DLπ

A Linear π-Calculus with Dependent Types in Agda

Luca Ciccone   Luca Padovani

5 June 2020

Motivation

Session types = linear channels + pairs + sums [Dardha et al., 2017]

- Session = chain of one-shot communications
- Messages are **pairs**: payload + continuation channel

Motivation

Session types = linear channels + pairs + sums [Dardha et al., 2017]

- Session = chain of one-shot communications
- Messages are **pairs**: payload + continuation channel

Dependent session types = linear channels + **dependent** pairs?

- Value-dependent session types [Toninho et al., 2011]
- Liquid Pi [Griffith and Gunter, 2013]
- Dependent protocols in Idris [Brady, 2017]
- Dependent session-typed processes [Toninho and Yoshida, 2018]
- Label-dependent session types [Thiemann and Vasconcelos, 2020]

Motivation

Session types = linear channels + pairs + sums [Dardha et al., 2017]

- Session = chain of one-shot communications
- Messages are **pairs**: payload + continuation channel

Dependent session types = linear channels + **dependent** pairs?

- Value-dependent session types [Toninho et al., 2011]
- Liquid Pi [Griffith and Gunter, 2013]
- Dependent protocols in Idris [Brady, 2017]
- Dependent session-typed processes [Toninho and Yoshida, 2018]
- Label-dependent session types [Thiemann and Vasconcelos, 2020]

Yes!

Goal

Develop a minimal, Agda-based, linear π-calculus with dependent pairs ($\mathtt{DL}\pi$) in which dependent session types can be encoded

Goal

Develop a minimal, Agda-based, linear π-calculus with dependent pairs ($\mathtt{DL}\pi$) in which dependent session types can be encoded

Analogous expressiveness of Brady [2017]'s DSL and Toninho and Yoshida [2018]'s calculus, but:

- different type structure
  - linear channels + dependent pairs instead of session types
- Agda mechanisation of the metatheory
- we lift from Agda all the machinery related to dependent types
  - **computation** of data-dependent types and processes

DLπ - Processes

Terms	$M, N ::=$	p	pure term
	$\mid$	u	name
	$\mid$	M, N	pair
Processes	$P, Q ::=$	idle	inaction
	$\mid$	$u(x).P$	input
	$\mid$	$u\langle M\rangle$	output
	$\mid$	$\mathsf{let}\ x, y = M\ \mathsf{in}\ P$	pair splitting
	$\mid$	$P \mid Q$	parallel composition
	$\mid$	$(a)P$	restriction
	$\mid$	$*P$	replication

DLπ - Processes

Terms	$M, N ::=$	p	pure term
	$\mid$	u	name
	$\mid$	M, N	pair
Processes	$P, Q ::=$	idle	inaction
	$\mid$	$u(x).P$	input
	$\mid$	$u\langle M\rangle$	output
	$\mid$	$\mathsf{let}\ x, y = M\ \mathsf{in}\ P$	pair splitting
	$\mid$	$P \mid Q$	parallel composition
	$\mid$	$(a)P$	restriction
	$\mid$	$*P$	replication

- Pure terms injected in DLπ

DLπ - Types

Domains $A, B \in \mathcal{A}$ pure types
$\sigma, \rho \in \{0, 1, \omega\}$ multiplicities

Types $t, s ::= A$ pure type
$\mid\ {}^{\sigma,\rho}[t]$ channel type
$\mid\ \Sigma(x : t)s$ linear dependent pair

DLπ - Types

$$
\begin{array}{llcll}
\textbf{Domains} & A, B & \in & \mathcal{A} & \text{pure types} \\
 & \sigma, \rho & \in & \{0, 1, \omega\} & \text{multiplicities} \\
 & & & & \\
\textbf{Types} & t, s & ::= & A & \text{pure type} \\
 & & \mid & {}^{\sigma,\rho}[t] & \text{channel type} \\
 & & \mid & \Sigma(x : t)s & \text{linear dependent pair}
\end{array}
$$

- Multiplicities can be combined with a *sum*

DLπ - Types

$$
\begin{array}{llcll}
\textbf{Domains} & A, B & \in & \mathcal{A} & \text{pure types} \\
 & \sigma, \rho & \in & \{0, 1, \omega\} & \text{multiplicities} \\
\\
\textbf{Types} & t, s & ::= & A & \text{pure type} \\
 & & | & {}^{\sigma,\rho}[t] & \text{channel type} \\
 & & | & \Sigma(x : t)s & \text{linear dependent pair}
\end{array}
$$

- Multiplicities can be combined with a *sum*
  $1 + 1 = \omega$, $\omega + \omega = \omega$

DLπ - Types

$$
\begin{array}{llcll}
\textbf{Domains} & A, B & \in & \mathcal{A} & \text{pure types} \\
 & \sigma, \rho & \in & \{0, 1, \omega\} & \text{multiplicities} \\
 & & & & \\
\textbf{Types} & t, s & ::= & A & \text{pure type} \\
 & & \mid & {}^{\sigma,\rho}[t] & \text{channel type} \\
 & & \mid & \Sigma(x : t)s & \text{linear dependent pair}
\end{array}
$$

- Multiplicities can be combined with a *sum*
  $1 + 1 = \omega$, $\omega + \omega = \omega$
- Types can be combined

DLπ - Types

$$
\begin{array}{llcll}
\textbf{Domains} & A, B & \in & \mathcal{A} & \text{pure types} \\
 & \sigma, \rho & \in & \{0, 1, \omega\} & \text{multiplicities} \\
 & & & & \\
\textbf{Types} & t, s & ::= & A & \text{pure type} \\
 & & \mid & {}^{\sigma,\rho}[t] & \text{channel type} \\
 & & \mid & \Sigma(x : t)s & \text{linear dependent pair}
\end{array}
$$

- Multiplicities can be combined with a *sum*
  $1 + 1 = \omega$, $\omega + \omega = \omega$
- Types can be combined
  - Partial operation (in Agda represented as a **relation**)

DLπ - Types

$$
\begin{array}{llcll}
\textbf{Domains} & A, B & \in & \mathcal{A} & \text{pure types} \\
 & \sigma, \rho & \in & \{0, 1, \omega\} & \text{multiplicities} \\
 & & & & \\
\textbf{Types} & t, s & ::= & A & \text{pure type} \\
 & & \mid & {}^{\sigma,\rho}[t] & \text{channel type} \\
 & & \mid & \Sigma(x : t)s & \text{linear dependent pair}
\end{array}
$$

- Multiplicities can be combined with a *sum*
  $1 + 1 = \omega$, $\omega + \omega = \omega$
- Types can be combined
  - Partial operation (in Agda represented as a **relation**)
- Pairs are **linear**

Parallel composition

$$\dfrac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}\ \text{T-PAR}$$

Parallel composition

$$\frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}\ \text{T-PAR}$$

- Resources are combined

$$\Gamma_1 + \Gamma_2 = \Gamma_1, \Gamma_2 \qquad \mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$$

$$(u : t, \Gamma_1) + (u : s, \Gamma_2) = (u : t + s), (\Gamma_1 + \Gamma_2)$$

Parallel composition

$$\frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}\ \textsc{T-Par}$$

- Resources are combined

$$\Gamma_1 + \Gamma_2 = \Gamma_1, \Gamma_2 \qquad \mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$$
$$(u : t, \Gamma_1) + (u : s, \Gamma_2) = (u : t + s), (\Gamma_1 + \Gamma_2)$$

- Sum of types

$$^{1,0}[t] + {}^{0,1}[t] = {}^{1,1}[t]$$

Receive - Send

$$\textsc{T-Input}\quad \frac{\Gamma_1 \vdash u : {}^{1,0}[t] \qquad \Gamma_2, x : t \vdash P}{\Gamma_1 + \Gamma_2 \vdash u(x).P}$$

$$\textsc{T-Output}\quad \frac{\Gamma_1 \vdash u : {}^{0,1}[t] \qquad \Gamma_2 \vdash M : t}{\Gamma_1 + \Gamma_2 \vdash u\langle M\rangle}$$

- Resources must be combined

Receive - Send

$$\dfrac{\Gamma_1 \vdash u : {}^{1,0}[t] \qquad \Gamma_2, x : t \vdash P}{\Gamma_1 + \Gamma_2 \vdash u(x).P}\ \textsc{T-Input} \qquad \dfrac{\Gamma_1 \vdash u : {}^{0,1}[t] \qquad \Gamma_2 \vdash M : t}{\Gamma_1 + \Gamma_2 \vdash u\langle M \rangle}\ \textsc{T-Output}$$

- Resources must be combined
- Linear channels
  - Input $1, 0$
  - Output $0, 1$

Receive - Send

$$\frac{\Gamma_1 \vdash u : {}^{1,0}[t] \qquad \Gamma_2, x : t \vdash P}{\Gamma_1 + \Gamma_2 \vdash u(x).P}\ \text{T-INPUT}$$

$$\frac{\Gamma_1 \vdash u : {}^{0,1}[t] \qquad \Gamma_2 \vdash M : t}{\Gamma_1 + \Gamma_2 \vdash u\langle M \rangle}\ \text{T-OUTPUT}$$

- Resources must be combined
- Linear channels
  - Input $1, 0$
  - Output $0, 1$

Dependent Pairs

$$\frac{\Gamma_1 \vdash M : t \qquad \Gamma_2 \vdash N : s\{[\![M]\!]/x\}}{\Gamma_1 + \Gamma_2 \vdash M, N : \Sigma(x : t)s}\ \text{T-PAIR}$$

Dependent Pairs

$$\frac{\Gamma_1 \vdash M : t \qquad \Gamma_2 \vdash N : s\{[\![M]\!]/x\}}{\Gamma_1 + \Gamma_2 \vdash M, N : \Sigma(x : t)s}\ \text{T-PAIR}$$

- Filter function introduced

Dependent Pairs

$$\frac{\Gamma_1 \vdash M : t \qquad \Gamma_2 \vdash N : s\{[\![M]\!]/x\}}{\Gamma_1 + \Gamma_2 \vdash M, N : \Sigma(x : t)s}\ \text{T-PAIR}$$

- Filter function introduced
  - Map from DLπ terms to pure terms

$$[\![p]\!] = p \qquad [\![x]\!] = x \qquad [\![a]\!] = \mathtt{tt} \qquad [\![M, N]\!] = [\![M]\!], [\![N]\!]$$

Dependent Pairs

$$\frac{\Gamma_1 \vdash M : t \qquad \Gamma_2 \vdash N : s\{[\![M]\!]/x\}}{\Gamma_1 + \Gamma_2 \vdash M, N : \Sigma(x : t)s} \ \text{T-PAIR}$$

- Filter function introduced
  - Map from DLπ terms to pure terms

$$[\![p]\!] = p \qquad [\![x]\!] = x \qquad [\![a]\!] = \mathtt{tt} \qquad [\![M, N]\!] = [\![M]\!], [\![N]\!]$$

- From the point of view of types

$$[\![A]\!] = A \qquad [\![^{\sigma,\rho}[t]]\!] = \top \qquad [\![\Sigma(x : t)s]\!] = \Sigma(x : [\![t]\!])[\![s]\!]$$

Dependent Pairs

$$\frac{\Gamma_1 \vdash M : t \qquad \Gamma_2 \vdash N : s\{\llbracket M \rrbracket / x\}}{\Gamma_1 + \Gamma_2 \vdash M, N : \Sigma(x : t)s} \; \text{T-PAIR}$$

- Filter function introduced
  - Map from DLπ terms to pure terms

$$\llbracket p \rrbracket = p \qquad \llbracket x \rrbracket = x \qquad \llbracket a \rrbracket = \mathtt{tt} \qquad \llbracket M, N \rrbracket = \llbracket M \rrbracket, \llbracket N \rrbracket$$

- From the point of view of types

$$\llbracket A \rrbracket = A \qquad \llbracket^{\sigma,\rho}[t] \rrbracket = \top \qquad \llbracket \Sigma(x : t)s \rrbracket = \Sigma(x : \llbracket t \rrbracket)\llbracket s \rrbracket$$

- Channels are erased

Dependent Pairs

$$\frac{\Gamma_1 \vdash M : t \qquad \Gamma_2 \vdash N : s\{[\![M]\!]/x\}}{\Gamma_1 + \Gamma_2 \vdash M, N : \Sigma(x : t)s} \text{ T-PAIR}$$

- Filter function introduced
  - Map from DLπ terms to pure terms

$$[\![p]\!] = p \qquad [\![x]\!] = x \qquad [\![a]\!] = \mathtt{tt} \qquad [\![M, N]\!] = [\![M]\!], [\![N]\!]$$

- From the point of view of types

$$[\![A]\!] = A \qquad [\![{}^{\sigma,\rho}[t]]\!] = \top \qquad [\![\Sigma(x : t)s]\!] = \Sigma(x : [\![t]\!])[\![s]\!]$$

- Channels are erased
  - No dependency on channels

Multiplicities

```
data Mult : Set where
  #0 #1 #ω : Mult

data MSplit : Mult → Mult → Mult → Set
```

Multiplicities

```
data Mult : Set where
  #0 #1 #ω : Mult

data MSplit : Mult → Mult → Mult → Set
```

- Types for defining combinations
  - Combination of two multiplicities (*sum*)
- MSplit σ σ_1 σ_2 is inhabited if and only if $\sigma = \sigma_1 + \sigma_2$
- Relations lifted at type level

Types

```
mutual
  data Type : Set₁ where
    Pure  : Set → Type
    Chan  : Mult → Mult → Type → Type
    Pair  : (t : Type) → (⟦ t ⟧ → Type) → Type

  ⟦_⟧ : Type → Set
  ⟦ Pure A ⟧     = A
  ⟦ Chan _ _ _ ⟧ = ⊤
  ⟦ Pair t f ⟧   = Σ ⟦ t ⟧ λ x → ⟦ f x ⟧
```

Types

```
mutual
  data Type : Set₁ where
    Pure  : Set → Type
    Chan  : Mult → Mult → Type → Type
    Pair  : (t : Type) → (⟦ t ⟧ → Type) → Type

  ⟦_⟧ : Type → Set
  ⟦ Pure A ⟧      = A
  ⟦ Chan _ _ _ ⟧  = ⊤
  ⟦ Pair t f ⟧    = Σ ⟦ t ⟧ λ x → ⟦ f x ⟧
```

- Inductive-Recursive definition
- Interpretation function
- Higher level of Set

Representation of names

- Intrinsically typed terms and processes

Representation of names

- Intrinsically typed terms and processes
    - Instances are type derivations

Representation of names

- Intrinsically typed terms and processes
  - Instances are type derivations

```
data Name : ℕ → Context → (t : Type) → ⟦ t ⟧ → Set₁ where
  here : ∀{ Γ t p } → CNull Γ → Name zero (t # p :: Γ) t p
  next : ∀{ k Γ t s p q } → TNull s → Name k Γ t p →
         Name (suc k) (s # q :: Γ) t p
```

Representation of names

- Intrinsically typed terms and processes
  - Instances are type derivations

```
data Name : ℕ → Context → (t : Type) → ⟦ t ⟧ → Set₁ where
  here : ∀{ Γ t p } → CNull Γ → Name zero (t # p :: Γ) t p
  next : ∀{ k Γ t s p q } → TNull s → Name k Γ t p →
         Name (suc k) (s # q :: Γ) t p
```

- de Bruijn representation
- Association to Agda value
- Type isomorphic to natural numbers

Representation of terms

$$\frac{\Gamma_1 \vdash M : t \qquad \Gamma_2 \vdash N : s\{\llbracket M \rrbracket / x\}}{\Gamma_1 + \Gamma_2 \vdash M, N : \Sigma(x : t)s}\ \textsc{T-Pair}$$

```
data Term : Context → (t : Type) → ⟦ t ⟧ → Set₁ where
  name : ∀{ k Γ t p } → Name k Γ t p → Term Γ t p
  pure  : ∀{ Γ A } → CNull Γ → (p : A) → Term Γ (Pure A) p
  pair  : ∀{ Γ Γ₁ Γ₂ t f p q } → CSplit Γ Γ₁ Γ₂ →
          Term Γ₁ t p → Term Γ₂ (f p) q → Term Γ (Pair t f) (p , q)
```

Parallel composition

$$\textsc{T-Par}\ \frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}$$

```
data Process : Context → Set₁ where
  Par : ∀{ Γ Γ₁ Γ₂ } → CSplit Γ Γ₁ Γ₂ →
        Process Γ₁ → Process Γ₂ → Process Γ
```

Parallel composition

$$\frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}\ \text{T-PAR}$$

```
data Process : Context → Set₁ where
  Par : ∀{ Γ Γ₁ Γ₂ } → CSplit Γ Γ₁ Γ₂ →
        Process Γ₁ → Process Γ₂ → Process Γ
```

- Resources combined

Receive

T-INPUT

$$\frac{\Gamma_1 \vdash u : {}^{1,0}[t] \qquad \Gamma_2, x : t \vdash P}{\Gamma_1 + \Gamma_2 \vdash u(x).P}$$

$$\begin{aligned}
\mathsf{Recv} : \forall\{\, \Gamma\ \Gamma_1\ \Gamma_2\ t \,\} &\to \mathsf{CSplit}\ \Gamma\ \Gamma_1\ \Gamma_2 \to \\
&\mathsf{Term}\ \Gamma_1\ (\mathsf{Chan}\ \#1\ \#0\ t)\ _ \to \\
&((x : [\![\ t\ ]\!]) \to \mathsf{Process}\ (t\ \#\ x :: \Gamma_2)) \to \mathsf{Process}\ \Gamma
\end{aligned}$$

Receive

$$\frac{\Gamma_1 \vdash u : {}^{1,0}[t] \qquad \Gamma_2, x : t \vdash P}{\Gamma_1 + \Gamma_2 \vdash u(x).P}\ \text{T-INPUT}$$

$$\begin{aligned}\mathsf{Recv} : \forall\{\ \Gamma\ \Gamma_1\ \Gamma_2\ t\ \} &\to \mathsf{CSplit}\ \Gamma\ \Gamma_1\ \Gamma_2 \to \\ &\mathsf{Term}\ \Gamma_1\ (\mathsf{Chan}\ \#1\ \#0\ t)\ \_ \to \\ &((x : [\![\ t\ ]\!]) \to \mathsf{Process}\ (t\ \#\ x :: \Gamma_2)) \to \mathsf{Process}\ \Gamma\end{aligned}$$

- Agda value of a channel omitted (it is tt)
- Continuation inside a function
  - Agda value of the received message **stored in context**

Pair splitting

$$\frac{\Gamma_1 \vdash M : \Sigma(x : t)s \qquad \Gamma_2, x : t, y : s \vdash P}{\Gamma_1 + \Gamma_2 \vdash \textbf{let } x, y = M \textbf{ in } P}\ \text{T-LET}$$

$$\begin{aligned}\text{Let} : \forall\{\ \Gamma\ \Gamma_1\ \Gamma_2\ t\ f\ p\ q\ \} \to\ & \text{CSplit}\ \Gamma\ \Gamma_1\ \Gamma_2 \to \\ & \text{Term}\ \Gamma_1\ (\text{Pair}\ t\ f)\ (p\ ,\ q) \to \\ & \text{Process}\ (t\ \#\ p\ ::\ f\ p\ \#\ q\ ::\ \Gamma_2) \to \text{Process}\ \Gamma\end{aligned}$$

Pair splitting

$$\frac{\Gamma_1 \vdash M : \Sigma(x : t)s \qquad \Gamma_2, x : t, y : s \vdash P}{\Gamma_1 + \Gamma_2 \vdash \textbf{let } x, y = M \textbf{ in } P}\ \text{T-LET}$$

```
Let : ∀{ Γ Γ₁ Γ₂ t f p q } → CSplit Γ Γ₁ Γ₂ →
      Term Γ₁ (Pair t f) (p , q) →
      Process (t # p :: f p # q :: Γ₂) → Process Γ
```

- Let reflects the pair construction

Example 1: successor of a number

$$Q_1(u) \stackrel{\text{def}}{=} u(x, y).y\langle x + 1\rangle$$

Example 1: successor of a number

$$Q_1(u) \stackrel{\text{def}}{=} u(x, y).y\langle x + 1\rangle$$

- Agda code

  $\mathsf{t}_1 : \mathsf{Type}$
  $\mathsf{t}_1 = \mathsf{Chan}\ \#1\ \#0\ (\mathsf{Pair}\ (\mathsf{Pure}\ \mathbb{N})\ \lambda\ \_ \rightarrow \mathsf{Chan}\ \#0\ \#1\ (\mathsf{Pure}\ \mathbb{N}))$

Example 1: successor of a number

$$Q_1(u) \stackrel{\text{def}}{=} u(x, y).y\langle x + 1\rangle$$

- Agda code

```
t₁ : Type
t₁ = Chan #1 #0 (Pair (Pure ℕ) λ _ → Chan #0 #1 (Pure ℕ))

Q₁ : Process (t₁ # _ :: [])
Q₁ = Recv (L [])   (name (here [])) λ (x , _) →
     Let  (L [])   (name (here [])) $
     Send (R L []) (name (here [])) (pure (P :: []) (x + 1))
```

- Resource are distributed where needed
- Pattern matching on the Agda value of the received message

Example 2: predecessor of a number

$$Q_2(u) \stackrel{\text{def}}{=} u(x, v).v(y, w).w\langle \mathsf{pred}(x, y)\rangle$$

Example 2: predecessor of a number

$$Q_2(u) \stackrel{\text{def}}{=} u(x, v).v(y, w).w\langle \mathrm{pred}(x, y)\rangle$$

$$\begin{aligned}
&\mathsf{t}_2 : \mathsf{Type}\\
&\mathsf{t}_2 = \mathsf{Chan}\ \#1\ \#0\ (\mathsf{Pair}\ (\mathsf{Pure}\ \mathbb{N})\ \lambda\ x \rightarrow\\
&\qquad \mathsf{Chan}\ \#1\ \#0\ (\mathsf{Pair}\ (\mathsf{Pure}\ (x \not\equiv 0))\ \lambda\ _ \rightarrow \mathsf{Chan}\ \#0\ \#1\ (\mathsf{Pure}\ \mathbb{N})))
\end{aligned}$$

Example 2: predecessor of a number

$$Q_2(u) \stackrel{\text{def}}{=} u(x, v).v(y, w).w\langle \text{pred}(x, y)\rangle$$

```
t₂ : Type
t₂ = Chan #1 #0 (Pair (Pure ℕ) λ x →
     Chan #1 #0 (Pair (Pure (x ≢ 0)) λ _ → Chan #0 #1 (Pure ℕ)))

Q₂ : Process (t₂ # _ :: [])
Q₂ = Recv (L [])     (name (here [])) λ (x , _) →
     Let  (L [])     (name (here [])) $
     Recv (R L [])   (name (here [])) λ (y , _) →
     Let  (L R [])   (name (here [])) $
     Send (R L R []) (name (here [])) (pure (P :: P :: []) (pred x y))
```

Example 3: combining successor and predecessor

$$Q_3 \stackrel{\text{def}}{=} *a(x, y).F(x, y)$$

$$F(\text{true}, \; y) = \; Q_1(y)$$
$$F(\text{false}, \; y) = \; Q_2(y)$$

Example 3: combining successor and predecessor

$$Q_3 \stackrel{\text{def}}{=} *a(x, y).F(x, y)$$

$$\begin{aligned} F(\mathsf{true}\,,\ y) &= \ Q_1(y) \\ F(\mathsf{false},\ y) &= \ Q_2(y) \end{aligned}$$

$\mathsf{t}_3 \ : \ \mathsf{Type}$
$\mathsf{t}_3 = \mathsf{Chan}\ \#\omega\ \#0\ (\mathsf{Pair}\ (\mathsf{Pure\ Bool})\ (\lambda\ b \to \mathsf{if}\ b\ \mathsf{then}\ \mathsf{t}_1\ \mathsf{else}\ \mathsf{t}_2))$

Example 3: combining successor and predecessor

$$Q_3 \stackrel{\text{def}}{=} *a(x,y).F(x,y)$$

$$\begin{aligned} F(\mathsf{true}\,,\; y) &= \; Q_1(y) \\ F(\mathsf{false},\; y) &= \; Q_2(y) \end{aligned}$$

```
t₃ : Type
t₃ = Chan #ω #0 (Pair (Pure Bool) (λ b → if b then t₁ else t₂))

Q₃ : Process (t₃ # _ :: [])
Q₃ = Rep (chan sc1 sc0 :: []) $
       Recv (L []) (name (here [])) λ
       { (true , _) → Let (L []) (name (here [])) (weaken Q₁)
       ; (false , _) → Let (L []) (name (here [])) (weaken Q₂) }
```

- Explicit case analysis inside the lambda
- Weakened processes

Plain session types

$$\begin{array}{rcl} T, S & ::= & \textcolor{blue}{\mathsf{end}} \mid ?m.T \mid !m.T \mid T \mathbin{\&} S \mid T \oplus S \\ m & ::= & A \mid T \end{array}$$

Plain session types

$$
\begin{array}{rcl}
T, S & ::= & \textcolor{blue}{\mathsf{end}} \mid ?m.T \mid !m.T \mid T \,\&\, S \mid T \oplus S \\
m & ::= & A \mid T
\end{array}
$$

- Branches and Choices involve a single bit
  - We assume that it is represented by a Boolean value
- Branches and Choices introduce a simple dependency

Encoding [Dardha et al., 2017]

$$
\begin{aligned}
\llbracket \mathsf{end} \rrbracket &= {}^{0,0}[\top] \\
\llbracket ?m.T \rrbracket &= {}^{1,0}[\llbracket m \rrbracket \times \llbracket T \rrbracket] \\
\llbracket !m.T \rrbracket &= {}^{0,1}[\llbracket m \rrbracket \times \llbracket \overline{T} \rrbracket] \\
\llbracket T \,\&\, S \rrbracket &= {}^{1,0}[\Sigma(x : \mathsf{Bool})\ \mathsf{if}\ x\ \mathsf{then}\ \llbracket T \rrbracket\ \mathsf{else}\ \llbracket S \rrbracket] \\
\llbracket T \oplus S \rrbracket &= {}^{0,1}[\Sigma(x : \mathsf{Bool})\ \mathsf{if}\ x\ \mathsf{then}\ \llbracket \overline{T} \rrbracket\ \mathsf{else}\ \llbracket \overline{S} \rrbracket]
\end{aligned}
$$

- Encoding adapted to DLπ: dependent pairs subsume sums
- One-shot communications
  - Payloads have a continuation associated
- Continuations on send operations are dualized
- $\mathsf{if}\ x\ \mathsf{then}\ t\ \mathsf{else}\ s$ is a term at the functional layer

Value-dependent session types [Toninho et al., 2011]

$$T, S \ ::= \ \cdots \ \mid \ \forall x : A.T \ \mid \ \exists x : A.T$$

- Extension with quantifiers

Value-dependent session types [Toninho et al., 2011]

$$T, S \ ::= \ \cdots \ \mid \forall x : A.T \ \mid \ \exists x : A.T$$

- Extension with quantifiers
- Exchanged messages are bound
    - $\forall$ and $\exists$ represent input/output operations

Value-dependent session types [Toninho et al., 2011]

$$T, S \ ::= \ \cdots \ \mid \ \forall x : A.T \ \mid \ \exists x : A.T$$

- Extension with quantifiers
- Exchanged messages are bound
  - $\forall$ and $\exists$ represent input/output operations
- Properties can be expressed

$$\forall x : \mathbb{N}.?(x \not\equiv 0).!\mathbb{N}.\mathsf{end}$$

Value-dependent session types [Toninho et al., 2011]

$$T, S \ ::= \ \cdots \ \mid \forall x : A.T \mid \exists x : A.T$$

- Extension with quantifiers
- Exchanged messages are bound
    - $\forall$ and $\exists$ represent input/output operations
- Properties can be expressed

$$\forall x : \mathbb{N}.?(x \not\equiv 0).!\mathbb{N}.\mathsf{end}$$

- Extension of the encoding

$$\llbracket \forall x : A.T \rrbracket = {}^{1,0}[\Sigma(x : A)\llbracket T \rrbracket]$$
$$\llbracket \exists x : A.T \rrbracket = {}^{0,1}[\Sigma(x : A)\llbracket \overline{T} \rrbracket]$$

Label-dependent session types [Thiemann and Vasconcelos, 2020]

$$T, S \;::=\; \mathsf{end} \;\mid\; ?x : m.T \;\mid\; !x : m.S \;\mid\; \mathsf{case}\ x\ \mathsf{of}\ \{T, S\}$$

- No branches and choices constructs

Label-dependent session types [Thiemann and Vasconcelos, 2020]

$$T, S \ ::= \ \mathsf{end} \ \mid \ ?x : m.T \ \mid \ !x : m.S \ \mid \ \mathsf{case}\ x\ \mathsf{of}\ \{T, S\}$$

- No branches and choices constructs
- Label dependency
  - $\mathsf{case}\ x\ \mathsf{of}\ \{T, S\}$ for pattern matching over a label

Label-dependent session types [Thiemann and Vasconcelos, 2020]

$$T, S \ ::= \ \mathsf{end} \ \mid \ ?x : m.T \ \mid \ !x : m.S \ \mid \ \mathsf{case}\ x\ \mathsf{of}\ \{T, S\}$$

- No branches and choices constructs
- Label dependency
  - $\mathsf{case}\ x\ \mathsf{of}\ \{T, S\}$ for pattern matching over a label
- We consider boolean values as set of labels

Encoding of label-dependent session types

$$
\begin{aligned}
\llbracket \mathsf{end} \rrbracket &= {}^{0,0}[\top] \\
\llbracket ?x : m.T \rrbracket &= {}^{1,0}[\Sigma(x : \llbracket m \rrbracket)\llbracket T \rrbracket] \\
\llbracket !x : m.T \rrbracket &= {}^{0,1}[\Sigma(x : \llbracket m \rrbracket)\llbracket \overline{T} \rrbracket] \\
\llbracket \mathsf{case}\ x\ \mathsf{of}\ \{T, S\} \rrbracket &= \mathsf{if}\ x\ \mathsf{then}\ \llbracket T \rrbracket\ \mathsf{else}\ \llbracket S \rrbracket
\end{aligned}
$$

- Again, we take advantage of the functional layer
- An environment is needed to track labels

Properties of the encodings

Encodings $\llbracket \cdot \rrbracket$ are **not injective**

$$\begin{array}{rclcl} \llbracket ?\mathsf{Bool}.T \rrbracket & = & \llbracket T \,\&\, T \rrbracket & = & \llbracket \forall x : \mathsf{Bool}.T \rrbracket \\ \llbracket !\mathsf{Bool}.T \rrbracket & = & \llbracket T \oplus T \rrbracket & = & \llbracket \exists x : \mathsf{Bool}.T \rrbracket \end{array}$$

Consequences

- Semantics of different constructs overlap
- Encoding is not invertible
- Decoding $\llparenthesis \cdot \rrparenthesis$ produces a "canonical" session type that is proved to be **bisimilar** to the original one (details in the repository)

Summary

"Simple" but expressive type system

- Linear channels + linear dependent pairs
- Agda: mechanised methatheory + functional layer
- Capable of encoding dependent session types

Summary

"Simple" but expressive type system

- Linear channels + linear dependent pairs
- Agda: mechanised methatheory + functional layer
- Capable of encoding dependent session types

More in the repository

- **Full structural congruence** and **reduction**
  - Implicitly: congruence and reduction **preserve typing**
- Additional properties
  - Linear channels not discarded and used at most once
- Examples of **variable-length** protocols
- **Recursive protocols** (with **sized types** [Abel, 2010])

Summary

"Simple" but expressive type system

- Linear channels + linear dependent pairs
- Agda: mechanised methatheory + functional layer
- Capable of encoding dependent session types

More in the repository

- **Full structural congruence** and **reduction**
  - Implicitly: congruence and reduction **preserve typing**
- Additional properties
  - Linear channels not discarded and used at most once
- Examples of **variable-length** protocols
- **Recursive protocols** (with **sized types** [Abel, 2010])

Ongoing work

- **Inference** of multiplicities, split and null relations (Agda prototype)
- Library implementation of dependent session types (Idris prototype)

Thank you

https://gitlab.di.unito.it/luca.padovani/DependentLinearPi

Variable-length protocol

$$Q_4 \stackrel{\text{def}}{=} *a(n, v).F(n, v, 1)$$

$$\begin{aligned}
F(0 \quad\;\; , \; v, \; z) &= \; v\langle z\rangle \\
F(n+1, \; v, \; z) &= \; v(x, y).F(n, y, x * z)
\end{aligned}$$

Variable-length protocol

$$Q_4 \stackrel{\text{def}}{=} *a(n, v).F(n, v, 1)$$

$$\begin{aligned} F(0 \quad\;\, , \; v, \; z) &= \; v\langle z \rangle \\ F(n+1, \; v, \; z) &= \; v(x, y).F(n, y, x * z) \end{aligned}$$

$\mathsf{f} : \mathbb{N} \to \mathsf{Type}$
$\mathsf{f}\ \mathsf{zero} \quad\;\; = \mathsf{Chan}\ \#0\ \#1\ (\mathsf{Pure}\ \mathbb{N})$
$\mathsf{f}\ (\mathsf{suc}\ n) = \mathsf{Chan}\ \#1\ \#0\ (\mathsf{Pair}\ (\mathsf{Pure}\ \mathbb{N})\ \lambda\ \_ \to \mathsf{f}\ n)$

$\mathsf{t}_4 : \mathsf{Type}$
$\mathsf{t}_4 = \mathsf{Chan}\ \#\omega\ \#0\ (\mathsf{Pair}\ (\mathsf{Pure}\ \mathbb{N})\ \mathsf{f})$

Andreas Abel. Miniagda: Integrating sized and dependent types. In *PAR@ITP 2010*, volume 5 of *EPiC Series*, pages 18–32. EasyChair, 2010. URL `https://arxiv.org/pdf/1012.4896.pdf`.

Edwin Brady. Type-driven development of concurrent communicating systems. *Computer Science (AGH)*, 18(3), 2017. doi: 10.7494/csci.2017.18.3.1413.

Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017. doi: 10.1016/j.ic.2017.06.002.

Dennis Griffith and Elsa L. Gunter. Liquidpi: Inferrable dependent session types. In *NFM 2013*, volume 7871 of *LNCS*, pages 185–197. Springer, 2013. doi: 10.1007/978-3-642-38088-4_13.

Peter Thiemann and Vasco T. Vasconcelos. Label-dependent session types. *Proc. ACM Program. Lang.*, 4(POPL):67:1–67:29, 2020. doi: 10.1145/3371135.

Bernardo Toninho and Nobuko Yoshida. Depending on session-typed processes. In *FOSSACS 2018*, volume 10803 of *LNCS*, pages 128–145. Springer, 2018. doi: 10.1007/978-3-319-89366-2_7.

Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In *PPDP 2011*, pages 161–172. ACM, 2011. doi: 10.1145/2003476.2003499.