Type reconstruction for the linear π-calculus with composite and equi-recursive types

Luca Padovani – Dipartimento di Informatica – Torino

“The topic may look a bit old but is important”

- Kobayashi, Pierce, Turner, *Linearity and the π-calculus*, 1999
- Igarashi, Kobayashi, *Type reconstruction for linear π-calculus with I/O subtyping*, 2000

Session = private conversation on **linearized** channels

- Honda, *Types for dyadic interaction*, 1993
- . . . long list of works (esp. in last decade)

safety + fidelity + progress

Sessions and the linear π-calculus

Dyadic sessions (with 2 participants)

- Kobayashi, *Type systems for concurrent programs*, 2002
- Dardha, Giachino, Sangiorgi, *Session types revisited*, 2012

Multiparty sessions (with ≥ 2 participants)

- Padovani, *Deadlock and lock freedom in the linear π-calculus*, 2014 (long version on my home page)

▶ Moral

The linear π-calculus is a foundational model for sessions

Outline

1. Motivation
2. The linear π-calculus with composite and recursive types
3. Type reconstruction
4. Concluding remarks

The linear π-calculus

Sessions vs linear channels

$a?(x).a!\langle x+1\rangle \ldots$ $\qquad$ $a?(x,y).(\nu b)y!\langle x+1, b\rangle$

$a : ?\texttt{int}.!\texttt{bool} \ldots$ $\qquad$ $a : [\texttt{int} \times [\texttt{bool} \times \ldots]^{0,1}]^{1,0}$

Extensions

- pairs $\Rightarrow$ product type
- alternative protocols $\Rightarrow$ disjoint sums
- iterative protocols, XML documents, ... $\Rightarrow$ recursive types

Types

Type	t	$::=$	$\texttt{unit}, \texttt{int}$	basic types
		$\mid$	$[t]^{\kappa,\kappa}$	channel type
		$\mid$	$t \times t$	product
		$\mid$	$t \oplus t$	disjoint sum
		$\mid$	α	type variable
		$\mid$	$\mu\alpha.t$	recursive type
Use	κ	$::=$	0	never
		$\mid$	1	once
		$\mid$	ω	unlimited

Typing

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \qquad \frac{\Gamma, x : t \vdash P \qquad 0 < \kappa}{\Gamma + u : [t]^{\kappa,0} \vdash u?(x).P}$$

Typing

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \qquad\qquad \frac{\Gamma, x : t \vdash P \qquad 0 < \kappa}{\Gamma + u : [t]^{\kappa,0} \vdash u?(x).P}$$

$$u : t_1 \;+\; u : t_2 \;=\; u : t_1 + t_2$$

Typing

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \qquad\qquad \frac{\Gamma, x : t \vdash P \qquad 0 < \kappa}{\Gamma + u : [t]^{\kappa,0} \vdash u?(x).P}$$

$$u : t_1 \;+\; u : t_2 \;=\; u : t_1 + t_2$$

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

Typing

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \qquad\qquad \frac{\Gamma, x : t \vdash P \qquad 0 < \kappa}{\Gamma + u : [t]^{\kappa,0} \vdash u?(x).P}$$

$$u : t_1 \quad + \quad u : t_2 \quad = \quad u : t_1 + t_2$$

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

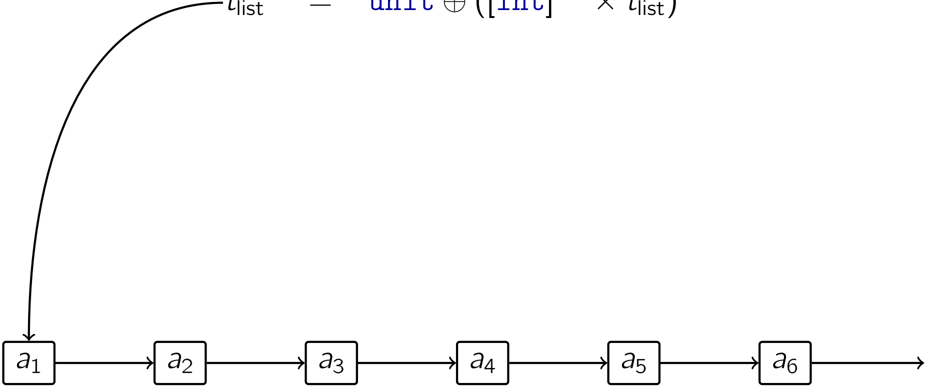
▶ In this work

Extend $+$ component-wise to **composite** types

Extend $+$ coinductively to **recursive** types

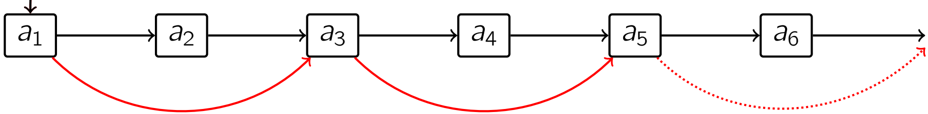
Example: list of linear channels

$$t_{\text{list}} \quad = \quad \texttt{unit} \oplus ([\texttt{int}]^{0,1} \times t_{\text{list}})$$



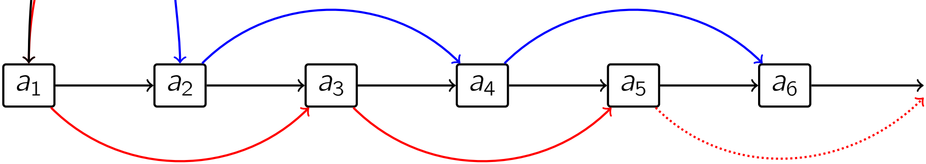
Example: list of linear channels

$$
\begin{aligned}
t_{\text{list}} &= \texttt{unit} \oplus ([\texttt{int}]^{0,1} \times t_{\text{list}}) \\
\| \\
t_{\text{odd}} &= \texttt{unit} \oplus ([\texttt{int}]^{0,1} \times t_{\text{even}}) \\
+ \\
t_{\text{even}}
\end{aligned}
$$



Example: list of linear channels

$$
\begin{aligned}
t_{\text{list}} &= \texttt{unit} \oplus ([\texttt{int}]^{0,1} \times t_{\text{list}}) \\
\| & \\
t_{\text{odd}} &= \texttt{unit} \oplus ([\texttt{int}]^{0,1} \times t_{\text{even}}) \\
+ & \\
t_{\text{even}} &= \texttt{unit} \oplus ([\texttt{int}]^{0,0} \times t_{\text{odd}})
\end{aligned}
$$



Example: sharing a list of linear channels

$$
\begin{array}{llll}
*odd?(x).\texttt{case}\ x\ \texttt{of} & () & \Rightarrow & \mathbf{0} \\
 & y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle
\end{array}
$$

$$
\begin{array}{llll}
*even?(x).\texttt{case}\ x\ \texttt{of} & () & \Rightarrow & \mathbf{0} \\
 & y : z & \Rightarrow & odd!\langle z\rangle
\end{array}
$$

$$
\frac{L : t_{\text{odd}} \vdash odd!\langle L\rangle \qquad L : t_{\text{even}} \vdash even!\langle L\rangle}{L : t_{\text{list}} \vdash odd!\langle L\rangle \mid even!\langle L\rangle}
$$

Example: sharing a list of linear channels

$$*odd?(x).\texttt{case}\ x\ \texttt{of}\ \begin{array}{lcl} () & \Rightarrow & \mathbf{0} \\ y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle \end{array}$$

$$*even?(x).\texttt{case}\ x\ \texttt{of}\ \begin{array}{lcl} () & \Rightarrow & \mathbf{0} \\ y : z & \Rightarrow & odd!\langle z\rangle \end{array}$$

$$\frac{L : t_{\mathrm{odd}} \vdash odd!\langle L\rangle \qquad L : t_{\mathrm{even}} \vdash even!\langle L\rangle}{L : t_{\mathrm{list}} \vdash odd!\langle L\rangle \mid even!\langle L\rangle}$$

Type reconstruction

▶ Problem statement

- given P, find Γ such that $\Gamma \vdash P$, if there is one
- maximize the number of **linear** channels in Γ

Too much **guessing** in the typing rules

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \qquad\qquad \frac{\Gamma, x : t \vdash P \qquad 0 < \kappa}{\Gamma + u : [t]^{\kappa,0} \vdash u?(x).P}$$

Type reconstruction

▶ Problem statement

- given P, find Γ such that $\Gamma \vdash P$, if there is one
- maximize the number of **linear** channels in Γ

Too much **guessing** in the typing rules

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \qquad \frac{\Gamma, x : t \vdash P \qquad 0 < \kappa}{\Gamma + u : [t]^{\kappa,0} \vdash u?(x).P}$$

Type reconstruction

▶ Problem statement

- given P, find Γ such that $\Gamma \vdash P$, if there is one
- maximize the number of **linear** channels in Γ

Too much **guessing** in the typing rules

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \qquad \frac{\Gamma, x : t \vdash P \qquad 0 < \kappa}{\Gamma + u : [t]^{\kappa,0} \vdash u?(x).P}$$

Type reconstruction

▶ Problem statement

- given P, find Γ such that $\Gamma \vdash P$, if there is one
- maximize the number of **linear** channels in Γ

Too much **guessing** in the typing rules

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \qquad \frac{\Gamma, x : t \vdash P \qquad 0 < \kappa}{\Gamma + u : [t]^{\kappa,0} \vdash u?(x).P}$$

Computing constraints

1. type variables α denote unknown types
2. use variables ρ denote unknown uses
3. $\Gamma \vdash P \quad \Rightarrow \quad P \triangleright \Gamma; \mathcal{C}$

combine environments

$$\frac{\Gamma_i \vdash P_i \ {}^{(i=1,2)}}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2} \Rightarrow \frac{P_i \triangleright \Gamma_i; \mathcal{C}_i \ {}^{(i=1,2)} \qquad \Gamma_1 \sqcup \Gamma_2 \leadsto \Gamma; \mathcal{C}_3}{P_1 \mid P_2 \triangleright \Gamma; \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3}$$

Combining environments

$$\frac{\mathsf{dom}(\Gamma_1) \cap \mathsf{dom}(\Gamma_2) = \emptyset}{\Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma_1, \Gamma_2; \emptyset}$$

Combining environments

$$\frac{\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset}{\Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma_1, \Gamma_2; \emptyset}$$

$$\frac{\Gamma_1 \sqcup \Gamma_2 \rightsquigarrow \Gamma; \mathcal{C}}{(\Gamma_1, u : t_1) \sqcup (\Gamma_2, u : t_2) \rightsquigarrow \Gamma, u : \alpha; \mathcal{C} \cup \{\alpha = t_1 + t_2\}}$$

Combining environments

$$\frac{\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset}{\Gamma_1 \sqcup \Gamma_2 \leadsto \Gamma_1, \Gamma_2; \emptyset}$$

α is the combination of t_1 and t_2

$$\frac{\Gamma_1 \sqcup \Gamma_2 \leadsto \Gamma; \mathcal{C}}{(\Gamma_1, u : t_1) \sqcup (\Gamma_2, u : t_2) \leadsto \Gamma, u : \alpha; \mathcal{C} \cup \{\alpha = t_1 + t_2\}}$$

unknown type, fresh type variable

Example: lists of linear channels

$$*odd?(x).\texttt{case}\ x\ \texttt{of}\quad \begin{array}{lcl} () & \Rightarrow & \mathbf{0} \\ y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle \end{array}$$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

Example: lists of linear channels

$x : \alpha$

$$*odd?(x).\texttt{case}\ x\ \texttt{of}\ \begin{array}{lcl} () & \Rightarrow & \mathbf{0} \\ y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle \end{array}$$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

Example: lists of linear channels

$x : \alpha$

$$*odd?(x).\texttt{case}\ x\ \texttt{of}\ \begin{array}{lcl} () & \Rightarrow & \mathbf{0} \\ y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle \end{array}$$

$\alpha = \alpha_1 \oplus \alpha_2$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

Example: lists of linear channels

$x : \alpha$

$\alpha_1 = \texttt{unit}$

$$*odd?(x).\texttt{case}\ x\ \texttt{of}\ \begin{array}{lcl} () & \Rightarrow & \mathbf{0} \\ y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle \end{array}$$

$\alpha = \alpha_1 \oplus \alpha_2$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

Example: lists of linear channels

$x : \alpha$

$\alpha_1 = \texttt{unit}$

$$*odd?(x).\texttt{case}\ x\ \texttt{of}\ \ ()\ \Rightarrow\ \mathbf{0}$$
$$y : z\ \Rightarrow\ y!\langle 3\rangle \mid even!\langle z\rangle$$

$\alpha = \alpha_1 \oplus \alpha_2$

$\alpha_2 = \alpha_3 \times \alpha_4$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

Example: lists of linear channels

$x : \alpha$

$\alpha_1 = \texttt{unit}$

$$
\begin{array}{llcl}
*odd?(x).\texttt{case}\ x\ \texttt{of} & () & \Rightarrow & \mathbf{0} \\
 & y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle
\end{array}
$$

$\alpha = \alpha_1 \oplus \alpha_2$

$\alpha_2 = \alpha_3 \times \alpha_4$

$\alpha_3 = [\texttt{int}]^{0,1}$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

Example: lists of linear channels

$x : \alpha$

$\alpha_1 = \texttt{unit}$

$even : [\alpha_4]^{0,\omega}$

$$
\begin{aligned}
*odd?(x).\texttt{case}\ x\ \texttt{of}\quad & () && \Rightarrow\ \mathbf{0} \\
& y : z && \Rightarrow\ y!\langle 3\rangle \mid even!\langle z\rangle
\end{aligned}
$$

$\alpha = \alpha_1 \oplus \alpha_2$

$\alpha_2 = \alpha_3 \times \alpha_4$

$\alpha_3 = [\texttt{int}]^{0,1}$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

Example: lists of linear channels

$$
*odd?(x).\texttt{case}\ x\ \texttt{of}\ \begin{array}{lcl} () & \Rightarrow & \mathbf{0} \\ y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle \end{array}
$$

$L : \alpha$

$$
even!\langle L\rangle \mid odd!\langle L\rangle
$$

Example: lists of linear channels

$$*odd?(x).\texttt{case}\ x\ \texttt{of}\ \begin{array}{lcl} () & \Rightarrow & \mathbf{0} \\ y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle \end{array}$$

$L : \alpha$ $L : \beta$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

Example: lists of linear channels

$$
\ast odd?(x).\texttt{case}\ x\ \texttt{of}\quad
\begin{array}{lcl}
() & \Rightarrow & \mathbf{0} \\
y : z & \Rightarrow & y!\langle 3\rangle \mid even!\langle z\rangle
\end{array}
$$

$L : \alpha$ $\quad$ $L : \beta$

$$even!\langle L\rangle \mid odd!\langle L\rangle$$

$L : \gamma,\ \gamma = \alpha + \beta$

Results

Theorem (Correctness)

If $P \triangleright \Gamma; \mathcal{C}$ *and* σ *is a solution for* $\mathcal{C}$*, then* $\sigma\Gamma \vdash P$

assignment for the type/use variables in $\mathcal{C}$

Theorem (Completeness)

If $\Gamma' \vdash P$*, then* $P \triangleright \Gamma; \mathcal{C}$ *where* $\Gamma' = \sigma\Gamma$ *and* σ *is a solution of* $\mathcal{C}$

Results

Theorem (Correctness)

If $P \triangleright \Gamma; \mathcal{C}$ *and* σ *is a solution for* $\mathcal{C}$, *then* $\sigma\Gamma \vdash P$

Theorem (Completeness)

If $\Gamma' \vdash P$, *then* $P \triangleright \Gamma; \mathcal{C}$ *where* $\Gamma' = \sigma\Gamma$ *and* σ *is a solution of* $\mathcal{C}$

The (implicit) meaning of **type** constraints

$$[t]^{\kappa,\iota} = [t]^{\kappa_1,\iota_1} + [t]^{\kappa_2,\iota_2} \qquad \models \qquad \kappa = \kappa_1 + \kappa_2 \qquad \iota = \iota_1 + \iota_2$$

▶ Problem

- we must find all the (implicit) **use** constraints
- apply entailment until no new constraints are discovered...
- ...but how do we know that this process **terminates**?

$s = t_1 + t_2$ is indeed an awkward constraint

$$\begin{array}{ccccc} s & = & t_1 & + & t_2 \\ \| & & & & \\ s_1 & & & & \\ \times & & & & \\ s_2 & & & & \end{array}$$

- to discover all the implicit constraints it may be necessary to introduce new type variables
- not clear when to stop
(ambiguous decompositions + recursive types)

▶ Idea

Express composition as multiple binary relations

$s = t_1 + t_2$ is indeed an awkward constraint

$$
\begin{array}{ccccc}
s & = & t_1 & + & t_2 \\
\| & & \| & & \| \\
s_1 & = & \beta_1 & + & \beta_2 \\
\times & & \times & & \times \\
s_2 & = & \gamma_1 & + & \gamma_2
\end{array}
$$

- to discover all the implicit constraints it may be necessary to introduce new type variables
- not clear when to stop
  (ambiguous decompositions + recursive types)

▶ Idea

Express composition as multiple binary relations

$s = t_1 + t_2$ is indeed an awkward constraint

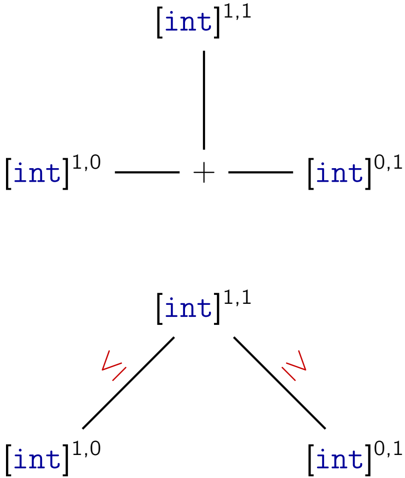
$$\begin{array}{ccccc} s & = & t_1 & + & t_2 \\ \| & & \| & & \| \\ s_1 & = & \beta_1 & + & \beta_2 \\ \times & & \times & & \times \\ s_2 & = & \gamma_1 & + & \gamma_2 \end{array}$$

- to discover all the implicit constraints it may be necessary to introduce new type variables
- not clear when to stop
  (ambiguous decompositions + recursive types)

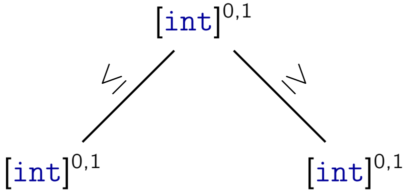
▶ Idea

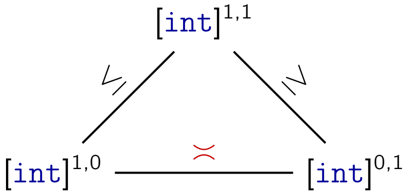
Express composition as multiple binary relations

Composition ∼ least upper bound

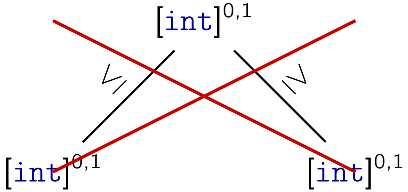


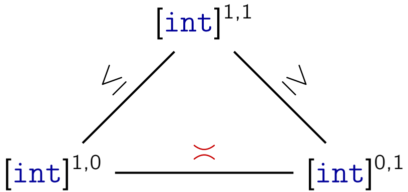
Composition = lub + compatibility





Composition = lub + compatibility





Solver

Input: a set of constraints $\mathcal{C}$
Output: either **fail** or a solution of $\mathcal{C}$

1. Saturate $\mathcal{C}$
2. Compute an *optimal* solution σ_{use} for the use constraints in $\mathcal{C}$, or **fail** if there is none
3. **fail** if $t \mathrel{\mathcal{R}} s \in \mathcal{C}$ and t, s have different topmost constructors
4. Let $\sigma_{type} = \{\alpha \mapsto \sup_{\mathcal{C},\sigma_{use}}(\{\alpha\}) \mid \alpha \in \mathrm{dom}(\mathcal{C})\}$
5. Return $\sigma_{use} \cup \sigma_{type}$

Solver

Input: a set of constraints $\mathcal{C}$
Output: either **fail** or a solution of $\mathcal{C}$

maximize number of linear channels

1. Saturate $\mathcal{C}$
2. Compute an *optimal* solution σ_{use} for the use constraints in $\mathcal{C}$, or **fail** if there is none

finitely many use assignments

3. **fail** if $t \mathrel{\mathcal{R}} s \in \mathcal{C}$ and t, s have different topmost constructors
4. Let $\sigma_{type} = \{\alpha \mapsto \sup_{\mathcal{C},\sigma_{use}}(\{\alpha\}) \mid \alpha \in \mathrm{dom}(\mathcal{C})\}$
5. Return $\sigma_{use} \cup \sigma_{type}$

Solver

Input: a set of constraints $\mathcal{C}$
Output: either **fail** or a solution of $\mathcal{C}$

1. Saturate $\mathcal{C}$
2. Compute an *optimal* solution σ_{use} for the use constraints in $\mathcal{C}$, or **fail** if there is none
3. **fail** if $t \mathrel{\mathcal{R}} s \in \mathcal{C}$ and t, s have different topmost constructors
4. Let $\sigma_{type} = \{\alpha \mapsto \sup_{\mathcal{C},\sigma_{use}}(\{\alpha\}) \mid \alpha \in \mathrm{dom}(\mathcal{C})\}$
5. Return $\sigma_{use} \cup \sigma_{type}$

Solver

Input: a set of constraints $\mathcal{C}$
Output: either **fail** or a solution of $\mathcal{C}$

1. Saturate $\mathcal{C}$
2. Compute an *optimal* solution σ_{use} for the use constraints in $\mathcal{C}$, or **fail** if there is none
3. **fail** if $t\ \mathcal{R}\ s \in \mathcal{C}$ and t, s have different topmost constructors
4. Let $\sigma_{type} = \{\alpha \mapsto \sup_{\mathcal{C},\sigma_{use}}(\{\alpha\}) \mid \alpha \in \mathrm{dom}(\mathcal{C})\}$
5. Return σ_{use}

least upper bound of α

Solver

Input: a set of constraints $\mathcal{C}$
Output: either **fail** or a solution of $\mathcal{C}$

1. Saturate $\mathcal{C}$
2. Compute an *optimal* solution σ_{use} for the use constraints in $\mathcal{C}$, or **fail** if there is none
3. **fail** if $t \mathrel{\mathcal{R}} s \in \mathcal{C}$ and t, s have different topmost constructors
4. Let $\sigma_{type} = \{\alpha \mapsto \sup_{\mathcal{C},\sigma_{use}}(\{\alpha\}) \mid \alpha \in \mathrm{dom}(\mathcal{C})\}$
5. Return $\sigma_{use} \cup \sigma_{type}$

Properties of the solver algorithm

Theorem (Correctness)

Given a set $\mathcal{C}$ of constraints:

1. *if the algorithm returns σ, then σ is a solution of $\mathcal{C}$*
2. *if the algorithm* **fail***s, then $\mathcal{C}$ has no solution*

Theorem (Termination)

The algorithm always terminates

Corollary (Completeness)

If $\mathcal{C}$ admits a solution, the algorithm finds one

Implementation

examples — bash — 80×24

```
uria:examples luca$ cat evenodd.hypha

  *odd?(m).
   case m of
   [ _        => {}
   ; (x, y) => x!3 | even!y ]
|
  *even?(m).
   case m of
   [ _        => {}
   ; (_, y) => odd!y ]

| odd!l | even!l

uria:examples luca$ echo; ../src/hypha evenodd.hypha; echo

  even : [(Unit ⊕ μA.(([Int]{0,0} × (Unit ⊕ ([Int]{0,1} × (Unit ⊕ A))))))]{ω,ω}
  l : (Unit ⊕ μA.(([Int]{0,1} × (Unit ⊕ A))))
  odd : [(Unit ⊕ μA.(([Int]{0,1} × (Unit ⊕ ([Int]{0,0} × (Unit ⊕ A))))))]{ω,ω}

uria:examples luca$
```

Concluding remarks

Pros

- conservative extension of linear π-calculus (same rules)
- boosts parallelism and data sharing in presence of linear values
- not tied to π-calculus

Cons

- only regular decompositions

Issues

- complexity (cannot use unification)
- subtyping (important for sessions)

Code

- `http://www.di.unito.it/~padovani/`