A Simple Concurrent Lambda Calculus For Encoding Session types

Jules Jacobs

Radboud University Nijmegen

mail@julesjacobs.com

Usual message passing:

- Stream of messages of fixed type
- e.g., Go, Rust

Usual message passing:

- Stream of messages of fixed type
- e.g., Go, Rust

Session types:

- Flexible message passing protocols
- Type of message can depend on the state of the protocol

Two flavours

π calculus: "everything is a channel"

- Elegant minimalist session types
  (Kobayashi, Dardha, Gay, Arslanagic, Perez, Caires, Pfenning, ...)

Two flavours

π calculus: "everything is a channel"

- Elegant minimalist session types (Kobayashi, Dardha, Gay, Arslanagic, Perez, Caires, Pfenning, ...)

λ calculus: "everything is a function"

- GV = linear λ calculus + channels + session types

Two flavours

π calculus: "everything is a channel"

- Elegant minimalist session types (Kobayashi, Dardha, Gay, Arslanagic, Perez, Caires, Pfenning, ...)

λ calculus: "everything is a function"

- GV = linear λ calculus + channels + session types
- Not aiming at a minimalist concurrent calculus
- Compiling concurrency away using CPS (Lindley, Morris, ...)

Two flavours

π calculus: "everything is a channel"

- Elegant minimalist session types (Kobayashi, Dardha, Gay, Arslanagic, Perez, Caires, Pfenning, ...)

λ calculus: "everything is a function"

- GV = linear λ calculus + channels + session types
- Not aiming at a minimalist concurrent calculus
- Compiling concurrency away using CPS (Lindley, Morris, ...)

This work: ƛ calculus = λ calculus + barriers

- Minimal concurrent extension of linear λ calculus
- Local encoding of session types as function types
- Simpler meta theory
- Minimal basis for extensions? (*e.g.*, priorities, sharing)

Session types in GV (Gay, Vasconcelos, Wadler, . . .)

$$
\begin{array}{l}
\textbf{let } c' = \textbf{fork}(\lambda c. \\
\qquad \textbf{let } (c, n) = \textbf{receive}(c) \textbf{ in} \\
\qquad \textbf{let } c = \textbf{send}(c,\ n \bmod 2 \equiv 0) \textbf{ in} \\
\qquad \textbf{close}(c)) \\
\textbf{let } c' = \textbf{send}(c', 3) \textbf{ in} \\
\textbf{let } (c', msg) = \textbf{receive}(c') \textbf{ in} \\
\textbf{close}(c')
\end{array}
$$

Session types in GV (Gay, Vasconcelos, Wadler, . . .)

$$
\begin{array}{l}
\textbf{let}\ c' : \textbf{!Int.\ ?Bool.\ End} = \textbf{fork}(\lambda c : \textbf{?Int.\ !Bool.\ End}. \\
\qquad \textbf{let}\ (c, n) = \textbf{receive}(c)\ \textbf{in} \\
\qquad \textbf{let}\ c = \textbf{send}(c,\ n \bmod 2 \equiv 0)\ \textbf{in} \\
\qquad \textbf{close}(c)) \\
\textbf{let}\ c' = \textbf{send}(c', 3)\ \textbf{in} \\
\textbf{let}\ (c', msg) = \textbf{receive}(c')\ \textbf{in} \\
\textbf{close}(c')
\end{array}
$$

Session types in GV (Gay, Vasconcelos, Wadler, . . .)

$$
\begin{array}{l}
\textbf{let}\ c' : \mathbf{!Int.\ ?Bool.\ End} = \textbf{fork}(\lambda c : \mathbf{?Int.\ !Bool.\ End}. \\
\qquad \textbf{let}\ (c : \mathbf{!Bool.\ End}, n : \mathbf{Int}) = \textbf{receive}(c)\ \textbf{in} \\
\qquad \textbf{let}\ c = \textbf{send}(c,\ n \bmod 2 \equiv 0)\ \textbf{in} \\
\qquad \textbf{close}(c)) \\
\textbf{let}\ c' : \mathbf{?Bool.\ End} = \textbf{send}(c', 3)\ \textbf{in} \\
\textbf{let}\ (c', msg) = \textbf{receive}(c')\ \textbf{in} \\
\textbf{close}(c')
\end{array}
$$

Session types in GV (Gay, Vasconcelos, Wadler, ...)

$$
\begin{array}{l}
\textbf{let}\ c' : \mathbf{!Int.\ ?Bool.\ End} = \textbf{fork}(\lambda c : \mathbf{?Int.\ !Bool.\ End}. \\
\qquad \textbf{let}\ (c : \mathbf{!Bool.\ End}, n : \mathbf{Int}) = \textbf{receive}(c)\ \textbf{in} \\
\qquad \textbf{let}\ c : \mathbf{End} = \textbf{send}(c,\ n \bmod 2 \equiv 0)\ \textbf{in} \\
\qquad \textbf{close}(c)) \\
\textbf{let}\ c' : \mathbf{?Bool.\ End} = \textbf{send}(c', 3)\ \textbf{in} \\
\textbf{let}\ (c' : \mathbf{End}, msg : \mathbf{Bool}) = \textbf{receive}(c')\ \textbf{in} \\
\textbf{close}(c')
\end{array}
$$

Session types in GV (Gay, Vasconcelos, Wadler, . . .)

Linear λ calculus: $\tau ::= \mathbf{0} \mid \mathbf{1} \mid \tau + \tau \mid \tau \times \tau \mid \tau \multimap \tau$

Session types: $s ::= {!}\tau.s \mid {?}\tau.s \mid s \oplus s \mid s \,\&\, s \mid \mathsf{End}$

$$
\begin{aligned}
\textbf{send} &: ({!}\tau.s) \times \tau \multimap s \\
\textbf{receive} &: ({?}\tau.s) \multimap (s \times \tau) \\
\textbf{tell}_{\mathbf{L}} &: (s_1 \oplus s_2) \multimap s_1 \\
\textbf{tell}_{\mathbf{R}} &: (s_1 \oplus s_2) \multimap s_2 \\
\textbf{ask} &: (s_1 \,\&\, s_2) \multimap (s_1 + s_2) \\
\textbf{close} &: \mathsf{End} \multimap \mathbf{1} \\
\textbf{fork} &: (s \multimap \mathbf{1}) \multimap \overline{s}
\end{aligned}
$$

$$
\begin{aligned}
\overline{{!}\tau.s} &\triangleq {?}\tau.\overline{s} \\
\overline{{?}\tau.s} &\triangleq {!}\tau.\overline{s} \\
\overline{s_1 \oplus s_2} &\triangleq \overline{s_1} \,\&\, \overline{s_2} \\
\overline{s_1 \,\&\, s_2} &\triangleq \overline{s_1} \oplus \overline{s_2} \\
\overline{\mathsf{End}} &\triangleq \mathsf{End}
\end{aligned}
$$

Session types in GV (Gay, Vasconcelos, Wadler, . . .)

Linear λ calculus: $\tau ::= \mathbf{0} \mid \mathbf{1} \mid \tau + \tau \mid \tau \times \tau \mid \tau \multimap \tau$

Session types: $s ::= !\tau.s \mid ?\tau.s \mid s \oplus s \mid s \,\&\, s \mid \mathsf{End}$

$$
\begin{aligned}
\textbf{send} &: (!\tau.s) \times \tau \multimap s \\
\textbf{receive} &: (?\tau.s) \multimap (s \times \tau) \\
\textbf{tell}_{\textbf{L}} &: (s_1 \oplus s_2) \multimap s_1 \\
\textbf{tell}_{\textbf{R}} &: (s_1 \oplus s_2) \multimap s_2 \\
\textbf{ask} &: (s_1 \,\&\, s_2) \multimap (s_1 + s_2) \\
\textbf{close} &: \mathsf{End} \multimap \mathbf{1} \\
\textbf{fork} &: (s \multimap \mathbf{1}) \multimap \overline{s}
\end{aligned}
$$

$$
\begin{aligned}
\overline{!\tau.s} &\triangleq ?\tau.\overline{s} \\
\overline{?\tau.s} &\triangleq !\tau.\overline{s} \\
\overline{s_1 \oplus s_2} &\triangleq \overline{s_1} \,\&\, \overline{s_2} \\
\overline{s_1 \,\&\, s_2} &\triangleq \overline{s_1} \oplus \overline{s_2} \\
\overline{\mathsf{End}} &\triangleq \mathsf{End}
\end{aligned}
$$

Session types in GV (Gay, Vasconcelos, Wadler, . . .)

Linear λ calculus: $\tau ::= \mathbf{0} \mid \mathbf{1} \mid \tau + \tau \mid \tau \times \tau \mid \tau \multimap \tau$

Session types: $s ::= !\tau.s \mid ?\tau.s \mid s \oplus s \mid s \,\&\, s \mid \mathsf{End}$

$$
\begin{aligned}
\textbf{send} &: (!\tau.s) \times \tau \multimap s \\
\textbf{receive} &: (?\tau.s) \multimap (s \times \tau) \\
\textbf{tell}_{\mathbf{L}} &: (s_1 \oplus s_2) \multimap s_1 \\
\textbf{tell}_{\mathbf{R}} &: (s_1 \oplus s_2) \multimap s_2 \\
\textbf{ask} &: (s_1 \,\&\, s_2) \multimap (s_1 + s_2) \\
\textbf{close} &: \mathsf{End} \multimap \mathbf{1} \\
\textbf{fork} &: (s \multimap \mathbf{1}) \multimap \overline{s}
\end{aligned}
$$

$$
\begin{aligned}
\overline{!\tau.s} &\triangleq ?\tau.\overline{s} \\
\overline{?\tau.s} &\triangleq !\tau.\overline{s} \\
\overline{s_1 \oplus s_2} &\triangleq \overline{s_1} \,\&\, \overline{s_2} \\
\overline{s_1 \,\&\, s_2} &\triangleq \overline{s_1} \oplus \overline{s_2} \\
\overline{\mathsf{End}} &\triangleq \mathsf{End}
\end{aligned}
$$

Session types in GV (Gay, Vasconcelos, Wadler, . . .)

Linear λ calculus: $\tau ::= \mathbf{0} \mid \mathbf{1} \mid \tau + \tau \mid \tau \times \tau \mid \tau \multimap \tau$

Session types: $s ::= {!}\tau.s \mid {?}\tau.s \mid s \oplus s \mid s \,\&\, s \mid \mathsf{End}$

$$\begin{aligned}
\textbf{send} &: ({!}\tau.s) \times \tau \multimap s \\
\textbf{receive} &: ({?}\tau.s) \multimap (s \times \tau) \\
\textbf{tell}_{\textbf{L}} &: (s_1 \oplus s_2) \multimap s_1 \\
\textbf{tell}_{\textbf{R}} &: (s_1 \oplus s_2) \multimap s_2 \\
\textbf{ask} &: (s_1 \,\&\, s_2) \multimap (s_1 + s_2) \\
\textbf{close} &: \mathsf{End} \multimap \mathbf{1} \\
\textbf{fork} &: (s \multimap \mathbf{1}) \multimap \overline{s}
\end{aligned}$$

$$\begin{aligned}
\overline{{!}\tau.s} &\triangleq {?}\tau.\overline{s} \\
\overline{{?}\tau.s} &\triangleq {!}\tau.\overline{s} \\
\overline{s_1 \oplus s_2} &\triangleq \overline{s_1} \,\&\, \overline{s_2} \\
\overline{s_1 \,\&\, s_2} &\triangleq \overline{s_1} \oplus \overline{s_2} \\
\overline{\mathsf{End}} &\triangleq \mathsf{End}
\end{aligned}$$

Session types in GV (Gay, Vasconcelos, Wadler, . . .)

Linear λ calculus: $\tau ::= \mathbf{0} \mid \mathbf{1} \mid \tau + \tau \mid \tau \times \tau \mid \tau \multimap \tau$

Session types: $s ::= !\tau.s \mid ?\tau.s \mid s \oplus s \mid s \,\&\, s \mid \mathsf{End}$

$$\begin{aligned}
\textbf{send} &: (!\tau.s) \times \tau \multimap s \\
\textbf{receive} &: (?\tau.s) \multimap (s \times \tau) \\
\textbf{tell}_{\mathbf{L}} &: (s_1 \oplus s_2) \multimap s_1 \\
\textbf{tell}_{\mathbf{R}} &: (s_1 \oplus s_2) \multimap s_2 \\
\textbf{ask} &: (s_1 \,\&\, s_2) \multimap (s_1 + s_2) \\
\textbf{close} &: \mathsf{End} \multimap \mathbf{1} \\
\textbf{fork} &: ((\alpha \multimap \beta) \multimap \mathbf{1}) \multimap (\beta \multimap \alpha)
\end{aligned}$$

$$\begin{aligned}
\overline{!\tau.s} &\triangleq ?\tau.\overline{s} \\
\overline{?\tau.s} &\triangleq !\tau.\overline{s} \\
\overline{s_1 \oplus s_2} &\triangleq \overline{s_1} \,\&\, \overline{s_2} \\
\overline{s_1 \,\&\, s_2} &\triangleq \overline{s_1} \oplus \overline{s_2} \\
\overline{\mathsf{End}} &\triangleq \mathsf{End}
\end{aligned}$$

“Session types”: $\lambda\!\!\!\!-$

$$\textbf{fork} : ((\alpha \multimap \beta) \multimap \mathbf{1}) \multimap (\beta \multimap \alpha)$$

“Session types”: $\lambda\!\!\!^-$

$$\textbf{fork} : ((\alpha \multimap \beta) \multimap \mathbf{1}) \multimap (\beta \multimap \alpha)$$

$$\textbf{fork}(\lambda x.\, E_1)$$

“Session types”: ƛ

$$\mathbf{fork} : ((\alpha \multimap \beta) \multimap \mathbf{1}) \multimap (\beta \multimap \alpha)$$

$$\mathbf{fork}(\lambda x.\ E_1)$$

$$\boxed{\alpha \multimap \beta}$$

“Session types”: $\lambdabar$

$$\textbf{fork} : ((\alpha \multimap \beta) \multimap \mathbf{1}) \multimap (\beta \multimap \alpha)$$

$$\textbf{let}\ x' = \textbf{fork}(\lambda \underset{\boxed{\alpha \multimap \beta}}{x}.\ E_1)\ \textbf{in}\ E_2$$

"Session types": $\lambdabar$

$$\textbf{fork} : ((\alpha \multimap \beta) \multimap \mathbf{1}) \multimap (\beta \multimap \alpha)$$

$$\textbf{let } \underset{\boxed{\beta \multimap \alpha}}{x'} = \textbf{fork}(\lambda \underset{\boxed{\alpha \multimap \beta}}{x}.\, E_1) \textbf{ in } E_2$$

“Session types”: ƛ

$$\mathbf{fork} : ((\alpha \multimap \beta) \multimap \mathbf{1}) \multimap (\beta \multimap \alpha)$$

$$\mathbf{let}\ \underbrace{x'}_{\beta \multimap \alpha} = \mathbf{fork}(\lambda \underbrace{x}_{\alpha \multimap \beta} .\, E_1)\ \mathbf{in}\ E_2$$

Single use!

$$\textbf{let}\ x' = \textbf{fork}(\lambda x.\ x\ 1)\ \textbf{in}\ 1 + x'\ ()$$

$$\textbf{let}\ x' = \textbf{fork}(\lambda x.\ x\ 1)\ \textbf{in}\ 1 + x'\ ()$$

$$\textbf{let } x' = \textbf{fork}(\lambda x.\, ())\ \textbf{in } 1 + 1$$

Operational semantics

$$\rho \in \mathsf{N} \xrightharpoonup{\text{fin}} \mathsf{Thread}(e) \mid \mathsf{Barrier}$$

Operational semantics

$$\rho \in \mathsf{N} \xrightharpoonup{\text{fin}} \mathsf{Thread}(e) \mid \mathsf{Barrier}$$

$$\{n \mapsto \mathsf{Thread}(K[\,e_1\,])\} \rightsquigarrow \{n \mapsto \mathsf{Thread}(K[\,e_2\,])\} \quad \text{if } e_1 \rightsquigarrow_{\text{pure}} e_2$$

$$\{n \mapsto \mathsf{Thread}(K[\,\mathbf{fork}(v)\,])\} \rightsquigarrow \left\{\begin{array}{l} n \mapsto \mathsf{Thread}(K[\ \langle k\rangle\ ]) \\ k \mapsto \mathsf{Barrier} \\ m \mapsto \mathsf{Thread}(v\ \ \langle k\rangle) \end{array}\right\} \qquad (\textbf{fork})$$

$$\left\{\begin{array}{l} n \mapsto \mathsf{Thread}(K_1[\ \langle k\rangle\ \ v_1\,]) \\ k \mapsto \mathsf{Barrier} \\ m \mapsto \mathsf{Thread}(K_2[\ \langle k\rangle\ \ v_2\,]) \end{array}\right\} \rightsquigarrow \left\{\begin{array}{l} n \mapsto \mathsf{Thread}(K_1[\,v_2\,]) \\ m \mapsto \mathsf{Thread}(K_2[\,v_1\,]) \end{array}\right\} \qquad (\textbf{sync})$$

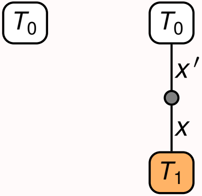
$$\{n \mapsto \mathsf{Thread}(())\} \rightsquigarrow \{\} \qquad (\text{exit})$$

$$\rho_1 \uplus \rho' \rightsquigarrow \rho_2 \uplus \rho' \quad \text{if } \rho_1 \rightsquigarrow \rho_2 \qquad (\text{frame})$$

$$\begin{aligned}
&\textbf{let } x' = \textbf{fork}(\lambda x.\ \textbf{let } (y, n) = x\ ()\\
&\qquad\qquad\qquad\qquad\quad \textbf{in } y\ (n \bmod 2 \equiv 0))\\
&\textbf{let } y' = \textbf{fork}(\lambda y.\ x'\ (y, 3))\\
&\textbf{in print}(y'\ ())
\end{aligned}$$

$\boxed{T_0}$

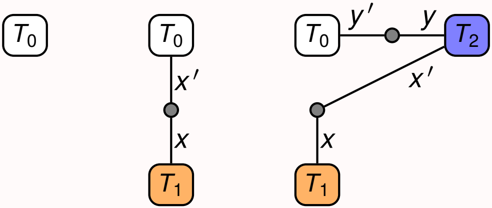
$$
\begin{aligned}
&\textbf{let } x' = \textbf{fork}(\lambda x.\ \textbf{let } (y, n) = x\ ()\\
&\qquad\qquad\qquad\qquad\quad \textbf{in } y\ (n \bmod 2 \equiv 0))\\
&\textbf{let } y' = \textbf{fork}(\lambda y.\ x'\ (y, 3))\\
&\textbf{in print}(y'\ ())
\end{aligned}
$$



$$\textbf{let } x' = \textbf{fork}(\lambda x.\ \textbf{let } (y, n) = x\ ()\ \textbf{in } y\ (n \bmod 2 \equiv 0))$$

$$\textbf{let } y' = \textbf{fork}(\lambda y.\ x'\ (y, 3))$$

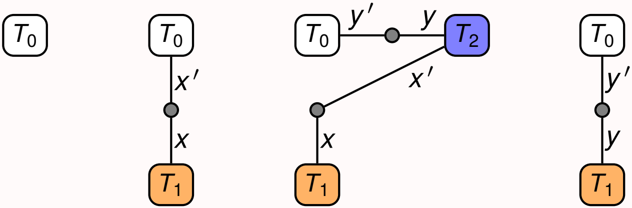
$$\textbf{in print}(y'\ ())$$



$$\textbf{let } x' = \textbf{fork}(\lambda x.\ \textbf{let } (y, n) = x\ ()\ \textbf{in } y\ (n \bmod 2 \equiv 0))$$

$$\textbf{let } y' = \textbf{fork}(\lambda y.\ x'\ (y, 3))$$

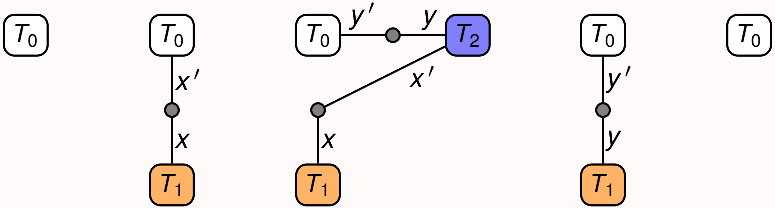
$$\textbf{in print}(y'\ ())$$



$$\textbf{let } x' = \textbf{fork}(\lambda x.\ \textbf{let } (y, n) = x\ ()\ \textbf{in } y\ (n \bmod 2 \equiv 0))$$

$$\textbf{let } y' = \textbf{fork}(\lambda y.\ x'\ (y, 3))$$

$$\textbf{in print}(y'\ ())$$



Channel operations as macros

$$\begin{aligned}
\mathbf{fork}_{\text{chan}}(f) &\triangleq \mathbf{fork}(f) \\
\mathbf{send}(c, x) &\triangleq \mathbf{fork}(\lambda c'.\ c\ (c', x)) \\
\mathbf{receive}(c) &\triangleq c\ () \\
\mathbf{close}(c) &\triangleq c\ ()
\end{aligned}$$

Channel operations as macros

$$\mathbf{fork}_{\mathrm{chan}}(f) \triangleq \mathbf{fork}(f)$$

$$\mathbf{send}(c, x) \triangleq \mathbf{fork}(\lambda c'.\, c\ (c', x))$$

$$\mathbf{receive}(c) \triangleq c\ ()$$

$$\mathbf{close}(c) \triangleq c\ ()$$

$$
\begin{array}{l}
\mathbf{let}\ c' = \mathbf{fork}(\lambda c. \\
\qquad \mathbf{let}\ (c, n) = \mathbf{receive}(c)\ \mathbf{in} \\
\qquad \mathbf{let}\ c = \mathbf{send}(c,\ n \bmod 2 \equiv 0)\ \mathbf{in} \\
\qquad \mathbf{close}(c)) \\
\mathbf{let}\ c' = \mathbf{send}(c', 3)\ \mathbf{in} \\
\mathbf{let}\ (c', msg) = \mathbf{receive}(c')\ \mathbf{in} \\
\mathbf{close}(c')
\end{array}
$$

Channel operations as macros

$$\begin{aligned}
\textbf{fork}_{\text{chan}}(f) &\triangleq \textbf{fork}(f) \\
\textbf{send}(c, x) &\triangleq \textbf{fork}(\lambda c'.\ c\ (c', x)) \\
\textbf{receive}(c) &\triangleq c\ () \\
\textbf{close}(c) &\triangleq c\ ()
\end{aligned}$$

$$\begin{aligned}
&\textbf{let}\ c' : [\![\ !\mathbf{Int}.\ ?\mathbf{Bool}.\ \mathbf{End}\ ]\!] = \textbf{fork}(\lambda c : [\![\ ?\mathbf{Int}.\ !\mathbf{Bool}.\ \mathbf{End}\ ]\!]. \\
&\qquad \textbf{let}\ (c, n) = \textbf{receive}(c)\ \textbf{in} \\
&\qquad \textbf{let}\ c = \textbf{send}(c,\ n \bmod 2 \equiv 0)\ \textbf{in} \\
&\qquad \textbf{close}(c)) \\
&\textbf{let}\ c' = \textbf{send}(c', 3)\ \textbf{in} \\
&\textbf{let}\ (c', msg) = \textbf{receive}(c')\ \textbf{in} \\
&\textbf{close}(c')
\end{aligned}$$

Channel operations as macros

$$\begin{aligned}
\textbf{fork}_{\text{chan}}(f) &\triangleq \textbf{fork}(f)\\
\textbf{send}(c, x) &\triangleq \textbf{fork}(\lambda c'.\ c\ (c', x))\\
\textbf{receive}(c) &\triangleq c\ ()\\
\textbf{close}(c) &\triangleq c\ ()
\end{aligned}$$

$$\begin{aligned}
&\textbf{let}\ c' : [\![\ \text{!Int. ?Bool. End}\ ]\!] = \textbf{fork}(\lambda c : [\![\ \text{?Int. !Bool. End}\ ]\!].\\
&\qquad \textbf{let}\ (c : [\![\ \text{!Bool. End}\ ]\!], n : \text{Int}) = \textbf{receive}(c)\ \textbf{in}\\
&\qquad \textbf{let}\ c : [\![\ \text{End}\ ]\!] = \textbf{send}(c,\ n \bmod 2 \equiv 0)\ \textbf{in}\\
&\qquad \textbf{close}(c))\\
&\textbf{let}\ c' : [\![\ \text{?Bool. End}\ ]\!] = \textbf{send}(c', 3)\ \textbf{in}\\
&\textbf{let}\ (c' : [\![\ \text{End}\ ]\!], msg : \text{Bool}) = \textbf{receive}(c')\ \textbf{in}\\
&\textbf{close}(c')
\end{aligned}$$

Session types as linear function types

$$
\begin{aligned}
[\![\mathrm{End}]\!] &\triangleq \mathbf{1} \multimap \mathbf{1} \\
[\![\mathbf{!}\tau.s]\!] &\triangleq [\![\overline{s}]\!] \times \tau \multimap \mathbf{1} \\
[\![\mathbf{?}\tau.s]\!] &\triangleq \mathbf{1} \multimap [\![s]\!] \times \tau \\
[\![s_1 \oplus s_2]\!], [\![s_1 \,\&\, s_2]\!] &\triangleq (\ldots)
\end{aligned}
$$

Session types as linear function types

$$
\begin{aligned}
[\![\mathsf{End}]\!] &\triangleq \mathbf{1} \multimap \mathbf{1} \\
[\![\mathbf{!}\tau.s]\!] &\triangleq [\![\overline{s}]\!] \times \tau \multimap \mathbf{1} \\
[\![\mathbf{?}\tau.s]\!] &\triangleq \mathbf{1} \multimap [\![s]\!] \times \tau \\
[\![s_1 \oplus s_2]\!], [\![s_1 \,\&\, s_2]\!] &\triangleq (\ldots)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{fork}_{\mathrm{chan}} &: ([\![s]\!] \multimap \mathbf{1}) \multimap [\![\overline{s}]\!] && \triangleq \lambda x.\, \mathbf{fork}(x) \\
\mathbf{close} &: [\![\mathsf{End}]\!] \multimap \mathbf{1} && \triangleq \lambda c.\, c\ () \\
\mathbf{send} &: [\![\mathbf{!}\tau.s]\!] \times \tau \multimap [\![s]\!] && \triangleq \lambda(c, x).\, \mathbf{fork}(\lambda c'.\, c\ (c', x)) \\
\mathbf{receive} &: [\![\mathbf{?}\tau.s]\!] \multimap [\![s]\!] \times \tau && \triangleq \lambda c.\, c\ () \\
\mathbf{tell_L}, \mathbf{tell_R}, \mathbf{ask} &: (\ldots) && \triangleq (\ldots)
\end{aligned}
$$

Session types as linear function types

$$
\begin{aligned}
[\![\mathrm{End}]\!] &\triangleq \mathbf{1} \multimap \mathbf{1} \\
[\![!\tau.s]\!] &\triangleq [\![\overline{s}]\!] \times \tau \multimap \mathbf{1} \\
[\![?\tau.s]\!] &\triangleq \mathbf{1} \multimap [\![s]\!] \times \tau \\
[\![s_1 \oplus s_2]\!], [\![s_1 \,\&\, s_2]\!] &\triangleq (\ldots)
\end{aligned}
$$

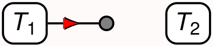
$$
\begin{aligned}
\mathbf{fork}_{\mathrm{chan}} &: ([\![s]\!] \multimap \mathbf{1}) \multimap [\![\overline{s}]\!] && \triangleq \lambda x.\, \mathbf{fork}(x) \\
\mathbf{close} &: [\![\mathrm{End}]\!] \multimap \mathbf{1} && \triangleq \lambda c.\, c\ () \\
\mathbf{send} &: [\![!\tau.s]\!] \times \tau \multimap [\![s]\!] && \triangleq \lambda(c, x).\, \mathbf{fork}(\lambda c'.\, c\ (c', x)) \\
\mathbf{receive} &: [\![?\tau.s]\!] \multimap [\![s]\!] \times \tau && \triangleq \lambda c.\, c\ () \\
\mathbf{tell_L}, \mathbf{tell_R}, \mathbf{ask} &: \quad (\ldots) && \triangleq (\ldots)
\end{aligned}
$$

Theorem. If GV program is well-typed, then macro expanded $\lambda\!\!\!^{-}$ program is well-typed

Theorem. Macro expanded $\lambda\!\!\!^{-}$ program simulates GV program

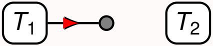
Deadlock freedom: linearity

$\textbf{let } x' = \textbf{fork}(\lambda x.\ ())\ \textbf{in } x'\ 0$ Deadlock!



Deadlock freedom: linearity

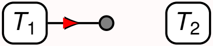
$$\textbf{let } x' = \textbf{fork}(\lambda x.\, ())\ \textbf{in } x'\ 0$$ Deadlock!



Ruled out by linear typing

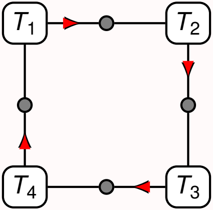
Deadlock freedom: linearity

$\textbf{let}\ x' = \textbf{fork}(\lambda x.\ ())\ \textbf{in}\ x'\ 0$ Deadlock!

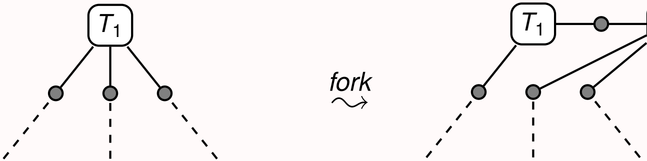


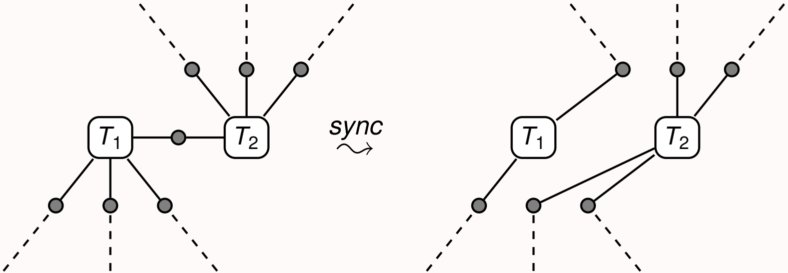
Ruled out by linear typing

But what about cycles?



Deadlock freedom: acyclicity





Mechanized proofs in Coq

Meta theory of ƛ + recursive types + non-linear types

- Global progress:
  $(e : 1) \wedge \{0 \mapsto e\} \leadsto \rho \implies \rho \text{ can step} \vee \rho = \{\}$
- Partial deadlock freedom
- Memory leak freedom
- Size $\approx \frac{1}{2}$ earlier GV mechanization

Mechanized proofs in Coq

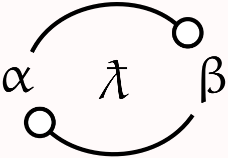
Meta theory of $\lambda\!\!\!\!^-$ + recursive types + non-linear types

- Global progress:
  $(e : 1) \wedge \{0 \mapsto e\} \rightsquigarrow \rho \implies \rho \text{ can step} \vee \rho = \{\}$
- Partial deadlock freedom
- Memory leak freedom
- Size $\approx \frac{1}{2}$ earlier GV mechanization

Session types in $\lambda\!\!\!\!^-$

- Compiler from GV to $\lambda\!\!\!\!^-$
- Proof that output $\lambda\!\!\!\!^-$ program is well-typed
- Proof that output $\lambda\!\!\!\!^-$ program simulates GV program

$$\underbrace{\textbf{fork} : ((\alpha \multimap \beta) \multimap \mathbf{1}) \multimap (\beta \multimap \alpha)}_{\text{Session types distilled}}$$



Questions?

mail@julesjacobs.com