Type reconstruction for the linear π-calculus

Luca Padovani

Dipartimento di Informatica – Università di Torino – Italy

15 december 2014

The linear π-calculus

Unlimited channel

$$^{\omega,\omega}[t]$$

n communications

Linear channel

$$^{1,1}[t]$$

1 communication

Unused channel

$$^{0,0}[t]$$

no communications

Kobayashi, Pierce, Turner, **Linearity and the pi-calculus**, TOPLAS 1999

The linear π-calculus: motivations

Why the focus on linear channels?

- $\geq$50% of channels are linear
- linear channels and simple and efficient to (de)allocate
- confluence $\Rightarrow$ deterministic parallelism
- linearity-aware behavioural equivalences

Type reconstruction

▶ Problem statement

Given an untyped program P, find Γ, if there is one, such that

1. $\Gamma \vdash P$
2. Γ is the "most precise" environment for P

- "most precise" $\Rightarrow$ maximise number of linear channels

Type reconstruction: motivations

- facility for the programmer

- inference tool of program's properties
  - linearity analysis
  - protocol analysis
  - deadlock analysis
  - lock analysis

Type reconstruction: motivations

- facility for the programmer

- inference tool of program's properties
  - linearity analysis
  - protocol analysis
  - deadlock analysis
  - lock analysis

Outline

1. Introduction
2. Linearity analysis
3. Protocol analysis
4. Deadlock analysis
5. Lock analysis
6. Final remarks

Demo

```
*fibo?(n, r).                -- fibo unlimited, r linear
 if n ≤ 1 then
   r!n
 else {
   new a in                  -- a linear
   new b in {                -- b linear
     fibo!(n - 1, a) |
     fibo!(n - 2, b) |
     a?x.b?y.r!(x + y)
   }
 }
```

$$\texttt{fibo} : {}^{\omega,\omega}[\texttt{int} \times {}^{0,1}[\texttt{int}]]$$

Type reconstruction: how it works 1/3

```
new a in { a!3 | a?x }
```

- $\alpha_0 = \alpha_1 + \alpha_2$ type combination $\neq$ unification
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{1,0}[\texttt{int}]$
- $\rho = 0 + 1$
- $\rho = 1 + 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

Type reconstruction: how it works 1/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

`new a in { a!3 | a?x }`

$\alpha_0 = {}^{\rho,\rho}[\texttt{int}]$ $\alpha_2 = {}^{1,0}[\texttt{int}]$

- $\alpha_0 = \alpha_1 + \alpha_2$ type combination $\neq$ unification
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{1,0}[\texttt{int}]$
- $\rho = 0 + 1$
- $\rho = 1 + 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

Type reconstruction: how it works 1/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

new a **in** { a!3 | a?x }

$\alpha_0 = {}^{\rho,\rho}[\texttt{int}]$ $\alpha_2 = {}^{1,0}[\texttt{int}]$

- $\alpha_0 = \alpha_1 + \alpha_2$
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{1,0}[\texttt{int}]$
- $\rho = 0 + 1$
- $\rho = 1 + 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

type combination $\neq$ unification

Type reconstruction: how it works 1/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

`new a in { a!3 | a?x }`

$\alpha_0 = {}^{\rho,\rho}[\texttt{int}]$ $\alpha_2 = {}^{1,0}[\texttt{int}]$

- $\alpha_0 = \alpha_1 + \alpha_2$ — type combination $\neq$ unification
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{1,0}[\texttt{int}]$
- $\rho = 0 + 1$
- $\rho = 1 + 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

Type reconstruction: how it works 1/3

$\alpha_1 = {}^{0,1}[\mathtt{int}]$

$\mathtt{new}\ a\ \mathtt{in}\ \{\ a!3 \mid a?x\ \}$

$\alpha_0 = {}^{\rho,\rho}[\mathtt{int}]$ $\alpha_2 = {}^{1,0}[\mathtt{int}]$

- $\alpha_0 = \alpha_1 + \alpha_2$ type combination $\neq$ unification
- ${}^{\rho,\rho}[\mathtt{int}] = {}^{0,1}[\mathtt{int}] + {}^{1,0}[\mathtt{int}]$
- $\rho = 0 + 1$
- $\rho = 1 + 0$
- $\alpha_0 = {}^{1,1}[\mathtt{int}]$

Type reconstruction: how it works 2/3

```
a!3 | a!4
```

- ${}^{\rho_1,\rho_2}[\mathtt{int}] = {}^{0,1}[\mathtt{int}] + {}^{0,1}[\mathtt{int}]$
- $\rho_1 = 0 + 0 = 0$
- $\rho_2 = 1 + 1 = \omega$ ω approximates 2 uses

Type reconstruction: how it works 2/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

`a!3 | a!4`

$\alpha_2 = {}^{0,1}[\texttt{int}]$

- ${}^{\rho_1,\rho_2}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{0,1}[\texttt{int}]$
- $\rho_1 = 0 + 0 = 0$
- $\rho_2 = 1 + 1 = \omega$

ω approximates 2 uses

Type reconstruction: how it works 2/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

a!3 | a!4

$\alpha_2 = {}^{0,1}[\texttt{int}]$

- ${}^{\rho_1,\rho_2}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{0,1}[\texttt{int}]$
- $\rho_1 = 0 + 0 = 0$
- $\rho_2 = 1 + 1 = \omega$

ω approximates 2 uses

Type reconstruction: how it works 2/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

a!3 | a!4

$\alpha_2 = {}^{0,1}[\texttt{int}]$

- ${}^{\rho_1,\rho_2}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{0,1}[\texttt{int}]$
- $\rho_1 = 0 + 0 = 0$
- $\rho_2 = 1 + 1 = \omega$

ω approximates 2 uses

Type reconstruction: how it works 3/3

`new a in { a!3 | b!a }`

- $\alpha_0 = \alpha_1 + \alpha_2$
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{\rho_1,\rho_2}[\texttt{int}]$
- $\rho = 0 + \rho_1$
- $\rho = 1 + \rho_2$
- $\rho_1 = 1$ ghost use!
- $\rho_2 = 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

Type reconstruction: how it works 3/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

$\texttt{new}\ \mathsf{a}\ \texttt{in}\ \{\ \mathsf{a}!3\ \mid\ b!\mathsf{a}\ \}$

$\alpha_0 = {}^{\rho,\rho}[\texttt{int}]$ $\alpha_2 = {}^{\rho_1,\rho_2}[\texttt{int}]$

- $\alpha_0 = \alpha_1 + \alpha_2$
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{\rho_1,\rho_2}[\texttt{int}]$
- $\rho = 0 + \rho_1$
- $\rho = 1 + \rho_2$
- $\rho_1 = 1$ ghost use!
- $\rho_2 = 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

Type reconstruction: how it works 3/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

new a **in** { a!3 | *b*!a }

$\alpha_0 = {}^{\rho,\rho}[\texttt{int}]$ $\alpha_2 = {}^{\rho_1,\rho_2}[\texttt{int}]$

- $\alpha_0 = \alpha_1 + \alpha_2$
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{\rho_1,\rho_2}[\texttt{int}]$
- $\rho = 0 + \rho_1$
- $\rho = 1 + \rho_2$
- $\rho_1 = 1$ ghost use!
- $\rho_2 = 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

Type reconstruction: how it works 3/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

$\texttt{new}\ \mathsf{a}\ \texttt{in}\ \{\ \mathsf{a}!3 \mid b!\mathsf{a}\ \}$

$\alpha_0 = {}^{\rho,\rho}[\texttt{int}]$ $\alpha_2 = {}^{\rho_1,\rho_2}[\texttt{int}]$

- $\alpha_0 = \alpha_1 + \alpha_2$
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{\rho_1,\rho_2}[\texttt{int}]$
- $\rho = 0 + \rho_1$
- $\rho = 1 + \rho_2$
- $\rho_1 = 1$ ghost use!
- $\rho_2 = 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

Type reconstruction: how it works 3/3

$\alpha_1 = {}^{0,1}[\texttt{int}]$

`new a in {  a!3  |  b!a  }`

$\alpha_0 = {}^{\rho,\rho}[\texttt{int}]$ $\alpha_2 = {}^{\rho_1,\rho_2}[\texttt{int}]$

- $\alpha_0 = \alpha_1 + \alpha_2$
- ${}^{\rho,\rho}[\texttt{int}] = {}^{0,1}[\texttt{int}] + {}^{\rho_1,\rho_2}[\texttt{int}]$
- $\rho = 0 + \rho_1$
- $\rho = 1 + \rho_2$
- $\rho_1 = 1$ ghost use!
- $\rho_2 = 0$
- $\alpha_0 = {}^{1,1}[\texttt{int}]$

Generalised type combination

$$^{\kappa_1,\kappa_2}[t] + {}^{\kappa_3,\kappa_4}[t] = {}^{\kappa_1+\kappa_3,\kappa_2+\kappa_4}[t]$$

Padovani, **Type Reconstruction for the Linear π-Calculus with Composite and Equi-Recursive Types**, FoSSaCS 2014

Example: pairs

```
*server?p.{ fst(p)!3 | snd(p)!false }
```

- $\alpha_0 = \alpha_1 + \alpha_2$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times \beta) + (\gamma \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times {}^{0,0}[\texttt{bool}]) + ({}^{0,0}[\texttt{int}] \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = {}^{0,1}[\texttt{int}] \times {}^{0,1}[\texttt{bool}]$

Example: pairs

$\alpha_1 = {}^{0,1}[\texttt{int}] \times \beta$, un$(\beta)$

```
*server?p.{ fst(p)!3 | snd(p)!false }
```

α_0

$\alpha_2 = \gamma \times {}^{0,1}[\texttt{int}]$, un$(\gamma)$

- $\alpha_0 = \alpha_1 + \alpha_2$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times \beta) + (\gamma \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times {}^{0,0}[\texttt{bool}]) + ({}^{0,0}[\texttt{int}] \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = {}^{0,1}[\texttt{int}] \times {}^{0,1}[\texttt{bool}]$

Example: pairs

$\alpha_1 = {}^{0,1}[\texttt{int}] \times \beta$, un($\beta$)

```
*server?p.{ fst(p)!3 | snd(p)!false }
```

α_0

$\alpha_2 = \gamma \times {}^{0,1}[\texttt{int}]$, un($\gamma$)

- $\alpha_0 = \alpha_1 + \alpha_2$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times \beta) + (\gamma \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times {}^{0,0}[\texttt{bool}]) + ({}^{0,0}[\texttt{int}] \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = {}^{0,1}[\texttt{int}] \times {}^{0,1}[\texttt{bool}]$

Example: pairs

$\alpha_1 = {}^{0,1}[\texttt{int}] \times \beta$, $\mathsf{un}(\beta)$

```
*server?p.{ fst(p)!3 | snd(p)!false }
```

α_0

$\alpha_2 = \gamma \times {}^{0,1}[\texttt{int}]$, $\mathsf{un}(\gamma)$

- $\alpha_0 = \alpha_1 + \alpha_2$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times \beta) + (\gamma \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times {}^{0,0}[\texttt{bool}]) + ({}^{0,0}[\texttt{int}] \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = {}^{0,1}[\texttt{int}] \times {}^{0,1}[\texttt{bool}]$

Example: pairs

$\alpha_1 = {}^{0,1}[\texttt{int}] \times \beta$, $\mathsf{un}(\beta)$

```
*server?p.{ fst(p)!3 | snd(p)!false }
```

α_0

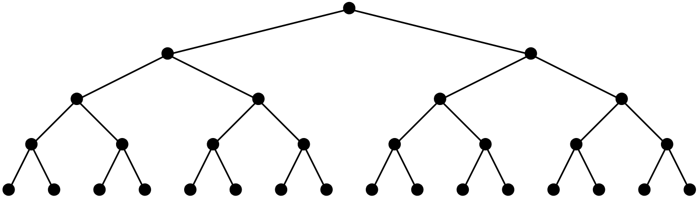
$\alpha_2 = \gamma \times {}^{0,1}[\texttt{int}]$, $\mathsf{un}(\gamma)$

- $\alpha_0 = \alpha_1 + \alpha_2$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times \beta) + (\gamma \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = ({}^{0,1}[\texttt{int}] \times {}^{0,0}[\texttt{bool}]) + ({}^{0,0}[\texttt{int}] \times {}^{0,1}[\texttt{bool}])$
- $\alpha_0 = {}^{0,1}[\texttt{int}] \times {}^{0,1}[\texttt{bool}]$

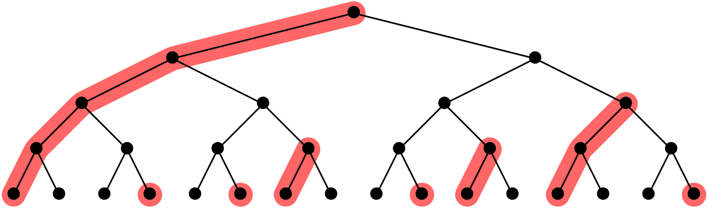
Demo: trees

```
  *case take? of
   { Leaf        ⇒ {}
   ; Node(c,l,r) ⇒ c!0 | take!l | skip!r }
|
  *case skip? of
   { Leaf        ⇒ {}
   ; Node(_,l,r) ⇒ skip!l | take!r }
|
  take!t | skip!t
```

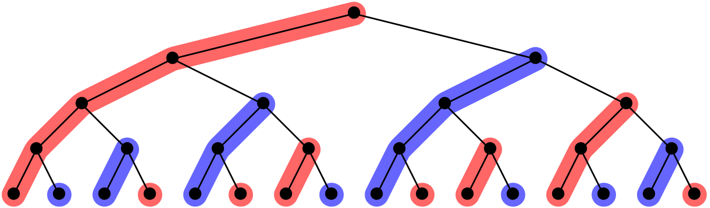
Channels used by `take` (red) and `skip` (blue)



Channels used by `take` (red) and `skip` (blue)



Channels used by `take` (red) and `skip` (blue)



Outline

Compiling binary sessions in the linear π-calculus

$$[\![s!e.P]\!] \quad = \quad \texttt{new}\ s'\ \texttt{in}\ s!(e, s').[\![P\{s'/s\}]\!]$$

- Kobayashi, **Type systems for concurrent programs**, 2002
- Demangeon, Honda, **Full Abstraction in a Subtyped pi-Calculus with Linear Types**, 2011
- Dardha, Giachino, Sangiorgi, **Session types revisited**, 2012

Compiling binary sessions in the linear π-calculus

$$
\begin{array}{rcl}
[\![\, s?x.P \,]\!] & = & s?(x, s').[\![\, P\{s'/s\} \,]\!] \\
[\![\, s!e.P \,]\!] & = & \texttt{new}\ s'\ \texttt{in}\ s!(e, s').[\![\, P\{s'/s\} \,]\!]
\end{array}
$$

- Kobayashi, **Type systems for concurrent programs**, 2002
- Demangeon, Honda, **Full Abstraction in a Subtyped pi-Calculus with Linear Types**, 2011
- Dardha, Giachino, Sangiorgi, **Session types revisited**, 2012

Compiling binary sessions in the linear π-calculus

$$
\begin{array}{rcl}
[\![\ s?x.P\ ]\!] & = & s?(x, s').[\![\ P\{s'/s\}\ ]\!] \\
[\![\ s!e.P\ ]\!] & = & \texttt{new}\ s'\ \texttt{in}\ s!(e, s').[\![\ P\{s'/s\}\ ]\!] \\
\\
[\![\ ?\texttt{int}.T\ ]\!] & = & {}^{1,0}[\texttt{int} \times [\![\ T\ ]\!]]
\end{array}
$$

- Kobayashi, **Type systems for concurrent programs**, 2002
- Demangeon, Honda, **Full Abstraction in a Subtyped pi-Calculus with Linear Types**, 2011
- Dardha, Giachino, Sangiorgi, **Session types revisited**, 2012

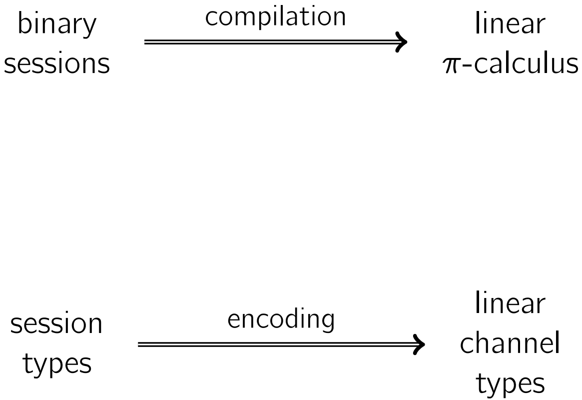
Compiling binary sessions in the linear π-calculus

$$
\begin{array}{rcl}
[\![s?x.P]\!] & = & s?(x, s').[\![P\{s'/s\}]\!] \\
[\![s!e.P]\!] & = & \texttt{new}\ s'\ \texttt{in}\ s!(e, s').[\![P\{s'/s\}]\!]
\end{array}
$$

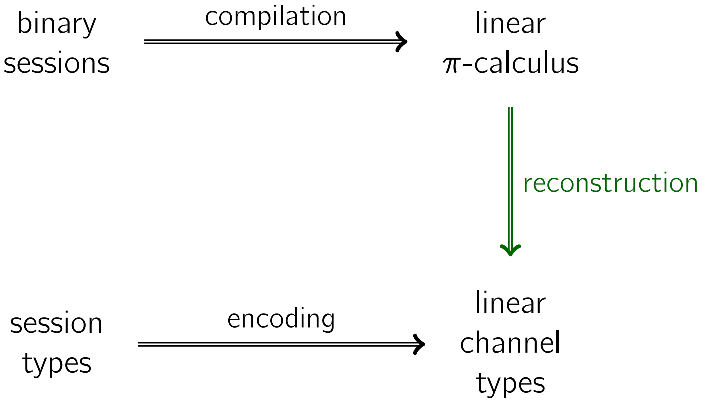
$$
\begin{array}{rcl}
[\![?\texttt{int}.T]\!] & = & {}^{1,0}[\texttt{int} \times [\![T]\!]] \\
[\![!\texttt{int}.T]\!] & = & {}^{0,1}[\texttt{int} \times [\![\overline{T}]\!]]
\end{array}
$$

- Kobayashi, **Type systems for concurrent programs**, 2002
- Demangeon, Honda, **Full Abstraction in a Subtyped pi-Calculus with Linear Types**, 2011
- Dardha, Giachino, Sangiorgi, **Session types revisited**, 2012

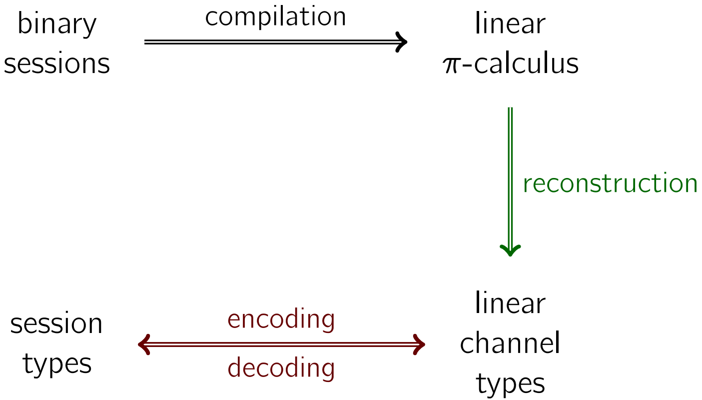
Session type reconstruction



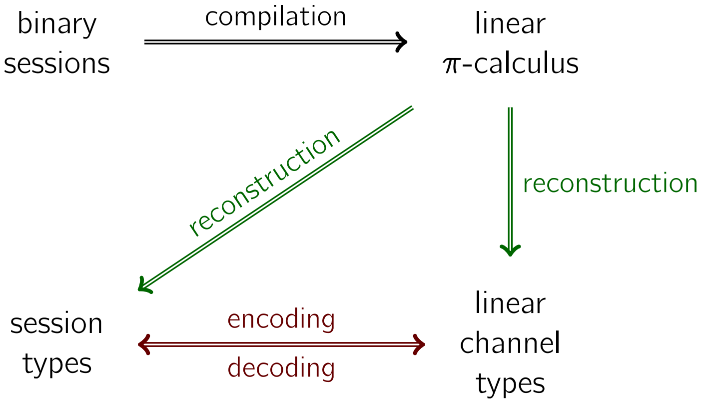
Session type reconstruction



Session type reconstruction



Session type reconstruction



Demo: math server

```
*server?s.
 case s? of
 { Quit    ⇒ {}
 ; Plus c1 ⇒ c1?(x,c2).
             c2?(y,c3).
             new c4 in { c3!(x + y, c4) | server!c4 }
 ; Eq c1   ⇒ c1?(x:Int,c2).
             c2?(y,c3).
             new c4 in { c3!(x = y, c4) | server!c4 }
 ; Neg c1  ⇒ c1?(x,c2).
             new c3 in { c2!(0 - x, c3) | server!c3 } }
```

Outline

Different processes, same typing

$a?x.b!\texttt{false} \mid a!3.b?y$

Different processes, same typing

$$\mathtt{a} : {}^{1,1}[\mathtt{int}], \mathtt{b} : {}^{1,1}[\mathtt{bool}] \vdash a?x.b!\mathtt{false} \mid a!3.b?y$$

Different processes, same typing

$$\mathtt{a} : {}^{1,1}[\mathtt{int}], \mathtt{b} : {}^{1,1}[\mathtt{bool}] \vdash a?x.b!\mathtt{false} \mid a!3.b?y$$

$$a?x.b!\mathtt{false} \mid b?y.a!3$$

Different processes, same typing

$$\mathtt{a} : {}^{1,1}[\mathtt{int}], \mathtt{b} : {}^{1,1}[\mathtt{bool}] \vdash a?x.b!\mathtt{false} \mid a!3.b?y$$

$$\mathtt{a} : {}^{1,1}[\mathtt{int}], \mathtt{b} : {}^{1,1}[\mathtt{bool}] \vdash a?x.b!\mathtt{false} \mid b?y.a!3$$

Strategy for deadlock analysis

1. assign each linear channel a level $\in \mathbb{Z}$

$$\kappa_1,\kappa_2[t]^h$$

2. make sure that channels are used in strict order

$$a\ ?x.b\ !\texttt{false} \mid b\ ?y.a\ !3$$

Strategy for deadlock analysis

1. assign each linear channel a level $\in \mathbb{Z}$

$$\kappa_1,\kappa_2[t]^h$$

2. make sure that channels are used in strict order

$$a^m?x.b^n!\texttt{false} \mid b^n?y.a^m!3$$

Strategy for deadlock analysis

1. assign each linear channel a level $\in \mathbb{Z}$

$$^{\kappa_1,\kappa_2}[t]^{h}$$

2. make sure that channels are used in strict order

$$a^m\overset{<}{\frown}?x.b^n!\mathtt{false} \mid b^n\overset{<}{\frown}?y.a^m!3$$

Typing rules

level of u smaller than
level of any channel in P

Input

$$\frac{\Gamma, x : t \vdash P \qquad h < |\Gamma|}{\Gamma, u : {}^{1,0}[t]^h \vdash u?x.P}$$

level of u smaller than
level of any channel in e

Output

$$\frac{\Gamma \vdash e : t \qquad h < |t|}{\Gamma, u : {}^{0,1}[t]^h \vdash u!e}$$

Problem: most recursive processes are **ill typed**

```
*fibo?(n, r ).
 if n ≤ 1 then
   r!n
 else {
   new a  in
   new b  in {
     fibo!(n - 1, a ) |
     fibo!(n - 2, b ) |
     a ?x.b ?y.r !(x + y)
   }
 }
```

Problem: most recursive processes are **ill typed**

```
*fibo?(n, r²).
 if n ≤ 1 then
   r!n
 else {
   new a  in
   new b  in {
     fibo!(n - 1, a ) |
     fibo!(n - 2, b ) |
     a ?x.b ?y.r²!(x + y)
   }
 }
```

Problem: most recursive processes are **ill typed**

```
*fibo?(n, r²).
 if n ≤ 1 then
   r!n
 else {
   new a  in
   new b¹ in {
     fibo!(n - 1, a ) |
     fibo!(n - 2, b¹) |
     a ?x.b¹?y.r²!(x + y)
   }
 }
```

Problem: most recursive processes are **ill typed**

```
*fibo?(n, r^2).
 if n ≤ 1 then
   r!n
 else {
   new a^0 in
   new b^1 in {
     fibo!(n - 1, a^0) |
     fibo!(n - 2, b^1) |
     a^0?x.b^1?y.r^2!(x + y)
   }
 }
```

Problem: most recursive processes are **ill typed**

```
*fibo?(n, r²).
 if n ≤ 1 then
   r!n
 else {
   new a⁰ in
   new b¹ in {
     fibo!(n - 1, a⁰) |
     fibo!(n - 2, b¹) |
     a⁰?x.b¹?y.r²!(x + y)
   }
 }
```

Is it **bad** if the levels of `a` and `b` don't match that of `r`? **NO!**

Typing rules with level polymorphism

Linear output

$$\frac{\Gamma \vdash e : t \qquad h < |t|}{\Gamma, u : {}^{0,1}[t]^{h} \vdash u \,!\, e}$$

Unlimited output

$$\frac{\Gamma \vdash e : \Updownarrow^{k} t}{\Gamma, u : {}^{0,\omega}[t] \vdash u \,!\, e}$$

Typing rules with level polymorphism

Linear output

$$\frac{\Gamma \vdash e : t \qquad h < |t|}{\Gamma, u : {}^{0,1}[t]^{h} \vdash u\,!\,e}$$

arbitrary up/down shifting on the levels of the channels in e

Unlimited output

$$\frac{\Gamma \vdash e : \Updownarrow^{k} t}{\Gamma, u : {}^{0,\omega}[t] \vdash u\,!\,e}$$

Type reconstruction: how it works 1/2

`a?`x`.b!false | b?`y`.a!3`

1. perform linearity analysis
2. assign fresh integer variables to channels
3. compute constraints
   - $h < k \Rightarrow h + 1 \leq k$
   - $k < h \Rightarrow k + 1 \leq h$
4. use integer programming solver

Type reconstruction: how it works 1/2

^{1,0}[`int`] ^{1,0}[`bool`]

`a?`*x*`.b!false | b?`*y*`.a!3`

^{0,1}[`bool`] ^{0,1}[`int`]

1. perform linearity analysis
2. assign fresh integer variables to channels
3. compute constraints
   - $h < k \Rightarrow h + 1 \leq k$
   - $k < h \Rightarrow k + 1 \leq h$
4. use integer programming solver

Type reconstruction: how it works 1/2

$^{1,0}[\texttt{int}]^h$ $\quad$ $^{1,0}[\texttt{bool}]^k$

$\texttt{a?}x\texttt{.b!false | b?}y\texttt{.a!3}$

$^{0,1}[\texttt{bool}]^k$ $\quad$ $^{0,1}[\texttt{int}]^h$

1. perform linearity analysis
2. assign fresh integer variables to channels
3. compute constraints
   - $h < k \Rightarrow h + 1 \leq k$
   - $k < h \Rightarrow k + 1 \leq h$
4. use integer programming solver

Type reconstruction: how it works 1/2

$^{1,0}[\mathtt{int}]^{h}$ $^{1,0}[\mathtt{bool}]^{k}$

$\mathtt{a?}x\mathtt{.b!false} \mid \mathtt{b?}y\mathtt{.a!3}$

$^{0,1}[\mathtt{bool}]^{k}$ $^{0,1}[\mathtt{int}]^{h}$

1. perform linearity analysis
2. assign fresh integer variables to channels
3. compute constraints
   - $h < k \Rightarrow h + 1 \leq k$
   - $k < h \Rightarrow k + 1 \leq h$
4. use integer programming solver

Type reconstruction: how it works 1/2

$^{1,0}[\texttt{int}]^{h}$ $\quad$ $^{1,0}[\texttt{bool}]^{k}$

$\texttt{a?}x\texttt{.b!false | b?}y\texttt{.a!3}$

$^{0,1}[\texttt{bool}]^{k}$ $\quad$ $^{0,1}[\texttt{int}]^{h}$

1. perform linearity analysis
2. assign fresh integer variables to channels
3. compute constraints
   - $h < k \Rightarrow h + 1 \leq k$
   - $k < h \Rightarrow k + 1 \leq h$
4. use integer programming solver

Type reconstruction: how it works 2/2

```
*fibo?(n, r ).
 if n ≤ 1 then
   r!n
 else {
   new a  in
   new b  in {
     fibo!(n - 1, a ) |     --
     fibo!(n - 2, b ) |     --
     a ?x.b ?y.r !(x + y)   --
   }
 }
```

Type reconstruction: how it works 2/2

```
*fibo?(n, r^n).
 if n ≤ 1 then
   r!n
 else {
   new a^h in
   new b^k in {
     fibo!(n - 1, a^h) |     --
     fibo!(n - 2, b^k) |     --
     a^h?x.b^k?y.r^n!(x + y)   --
   }
 }
```

Type reconstruction: how it works 2/2

```
*fibo?(n, r^n).
 if n ≤ 1 then
   r!n
 else {
   new a^h in
   new b^k in {
     fibo!(n - 1, a^h) |       -- h = n + δ1
     fibo!(n - 2, b^k) |       --
     a^h?x.b^k?y.r^n!(x + y)   --
   }
 }
```

Type reconstruction: how it works 2/2

```
*fibo?(n, r^n).
 if n ≤ 1 then
   r!n
 else {
   new a^h in
   new b^k in {
     fibo!(n - 1, a^h) |      -- h = n + δ1
     fibo!(n - 2, b^k) |      -- k = n + δ2
     a^h?x.b^k?y.r^n!(x + y)  --
   }
 }
```

Type reconstruction: how it works 2/2

```
*fibo?(n, r^n).
 if n ≤ 1 then
   r!n
 else {
   new a^h in
   new b^k in {
     fibo!(n - 1, a^h) |       -- h = n + δ_1
     fibo!(n - 2, b^k) |       -- k = n + δ_2
     a^h?x.b^k?y.r^n!(x + y)   -- h < k < n
   }
 }
```

Outline

Deadlocks vs locks

Definition (deadlock freedom)

P is deadlock free if $P \Longrightarrow Q \not\rightarrow$ implies that Q has no pending communications on linear channels

Deadlocks vs locks

Definition (deadlock freedom)

P is deadlock free if $P \Longrightarrow Q \not\longrightarrow$ implies that Q has no pending communications on linear channels

This is deadlock free

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

$\texttt{new a in } \{\ \texttt{a!3} \mid c\texttt{!a} \mid *c?x.c!x\ \}$

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

Strategy for lock analysis

1. assign each linear channel a finite number $k \in \mathbb{N}$ of tickets

$$^{\kappa_1,\kappa_2}[t]^{h}_{k}$$

2. each time a channel travels, one ticket is consumed
3. channels with no tickets cannot travel

```
new a in { a!3 | c!a | *c?x.c!x }
```

Typing rules with ticket consumption

Linear output

$$\frac{\Gamma \vdash e : \Updownarrow_1^0 t \qquad h < |t|}{\Gamma, u : {}^{0,1}[t]_k^h \vdash u \,!\, e}$$

Unlimited output

$$\frac{\Gamma \vdash e : \Updownarrow_1^k t}{\Gamma, u : {}^{0,\omega}[t] \vdash u \,!\, e}$$

Typing rules with ticket consumption

Linear output

one ticket consumed

$$\frac{\Gamma \vdash e : \Updownarrow_1^0 t \qquad h < |t|}{\Gamma, u : {}^{0,1}[t]_k^h \vdash u ! e}$$

Unlimited output

one ticket consumed

$$\frac{\Gamma \vdash e : \Updownarrow_1^k t}{\Gamma, u : {}^{0,\omega}[t] \vdash u ! e}$$

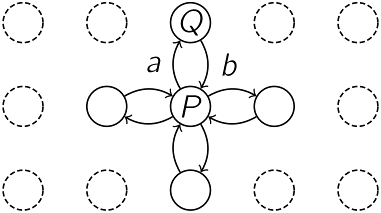
Type reconstruction and lock analysis

1. perform linearity analysis
2. assign fresh *natural* variables for both levels and tickets
3. compute constraints
4. use integer programming solver

Definition (lock freedom)

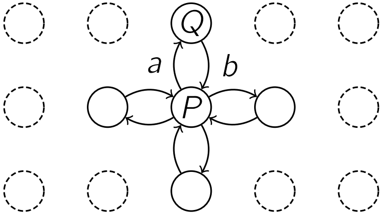
P is lock free if $P \Longrightarrow Q$ and Q has a pending communication on a linear channel c implies $Q \Longrightarrow R$ where the communication on c has occurred

Demo: full-duplex communication



$*node?(a\ ,b\ ).$`new` c `in` $\{\ a\ !c\ \ |\ b\ ?d\ .node!(c\ ,d\ )\ \}$

Demo: full-duplex communication



$$*node?(a^0_0, b^0_0).\texttt{new}\ c^1_3\ \texttt{in}\ \{\ \ a^0_0\,!\,c^1_2\ \mid\ b^0_0?d^1_1.node!(c^1_1, d^1_1)\ \ \}$$

Outline

FAQs

Q: Can multiparty sessions be compiled into the linear π-calculus?
A: Not always

Q: Can these type systems be applied to sessions directly?
A: Yes, attaching levels/tickets to actions

$$?^{h}_{k}\mathtt{int}.T$$

Q: Can these type systems be applied to concrete languages?
A: Yes, with some effort

Q: Do these type systems capture all (dead)lock-free processes?
A: No, there are very reasonable processes that are ill typed

References

- Kobayashi, Pierce, Turner, **Linearity and the pi-calculus**, TOPLAS 1999
- Padovani, **Type Reconstruction for the Linear π-Calculus with Composite and Equi-Recursive Types**, FoSSaCS 2014 (see also long version on my homepage)
- Padovani, **Deadlock and Lock Freedom in the Linear π-Calculus**, LICS 2014 (in TR encoding of multiparty sessions)
- Chen, Padovani, Tosatto, **Type Reconstruction Algorithms for Deadlock-Free and Lock-Free Processes**, in progress

`http://www.di.unito.it/~padovani/Software/hypha/`