Type Reconstruction Algorithms for Deadlock-Free and Lock-Free Linear π-Calculi

Luca Padovani Tzu-Chun Chen Andrea Tosatto

Dipartimento di Informatica – Università di Torino – Italy

Objective

What we want to do

- static analysis for **deadlock and lock detection**
- the problem is **undecidable** in general

How we want to do it

1. **language** for describing communicating processes
   - Kobayashi, Pierce, Turner, **Linearity and the pi-calculus**, TOPLAS 1999
2. **type systems** ensuring deadlock and lock freedom
   - Padovani, **Deadlock and lock freedom in the linear π-calculus**, CSL-LICS 2014
3. **type reconstruction** algorithms
   - this work

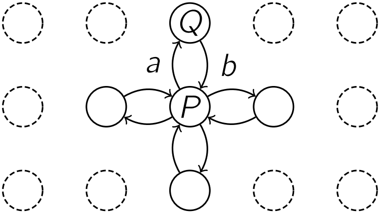
Outline

1. Language
2. The type systems at a glance
3. Type reconstruction algorithms
4. Demo
5. Concluding remarks

Outline

1. Language
2. The type systems at a glance
3. Type reconstruction algorithms
4. Demo
5. Concluding remarks

Example: full-duplex communication




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

a and b are linear channels

- linear = 1 communication
- linearity ⇒ **eventual** synchronization

node is an unlimited channel

- unlimited = any number of communications
- replicated input ⇒ **immediate** synchronization

The linear π-calculus

Unlimited channel

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

≥ 0 communications

Linear channel

$$p[t]$$

1 communication

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

The (dead)lock-free linear π-calculus

Unlimited channel

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

≥ 0 communications

Linear channel

$$p[t]_k^h$$

1 communication

- $h \in \mathbb{Z}$ level
- $k \in \mathbb{N}$ tickets

Padovani, **Deadlock and lock freedom in the linear π-calculus**, CSL-LICS 2014

Deadlock and lock freedom

Definition

P is **deadlock free** if $P \rightarrow^* \texttt{new}\ \tilde{a}\ \texttt{in}\ Q \not\rightarrow$ implies $\neg\mathsf{wait}(a, Q)$

P is **lock free** if $P \rightarrow^* \texttt{new}\ \tilde{a}\ \texttt{in}\ Q$ and $\mathsf{wait}(a, Q)$ implies
$Q \rightarrow^* R$ such that $\mathsf{sync}(a, R)$

Theorem (soundness)

1. $\vdash_0 P$ *implies* P *deadlock free*
2. $\vdash_1 P$ *implies* P *lock free*

Outline

Input and output

level of u smaller than level of any channel in P

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

a❶?x.b❷!x | b❷?x.a❶!x

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

a❷!a❷

Input and output

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

`a`❶`?x.b`❷`!x | b`❷`?x.a`❶`!x`

level of u smaller than
level of any channel in e

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

`a`❷`!a`❷

Recursive processes and level polymorphism

```
*node?(a ,b ).
 new c  in
 { a !c                      -- a blocks c
 | b ?d .node!(c ,d ) } -- b blocks c and d
```

Recursive processes and level polymorphism

```
*node?(a⁰,b⁰).
 new c  in
 { a⁰!c                      -- a blocks c
 | b⁰?d .node!(c ,d ) } -- b blocks c and d
```

Recursive processes and level polymorphism

```
*node?(a⁰,b⁰).
 new c¹ in
 { a⁰!c¹                     -- a blocks c
 | b⁰?d .node!(c¹,d ) } -- b blocks c and d
```

Recursive processes and level polymorphism

```
*node?(a^0,b^0).
 new c^1 in
 { a^0!c^1                     -- a blocks c
 | b^0?d^1.node!(c^1,d^1) } -- b blocks c and d
```

- ⊖ the levels of `c` and `d` don't match those of `a` and `b`

Recursive processes and level polymorphism

```
*node?(a^0,b^0).
 new c^1 in
 { a^0!c^1                      -- a blocks c
 | b^0?d^1.node!(c^1,d^1) } -- b blocks c and d
```

- − the levels of `c` and `d` don't match those of `a` and `b`
- + allow level polymorphism

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

Recursive processes and infinite delegations

```
*node?(a ,b ).
 node!(a ,b )
```

- keeps passing a and b around without using them

Recursive processes and infinite delegations

```
*node?(a①,b①).
 node!(a⓪,b⓪)
```

- ⊖ keeps passing `a` and `b` around without using them
- ⊕ use tickets to limit the number of delegations before use

$$\frac{\Gamma \vdash e : \Uparrow_1^k t}{\Gamma, u : !^\omega[t] \vdash u ! e}$$

Outline

Type reconstruction

▶ Problem statement

Given an **untyped** process P, find Γ, if possible, such that $\Gamma \vdash_k P$

The standard approach

1. use variables for unknown types/levels/tickets

$$u : \alpha$$

2. compute constraints for variables

$$\Gamma \vdash_k P \qquad \Rightarrow \qquad P \blacktriangleright_k \Delta; \varphi$$

3. look for a solution for the constraints φ

Taming complexity

Technically challenging setting

- same channel may have different types ⇒ **no unification**
- depending on whether a channel is linear or unlimited, level constraints may differ ⇒ **conditional constraints**

Taming complexity

Technically challenging setting

- same channel may have different types $\Rightarrow$ **no unification**
- depending on whether a channel is linear or unlimited, level constraints may differ $\Rightarrow$ **conditional constraints**

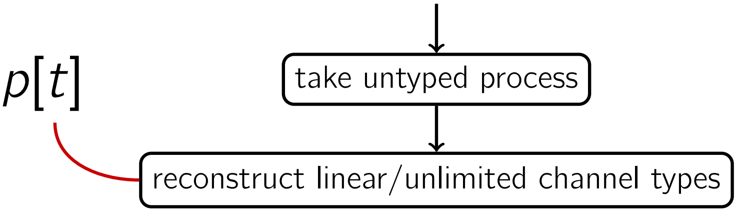
A different strategy

- Igarashi, Kobayashi, **Type reconstruction for linear π-calculus with I/O subtyping**, I&C 2000
- Padovani, **Type reconstruction for the linear π-calculus with composite and equi-recursive types**, FoSSaCS 2014

A more manageable work plan

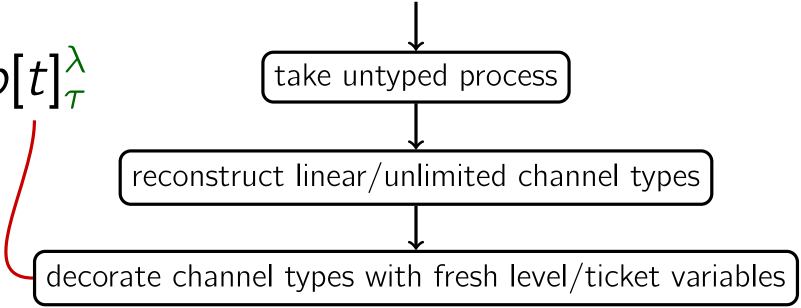
take untyped process

A more manageable work plan

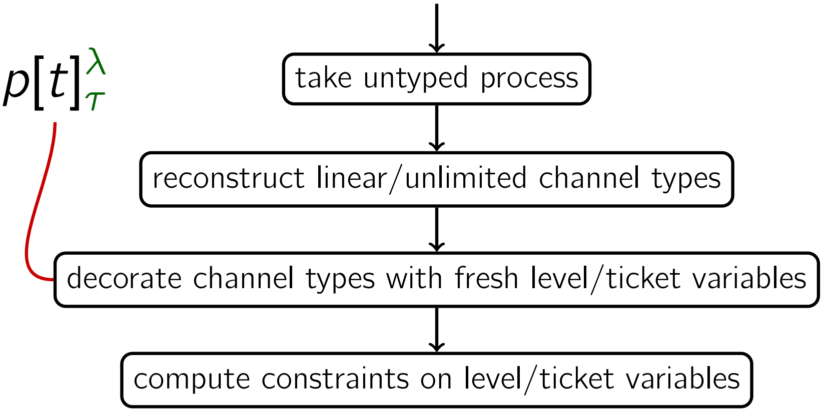


A more manageable work plan

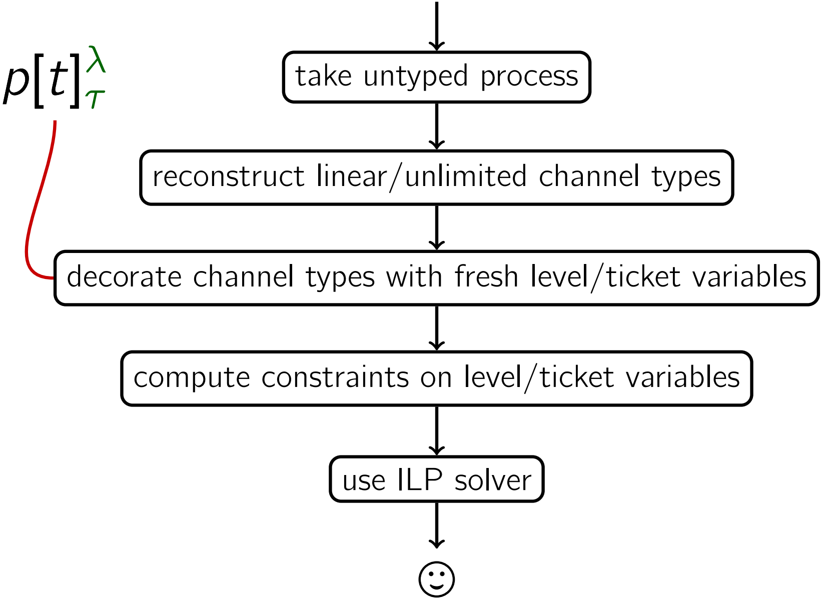
$p[t]^{\lambda}_{\tau}$



A more manageable work plan

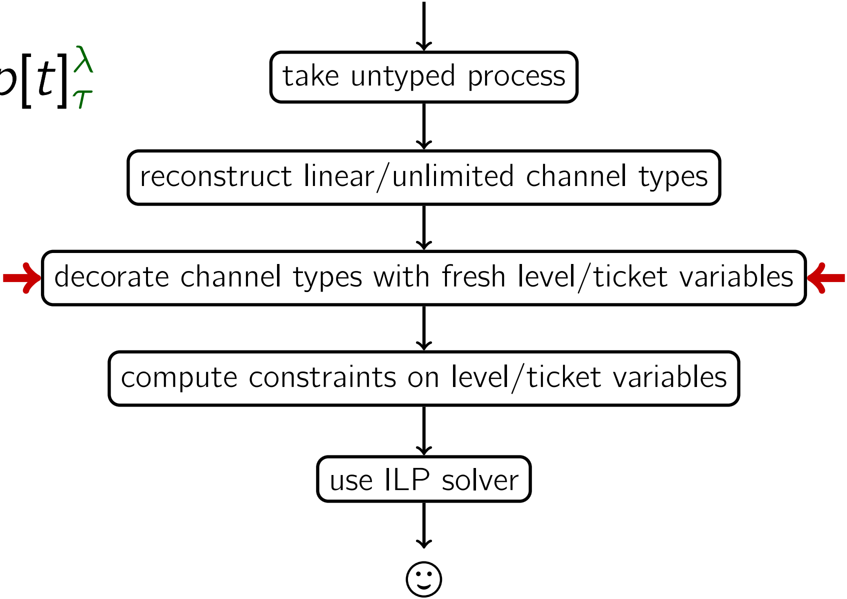


A more manageable work plan



A more manageable work plan

$p[t]^{\lambda}_{\tau}$



Example

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

$$\vdash \;\; a?x.b!x$$

▶

Example

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

$$a : ?[\texttt{int}] \quad , b : ![\texttt{int}] \quad \vdash \; a?x.b!x$$

▶

Example

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

$$a : ?[\texttt{int}]^{\lambda_1}, b : ![\texttt{int}]^{\lambda_2} \ \vdash \ a?x.b!x$$

▶

Example

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

$a : ?[\mathtt{int}]^{\lambda_1}, b : ![\mathtt{int}]^{\lambda_2} \ \vdash \ a?x.b!x$

- $\lambda_1 < \lambda_2$

Example

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

$a : ?[\texttt{int}]^{\lambda_1}, b : ![\texttt{int}]^{\lambda_2} \vdash a?x.b!x$

- $\lambda_1 < \lambda_2$

$a : ![\texttt{int}] \quad , b : ?[\texttt{int}] \quad \vdash b?x.a!x$

-

Example

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

$a : ?[\texttt{int}]^{\lambda_1}, b : ![\texttt{int}]^{\lambda_2} \vdash a?x.b!x$

- $\lambda_1 < \lambda_2$

$a : ![\texttt{int}]^{\lambda_3}, b : ?[\texttt{int}]^{\lambda_4} \vdash b?x.a!x$

- $\lambda_4 < \lambda_3$

Example

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

$$a : ?[\texttt{int}]^{\lambda_1}, b : ![\texttt{int}]^{\lambda_2} \vdash a?x.b!x \qquad \blacktriangleright \ \lambda_1 < \lambda_2$$

$$a : ![\texttt{int}]^{\lambda_3}, b : ?[\texttt{int}]^{\lambda_4} \vdash b?x.a!x \qquad \blacktriangleright \ \lambda_4 < \lambda_3$$

$$a : \#[\texttt{int}]^{\lambda_1}, b : \#[\texttt{int}]^{\lambda_2} \vdash a?x.b!x \mid b?x.a!x \qquad \blacktriangleright \ \begin{array}{l} \lambda_1 = \lambda_3 \\ \lambda_2 = \lambda_4 \\ \lambda_1 < \lambda_2 \\ \lambda_4 < \lambda_3 \end{array}$$

Another example, with level polymorphism

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

$$\begin{array}{l} \texttt{node} : \,!^{\omega}[?[\texttt{int}] \quad \times \,![\texttt{int}] \quad] \\ \qquad a : ?[\texttt{int}] \qquad\qquad\qquad \vdash \texttt{node}!(a, b) \blacktriangleright \\ \qquad b : \,![\texttt{int}] \end{array}$$

Another example, with level polymorphism

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

$$\begin{array}{l} \mathtt{node} : \mathtt{!}^{\omega}[\mathtt{?}[\mathtt{int}]^{\lambda_1} \times \mathtt{!}[\mathtt{int}]^{\lambda_2}] \\ \qquad a : \mathtt{?}[\mathtt{int}]^{\lambda_3} \\ \qquad b : \mathtt{!}[\mathtt{int}]^{\lambda_4} \end{array} \vdash \mathtt{node!}(a, b) \blacktriangleright$$

Another example, with level polymorphism

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

$$\begin{array}{r} \texttt{node} : {!}^{\omega}[?[\texttt{int}]^{\lambda_1} \times {!}[\texttt{int}]^{\lambda_2}] \\ a : ?[\texttt{int}]^{\lambda_3} \\ b : {!}[\texttt{int}]^{\lambda_4} \end{array} \vdash \texttt{node!}(a, b) \blacktriangleright \begin{array}{l} \lambda_3 = \lambda_1 + \lambda_5 \\ \lambda_4 = \lambda_2 + \lambda_5 \end{array}$$

Decorating finite types

$?[\texttt{int}] \quad \Rightarrow \quad ?[\texttt{int}]^{\lambda}$

Decorating **in**finite types

$?[\mathtt{int}] \qquad \Rightarrow \quad ?[\mathtt{int}]^{\lambda}$

$t = ?[\mathtt{int} \times t]$

Decorating **in**finite types

$$?[\texttt{int}] \quad \Rightarrow \quad ?[\texttt{int}]^{\lambda}$$

$$t = ?[\texttt{int} \times t] \quad \overset{?}{\Rightarrow} \quad T = ?[\texttt{int} \times T]^{\lambda_1}$$

Decorating **in**finite types

$$?[\mathtt{int}] \quad \Rightarrow \quad ?[\mathtt{int}]^{\lambda}$$

$$t = ?[\mathtt{int} \times t] \quad \stackrel{?}{\Rightarrow} \quad T = ?[\mathtt{int} \times T]^{\lambda_1}$$

$$\stackrel{?}{\Rightarrow} \quad T = ?[\mathtt{int} \times ?[\mathtt{int} \times T]^{\lambda_2}]^{\lambda_1}$$

Decorating **in**finite types

$$?[\texttt{int}] \quad \Rightarrow \quad ?[\texttt{int}]^{\lambda}$$

$$\begin{aligned}
t = ?[\texttt{int} \times t] \quad &\overset{?}{\Rightarrow} \quad T = ?[\texttt{int} \times T]^{\lambda_1} \\
&\overset{?}{\Rightarrow} \quad T = ?[\texttt{int} \times ?[\texttt{int} \times T]^{\lambda_2}]^{\lambda_1} \\
&\overset{?}{\Rightarrow} \quad T = ?[\texttt{int} \times ?[\texttt{int} \times ?[\texttt{int} \times T]^{\lambda_3}]^{\lambda_2}]^{\lambda_1} \\
&\overset{?}{\Rightarrow} \quad \cdots
\end{aligned}$$

- − We cannot generate infinitely many level variables
- − When do we stop unfolding a type? We don't know!
- + Be lazy!

Decorating **in**finite types

$$?[\texttt{int}] \quad \Rightarrow \quad ?[\texttt{int}]^{\lambda}$$

$$\begin{aligned} t = ?[\texttt{int} \times t] \quad &\overset{?}{\Rightarrow} \quad T = ?[\texttt{int} \times T]^{\lambda_1} \\ &\overset{?}{\Rightarrow} \quad T = ?[\texttt{int} \times ?[\texttt{int} \times T]^{\lambda_2}]^{\lambda_1} \\ &\overset{?}{\Rightarrow} \quad T = ?[\texttt{int} \times ?[\texttt{int} \times ?[\texttt{int} \times T]^{\lambda_3}]^{\lambda_2}]^{\lambda_1} \\ &\overset{?}{\Rightarrow} \quad \cdots \end{aligned}$$

$$\Rightarrow \quad ?[\texttt{int} \times \alpha^{t}]^{\lambda}$$

- − We cannot generate infinitely many level variables
- − When do we stop unfolding a type? We don't know!
- + Be lazy!

Results

Theorem (correctness)

If $P \blacktriangleright_k \Delta; \varphi$ *and* $\sigma \vDash \varphi$, *then* $\sigma\Delta \vdash_k P$

Theorem (completeness)

If $\Gamma \vdash_k P$, *then there exist* Δ, φ, *and* σ *such that*

- $P \blacktriangleright_k \Delta; \varphi$
- $\sigma \vDash \varphi$
- $\Gamma = \sigma\Delta$

Theorem (solver)

There exists an algorithm that, for every φ, *finds a* σ *such that* $\sigma \vDash \varphi$ *whenever such* σ *does exist*

Outline

Outline

On performances

Reconstruction times for hypercube of dimension N and side 5

N	**Proc.**	**Chan.**	**Lin.**	**Constr.**	**Levels**	**Tickets**	**Overall**
1	5	8	0.021	0.006	0.002	0.003	0.032
2	25	80	0.128	0.051	0.009	0.012	0.200
3	125	600	1.439	0.844	0.069	0.124	2.477
[redacted]							

On performances

Reconstruction times for hypercube of dimension N and side 5

N	**Proc.**	**Chan.**	**Lin.**	**Constr.**	**Levels**	**Tickets**	**Overall**
1	5	8	0.021	0.006	0.002	0.003	0.032
2	25	80	0.128	0.051	0.009	0.012	0.200
3	125	600	1.439	0.844	0.069	0.124	2.477
4	625	4000	33.803	26.422	1.116	3.913	65.254

On performances

Reconstruction times for hypercube of dimension N and side 5

N	**Proc.**	**Chan.**	**Lin.**	**Constr.**	**Levels**	**Tickets**	**Overall**
1	5	8	0.021	0.006	0.002	0.003	0.032
2	25	80	0.128	0.051	0.009	0.012	0.200
3	125	600	1.439	0.844	0.069	0.124	2.477
4	625	4000	33.803	26.422	1.116	3.913	65.254

- plenty of room for improvement
- ILP is potentially expensive, but not in practice

Related software

TyPiCal

`http://www-kb.is.s.u-tokyo.ac.jp/~koba/typical/`

- ➕ better precision for **unlimited** channels
  (allows reasoning on races)
- ➖ no **recursive types** and no level **polymorphism**
  (limits recursive processes and network topologies)

Hypha

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