]\,\&\,\mathsf{buffer}(\mathsf{pack}(\rho_{\mathrm{y}}, \mathrm{y}[\mathtt{b}])).\ \mathsf{Consume}\langle \mathrm{x}, \mathrm{y}, \rho_{\mathrm{x}}\rangle; \{\rho_{\mathrm{x}} \mapsto S_{\mathtt{c}}\}$$

Process Representation – Buffer

$$S_{\mathtt{b}} = \ \mu\, \mathtt{t}.\mathtt{q}\&?\mathsf{add}(\mathtt{Int}).\mathtt{q}\&?\mathsf{request}(\mathtt{Str}).\mathtt{q}\oplus!\mathsf{send}(\mathtt{Int}).t$$

Buffer Process:

$$\begin{aligned} \mathsf{B}\langle \mathtt{x} : \mathtt{tr}(\rho_{\mathtt{x}}), \rho_{\mathtt{x}} : \{\rho_{\mathtt{x}} \mapsto S_{\mathtt{b}}\}\rangle \quad &= \quad \mathtt{x}[\mathtt{q}] \,\&\, \mathsf{add}(\mathsf{i}).\ \mathtt{x}[\mathtt{q}] \,\&\, \mathsf{request}(\mathtt{r}). \\ &\qquad \mathtt{x}[\mathtt{p}] \oplus \mathsf{send}(\mathsf{i}).\ \mathsf{B}\langle \mathtt{x}, \rho_{\mathtt{x}}\rangle;\ \ \{\rho_{\mathtt{x}} \mapsto S_{\mathtt{b}}\} \end{aligned}$$

Typing

Linearity is enforced via capabilities

$$\frac{\Delta; \Gamma \vdash P; C_1 \quad \Delta; \Gamma \vdash Q; C_2}{\Delta; \Gamma \vdash P \mid Q; C_1 \otimes C_2}\ \textsc{TPar}$$

$$\frac{\Delta; \Gamma, \Gamma' \vdash P; C \otimes C' \qquad (\Gamma' = \{\mathtt{s}[\mathtt{p}] : \mathtt{tr}(\rho_{\mathtt{p}}), \rho_{\mathtt{p}} : \{\rho_{\mathtt{p}} \mapsto S_{\mathtt{p}}\}\}_{\mathtt{p} \in I}, C' = \otimes_{\mathtt{p} \in I}\{\rho_{\mathtt{p}} \mapsto S_{\mathtt{p}}\}) \text{ complete}}{\Delta; \Gamma \vdash (\nu\, s : \Gamma')P; C}\ \textsc{TRes}$$

Theorem (Subject reduction)

If $\Delta; \Gamma \vdash P; C$ and $P \longrightarrow P'$, then there exist Γ' and C' such that $\Delta; \Gamma' \vdash P'; C'$ and $(\Gamma; C) \longrightarrow^* (\Gamma'; C')$.

Proof

The proof is by induction on the derivation of $P \longrightarrow P'$.

Other Approaches and Conclusion

We presented a new MPST system that allows sharing of resources

We presented a detailed account of the producer-consumer case study

We proved communication safety.

Future work:

- progress and deadlock freedom
- functional languages

Thank you!

Questions?

TSelP

$$\frac{\Gamma \vdash v : \mathtt{tr}(\rho'); \emptyset \qquad \Delta; \Gamma \vdash P; C \otimes \{\rho \mapsto S_j, \rho' \mapsto U\} \qquad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \qquad j \in I}{\Delta; \Gamma \vdash \mathsf{c}[\mathtt{p}] \oplus \langle l_j(\mathsf{pack}(\rho', v))\rangle.P; C \otimes \{\rho \mapsto \mathtt{p} \oplus_{i \in I} !l_i(\exists[\rho' | \{\rho' \mapsto U\}].\mathtt{tr}(\rho')).S_i, \rho' \mapsto U\}}$$

TBrP

$$\frac{\Delta; \Gamma, \mathtt{v_i} : \mathtt{tr}(\rho_{\mathtt{i}}), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i; C \otimes \{\rho \mapsto S_i\} \qquad \forall i \in I \qquad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma}{\Delta; \Gamma \vdash \mathsf{c}[\mathtt{p}] \&_{i \in I} \{l_i(\mathsf{pack}(\rho_i, \mathtt{v_i})). P_i\}; C \otimes \{\rho \mapsto \mathtt{p} \&_{i \in I} ?l_i(\exists[\rho_i | \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

References I

Crary, K., Walker, D., and Morrisett, G. (1999).
Typed memory management in a calculus of capabilities.
In *POPL*, pages 262–275. ACM.

Walker, D. and Morrisett, J. G. (2000).
Alias types for recursive data structures.
In *TIC*, LNCS, pages 177–206. Springer.