Type-based deadlock analysis of linear communications

Luca Padovani

(with contributions from Tzu-Chun Chen, Luca Novara, Andrea Tosatto)

Dipartimento di Informatica – Università di Torino – Italy

IMT Lucca – 19 marzo 2015

Summary

What we want to achieve

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

How we want to do it

- **types**

What we need

- a **language** of communicating processes
- a **type system**

Outline

1. The linear π-calculus
2. Types for deadlock freedom
3. From the π-calculus to a programming language
4. References

Outline

Syntax of the π-calculus

$\mathbf{0}$	idle process
$u?x.P$	input
$*u?x.P$	persistent input
$u!v.P$	output
$P \mid Q$	parallel composition
$\texttt{new}\ a\ \texttt{in}\ P$	channel creation

Warm-up example

$$*succ?(x, c).c!(x + 1)$$

Warm-up example

$$*succ?(x, c).c!(x + 1)$$

$$succ!(3, a) \mid a?x \cdots$$

Warm-up example

$$*succ?(x, c).c!(x + 1)$$

$$\texttt{new}\ a\ \texttt{in}\ (succ!(3, a) \mid a?x \cdots)$$

Warm-up example

$$*succ?(x, c).c!(x + 1)$$

$$\texttt{new}\ a\ \texttt{in}\ (succ!(3, a) \mid a?x \cdots)$$

$$\texttt{new}\ b\ \texttt{in}\ (succ!(4, b) \mid b?y \cdots)$$

Encoding the recursive Fibonacci function

```
*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)
   }
 }
```

Channel types

$$*succ?(x, c).c!(x + 1)$$

$$\texttt{new}\ a\ \texttt{in}\ (succ!(3, a) \mid a?x \cdots)$$

- *succ*: $0, 1, 2, \ldots$ communications — unlimited
- *a*: **1** communication — linear

Channel types

$$*succ?(x, c).c!(x + 1)$$

$$\texttt{new}\ a\ \texttt{in}\ (succ!(3, a) \mid a?x \cdots)$$

^{1,0}[int]

c: 0, 1, 2, . . . communications unlimited

- *a*: **1** communication linear

Channel types

$$*succ?(x, c).c!(x + 1)$$

$$\texttt{new}\ a\ \texttt{in}\ (succ!(3, a) \mid a?x \cdots)$$

^{1,0}[int]

- *c*: $0, 1, 2, \ldots$ communications — unlimited
- *a*: **1** communication — linear

Channel types

$$*succ?(x, c).c!(x+1)$$

$$\texttt{new}\ a\ \texttt{in}\ (succ!(3, a) \mid a?x \cdots)$$

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

- …cc: $0, 1, 2, \ldots$ communications — unlimited
- a: **1** communication — linear

Channel types

$$*succ?(x, c).c!(x + 1)$$

$$\texttt{new}\ a\ \texttt{in}\ (succ!(3, a) \mid a?x \cdots)$$

$^{0}[\texttt{int}]$

- *succ*: $0, 1, 2, \ldots$ communications — unlimited
- *a*: **1** communication — linear

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

Channel types

$$*succ?(x, c).c!(x + 1)$$

$$\texttt{new}\ a\ \texttt{in}\ (succ!(3, a) \mid a?x \cdots)$$

- *succ*: $0, 1, 2, \ldots$ communications — unlimited
- communication — linear

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

Channel types are useful for...

- catching communication errors in programs
  - like sending a **string** instead of an **integer**
- improving the efficiency of programs
  - a linear channel can be **deallocated** right after usage
- deducing desirable properties of programs
  - communications on linear channels are **deterministic**
  - a **linear** channel is a **promise** of communication

How far can we go with linear channels?

What if we need to communicate **more than once**?

- We could use unlimited channels. . .
- . . . or **continuations**! (i.e. *more linear channels*)

```
new s in eq!(s).
         s!3.
         s!4.
         s?res
```

```
new s0 in eq!(s0).
new s1 in s0!(3,s1).
new s2 in s1!(4,s2).
          s2?res
```

all s_i are linear

How far can we go with linear channels?

What if we need to communicate **more than once**?

- ⊖ We could use unlimited channels. . .
- ⊕ . . . or **continuations**! (i.e. *more linear channels*)

```
new s in eq!(s).
         s!3.
         s!4.
         s?res
```

```
new s0 in eq!(s0).
new s1 in s0!(3,s1).
new s2 in s1!(4,s2).
          s2?res
```

all *si* are linear

How far can we go with linear channels?

What if we need to communicate **more than once**?

- We could use unlimited channels. . .
- . . . or **continuations**! (i.e. *more linear channels*)

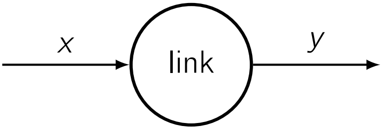
```
new s in eq!(s).
         s!3.
         s!4.
         s?res
```

```
new s0 in eq!(s0).
new s1 in s0!(3,s1).
new s2 in s1!(4,s2).
          s2?res
```

all si are linear

Example: persistent forwarder

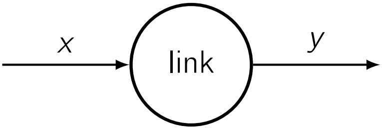
```
*link?(x,y).             -- x and y are linear channels
 x?(v,x').               -- x' is a continuation
 new c in y!(v,c).       -- c is a continuation
          link!(x',c)
```



Types of x and y...

Example: persistent forwarder

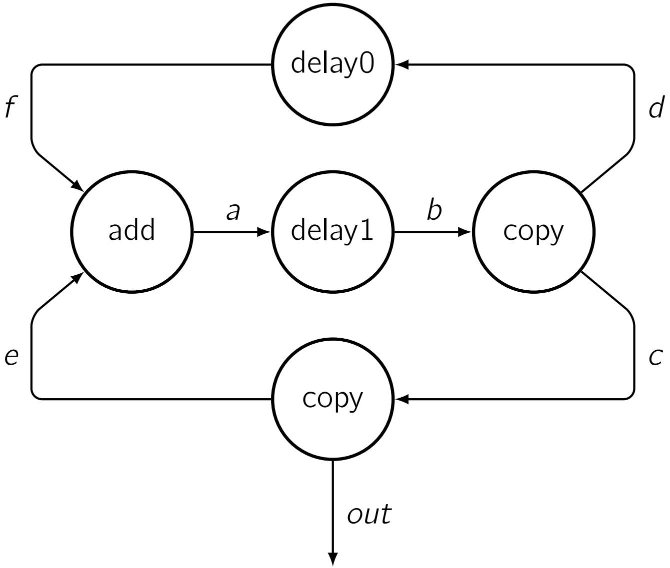
```
*link?(x,y).                -- x and y are linear channels
 x?(v,x').                  -- x' is a continuation
 new c in y!(v,c).          -- c is a continuation
          link!(x',c)
```



Types of x and y...

$$\texttt{x} : t \quad \text{where} \quad t = {}^{1,0}[\texttt{int} \times t]$$
$$\texttt{y} : s \quad \text{where} \quad s = {}^{0,1}[\texttt{int} \times t]$$

Example: the Fibonacci stream network



Outline

A suspicious mismatch

We said. . .

- a **linear** channel is a **promise** of communication

A suspicious mismatch

We said...

- a **linear** channel is a **promise** of communication

But...

Theorem (Kobayashi, Pierce, Turner, 1999)

Each linear channel of a well-typed process is used at most once

The problem

$$\texttt{new}\ a\ \texttt{in}\ \texttt{new}\ b\ \texttt{in}\ (a?x.b!x \mid b?x.a!3)$$

The problem

$$\texttt{new}\ a\ \texttt{in}\ \texttt{new}\ b\ \texttt{in}\ (a?x.b!x \mid b?x.a!3)$$

The problem

$$\texttt{new}\ a\ \texttt{in}\ \texttt{new}\ b\ \texttt{in}\ (a?x.b!x \mid b?x.a!3)$$

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

The problem

$$\texttt{new}\ a\ \texttt{in}\ \texttt{new}\ b\ \texttt{in}\ (a?x.b!x \mid b?x.a!3)$$

The problem

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

$$\texttt{new}\ a\ \texttt{in}\ \texttt{new}\ b\ \texttt{in}\ (a?x.b!x \mid b?x.a!3)$$

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

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

- the process is **well-typed**, *a* and *b* are **linear**
- **no communication** happens

Different processes, same typing

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

Different processes, same typing

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

$$a?x.b!x \mid b?x.a!3$$

Different processes, same typing

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

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

- types do not carry any information regarding **usage order**

Types 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

$$a?x.b!x \mid b?x.a!3$$

Types 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

$$a?x.b!x \mid b?x.a!3$$

Types for deadlock analysis

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

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

$]^m$

2. make sure that channels are used in strict order

$]^m$

$$a?x.b!x \mid b?x.a!3$$

Types for deadlock analysis

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

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

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

2. make sure that channels are used in $^{0,1}[\texttt{int}]^m$ er

$$a?x.b!x \mid b?x$$

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

$[\texttt{int}]^n$

Typing rules

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

Typing rules

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

smaller than
y channel in P

Typing rules

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

$$\frac{\Gamma_1 \vdash e : t \qquad \Gamma_2 \vdash P}{\Gamma_1, \Gamma_2, u : {}^{0,1}[t]^{h} \vdash u!e.P}$$

Typing rules

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

smaller than
ny channel in e

$$\frac{\vdash e : t \qquad \Gamma_2 \vdash P \qquad h < |t| \text{ and } h < |\Gamma_2|}{\Gamma_1, \Gamma_2, u : {}^{0,1}[t]^h \vdash u!e.P}$$

Properties

Definition (deadlock freedom)

P is deadlock free if $P \rightarrow^* Q \not\rightarrow$ implies that in Q there are no pending communications on linear channels

Theorem

If $\Gamma \vdash P$, then P is deadlock free

Sketch.

By contradiction. Suppose $P \rightarrow^* Q \not\rightarrow$ and there is a pending communication on a linear channel in Q.
Then one shows that there is a set of channels with strictly decreasing levels. This contraidcts the fact that Q is *finite* and contains finitely many channels. □

Some examples

$$\frac{}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid a!3.b?x}$$

Some examples

$$\dfrac{\dfrac{}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid a!3.b?x}$$

Some examples

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid a!3.b?x}$$

Some examples

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x} \qquad \dfrac{}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash a!3.b?x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid a!3.b?x}$$

Some examples

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x} \qquad \dfrac{\mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash b?x \quad 0 < 1}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash a!3.b?x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid a!3.b?x}$$

Some examples

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x} \qquad \dfrac{\mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash b?x \quad 0 < 1}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash a!3.b?x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid a!3.b?x}$$

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid b?x.a!3}$$

Some examples

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b\,!\,x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b\,!\,x} \qquad \dfrac{\mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash b?x \quad 0 < 1}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash a\,!\,3.b?x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b\,!\,x \mid a\,!\,3.b?x}$$

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b\,!\,x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b\,!\,x} \qquad \dfrac{}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash b?x.a\,!\,3}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b\,!\,x \mid b?x.a\,!\,3}$$

Some examples

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x} \qquad \dfrac{\mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash b?x \quad 0 < 1}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash a!3.b?x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid a!3.b?x}$$

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x} \qquad \dfrac{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0} \vdash a!3}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash b?x.a!3}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid b?x.a!3}$$

Some examples

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x} \qquad \dfrac{\mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash b?x \quad 0 < 1}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash a!3.b?x}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid a!3.b?x}$$

$$\dfrac{\dfrac{\mathtt{b} : {}^{0,1}[\mathbb{N}]^{1}, x : t \vdash b!x \quad 0 < 1}{\mathtt{a} : {}^{1,0}[\mathbb{N}]^{0}, \mathtt{b} : {}^{0,1}[\mathbb{N}]^{1} \vdash a?x.b!x} \qquad \dfrac{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0} \vdash a!3 \quad 1 \not< 0 \text{ ☠}}{\mathtt{a} : {}^{0,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,0}[\mathbb{N}]^{1} \vdash b?x.a!3}}{\mathtt{a} : {}^{1,1}[\mathbb{N}]^{0}, \mathtt{b} : {}^{1,1}[\mathbb{N}]^{1} \vdash a?x.b!x \mid b?x.a!3}$$

More deadlocks

$$\texttt{new}\ a\ \texttt{in}\ a?x.a!x$$

$$\texttt{new}\ a\ \texttt{in}\ a!a$$

A problem with recursive processes

```
*link?(x ,y ).
 x ?(z,a ).                 -- x blocks y and a
 new b  in y !(z,b ).       -- y blocks a and b
 link!(a ,b )
```

A problem with recursive processes

```
*link?(x⁰,y¹).
 x⁰?(z,a ).                 -- x blocks y and a
 new b  in y¹!(z,b ).       -- y blocks a and b
 link!(a ,b )
```

A problem with recursive processes

```
*link?(x^0,y^1).
 x^0?(z,a^2).                -- x blocks y and a
 new b  in y^1!(z,b ).       -- y blocks a and b
 link!(a^2,b )
```

A problem with recursive processes

```
*link?(x⁰,y¹).
 x⁰?(z,a²).                  -- x blocks y and a
 new b³ in y¹!(z,b³).        -- y blocks a and b
 link!(a²,b³)
```

A problem with recursive processes

```
*link?(x⁰,y¹).
 x⁰?(z,a²).                  -- x blocks y and a
 new b³ in y¹!(z,b³).        -- y blocks a and b
 link!(a²,b³)
```

Problem

- the levels of `a` and `b` don't match those of `x` and `y`
- type error

A problem with recursive processes

```
*link?(x⁰,y¹).
 x⁰?(z,a²).                   -- x blocks y and a
 new b³ in y¹!(z,b³).         -- y blocks a and b
 link!(a²,b³)
```

Problem

- the levels of `a` and `b` don't match those of `x` and `y`
- type error

Solution

- the mismatch is OK as long as it is a translation
- allow level polymorphism

Type reconstruction

▶ Problem statement

Given an untyped process P, find Γ, if there is one, such that $\Gamma \vdash P$

- facility for the programmer
- inference tool of program's properties
  - linearity analysis
  - deadlock analysis
  - possibly more...

Theorem

There exists a type reconstruction algorithm that is both correct and complete

Type reconstruction

▶ Problem statement

Given an untyped process P, find Γ, if there is one, such that $\Gamma \vdash P$

- ⊕ facility for the programmer
- ⊕ inference tool of program's properties
  - linearity analysis
  - deadlock analysis
  - possibly more...

Theorem

There exists a type reconstruction algorithm that is both correct and complete

Type reconstruction

▶ Problem statement

Given an untyped process P, find Γ, if there is one, such that $\Gamma \vdash P$

- ⊕ facility for the programmer
- ⊕ inference tool of program's properties
  - linearity analysis
  - deadlock analysis
  - possibly more...

Theorem

There exists a type reconstruction algorithm that is both correct and complete

Type reconstruction

▶ Problem statement

Given an untyped process P, find Γ, if there is one, such that $\Gamma \vdash P$

- ⊕ facility for the programmer
- ⊕ inference tool of program's properties
  - linearity analysis
  - deadlock analysis
  - possibly more...

Theorem

There exists a type reconstruction algorithm that is both correct and complete

Type reconstruction: a glimpse at the internals

```
*link?(x ,y ).
 x ?(z,a ).               --
 new b  in y !(z,b ).     --
 link!(a ,b )             --
```

1. perform linearity analysis
2. put integer variables in place of (unknown) levels
3. compute constraints on levels
4. use ILP solver

Type reconstruction: a glimpse at the internals

```
*link?(x^n,y^m).
 x^n?(z,a^h).                  --
 new b^k in y^m!(z,b^k).       --
 link!(a^h,b^k)                --
```

1. perform linearity analysis
2. put integer variables in place of (unknown) levels
3. compute constraints on levels
4. use ILP solver

Type reconstruction: a glimpse at the internals

```
*link?(x^n,y^m).
 x^n?(z,a^h).                  -- n < m ∧ n < h
 new b^k in y^m!(z,b^k).       --
 link!(a^h,b^k)                --
```

1. perform linearity analysis
2. put integer variables in place of (unknown) levels
3. compute constraints on levels
4. use ILP solver

Type reconstruction: a glimpse at the internals

```
*link?(x^n,y^m).
 x^n?(z,a^h).                  -- n < m ∧ n < h
 new b^k in y^m!(z,b^k).       -- m < h ∧ m < k
 link!(a^h,b^k)                --
```

1. perform linearity analysis
2. put integer variables in place of (unknown) levels
3. compute constraints on levels
4. use ILP solver

Type reconstruction: a glimpse at the internals

```
*link?(x^n,y^m).
 x^n?(z,a^h).                 -- n < m ∧ n < h
 new b^k in y^m!(z,b^k).      -- m < h ∧ m < k
 link!(a^h,b^k)               -- h = n + t ∧ k = m + t
```

1. perform linearity analysis
2. put integer variables in place of (unknown) levels
3. compute constraints on levels
4. use ILP solver

Type reconstruction: a glimpse at the internals

```
*link?(x^n,y^m).
 x^n?(z,a^h).                   -- n < m ∧ n < h
 new b^k in y^m!(z,b^k).        -- m < h ∧ m < k
 link!(a^h,b^k)                 -- h = n + t ∧ k = m + t
```

1. perform linearity analysis
2. put integer variables in place of (unknown) levels
3. compute constraints on levels
4. use ILP solver

Outline

Problem: programs have structure

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

Problem: programs have structure

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

A deadlock in CML [Reppy, 1999]

`send` a (`recv` b) | `send` b (`recv` a)

Ingredients

- call-by-value λ-calculus
- `open`, `send`, `recv`, `fork`
- **linear** channels

A deadlock in CML [Reppy, 1999]

`send a (recv b) | send b (recv a)`

Ingredients

- call-by-value λ-calculus
- open, … cv, fork
- **linear** channels

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

A deadlock in CML [Reppy, 1999]

`send` a (`recv` b) | `send` b (`recv` a)

Ingredients

- call-by-value λ-calculus
- `open`, `send`, `recv`, `fork`
- **linear** channels

A deadlock in CML [Reppy, 1999]

$$\texttt{send}\ a\ (\texttt{recv}\ b)\ |\ \texttt{send}\ b\ (\texttt{recv}\ a)$$

Ingredients

- call-by-valu
- open, send, recv, fork
- **linear** channels

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

$\texttt{int} \to \texttt{unit}$

A deadlock in CML [Reppy, 1999]

`send` *a* (`recv` *b*) | `send` *b* (`recv` *a*)

Ingredients

- call-by-value λ-calculus
- `open`, `send`, `recv`, `fork`
- **linear** channels

Effects

`send` a (`recv` b) | `send` b (`recv` a)

Effects

`send` a (`recv` b) | `send` b (`recv` a)

Effects

send a (recv b) | send b (recv a)

int $\rightarrow^{n}$ unit & $\bot$

Effects

`send` a (`recv` b) | `send` b (`recv` a)

Effects

$\texttt{int} \rightarrow^{n} \texttt{unit} \ \& \ \bot$

$\texttt{send}\ a\ (\texttt{recv}\ b) \mid \texttt{send}\ b\ (\texttt{recv}\ a)$

Effects

`send` a (`recv` b) | `send` b (`recv` a)

More on arrow types

$$f \equiv \lambda x.(\texttt{send}\ a^m\ x;\ \texttt{send}\ b^n\ x)$$

Which type for f?

$f : \texttt{int} \rightarrow^m \texttt{unit}$

$f : \texttt{int} \rightarrow^n \texttt{unit}$

More on arrow types

$$f \equiv \lambda x.(\texttt{send}\ a^m\ x;\ \texttt{send}\ b^n\ x)$$

Which type for f?

$f : \texttt{int} \to^m \texttt{unit}$

$f : \texttt{int} \to^n \texttt{unit}$

$\texttt{int}\ \&\ n$

$\texttt{int}\ \&\ m$

$(f\ 3);\ \texttt{recv}\ b \mid \texttt{recv}\ a$

More on arrow types

$$f \equiv \lambda x.(\texttt{send}\ a^m\ x;\ \texttt{send}\ b^n\ x)$$

Which type for f?

$f : \texttt{int} \to^m \texttt{unit}$

$f : \texttt{int} \to^n \texttt{unit}$

$\texttt{nt}\ \&\ n$

$\texttt{nt}\ \&\ m$

$^n\ \texttt{unit}\ \&\ \bot$

$(f\ 3);\ \texttt{recv}\ b \mid \texttt{recv}\ a$

$f\ (\texttt{recv}\ a) \mid \texttt{recv}\ b$

Typing abstractions

$$\frac{\Gamma, x : t \vdash e : s \;\&\; \sigma}{\Gamma \vdash \lambda x.e : t \to^{|\Gamma|,\sigma} s \;\&\; \bot}$$

$$\vdash \lambda x.x \qquad : \texttt{int} \to^{\top,\bot} \texttt{int}$$

Typing abstractions

$$\frac{\Gamma, x : t \vdash e : s \mathbin{\&} \sigma}{\Gamma \vdash \lambda x.e : t \to^{|\Gamma|,\sigma} s \mathbin{\&} \bot}$$

$$\vdash \lambda x.x \qquad : \mathtt{int} \to^{\top,\bot} \mathtt{int}$$

$$a : {}^{0,1}[\mathtt{int}]^{n} \vdash \lambda x.(x, a) \qquad : \mathtt{int} \to^{n,\bot} \mathtt{int} \times {}^{0,1}[\mathtt{int}]^{n}$$

Typing abstractions

$$\frac{\Gamma, x : t \vdash e : s \,\&\, \sigma}{\Gamma \vdash \lambda x.e : t \rightarrow^{|\Gamma|,\sigma} s \,\&\, \bot}$$

$$\begin{aligned}
\vdash \lambda x.x \quad & : \texttt{int} \rightarrow^{\top,\bot} \texttt{int} \\
a : {}^{0,1}[\texttt{int}]^{n} \vdash \lambda x.(x, a) \quad & : \texttt{int} \rightarrow^{n,\bot} \texttt{int} \times {}^{0,1}[\texttt{int}]^{n} \\
\vdash \lambda x.(\texttt{send}\ x\ 3) \quad & : {}^{0,1}[\texttt{int}]^{n} \rightarrow^{\top,n} \texttt{unit}
\end{aligned}$$

Typing abstractions

$$\frac{\Gamma, x : t \vdash e : s \;\&\; \sigma}{\Gamma \vdash \lambda x.e : t \to^{|\Gamma|,\sigma} s \;\&\; \bot}$$

$$\begin{array}{rll}
\vdash \lambda x.x & : \texttt{int} \to^{\top,\bot} \texttt{int} \\
a : {}^{0,1}[\texttt{int}]^{n} \vdash \lambda x.(x, a) & : \texttt{int} \to^{n,\bot} \texttt{int} \times {}^{0,1}[\texttt{int}]^{n} \\
\vdash \lambda x.(\texttt{send}\ x\ 3) & : {}^{0,1}[\texttt{int}]^{n} \to^{\top,n} \texttt{unit} \\
a : {}^{1,0}[\texttt{int}]^{n} \vdash \lambda x.(\texttt{recv}\ a{+}x) & : \texttt{int} \to^{n,n} \texttt{int}
\end{array}$$

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \to^{\sigma,\rho} s \mathbin{\&} \tau_1 \qquad \Gamma_2 \vdash e_2 : t \mathbin{\&} \tau_2 \qquad \tau_1 < |\Gamma_2| \qquad \tau_2 < \sigma}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \mathbin{\&} \rho \vee \tau_1 \vee \tau_2}$$

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \to^{\sigma,\rho} s \;\&\; \tau_1 \qquad \Gamma_2 \vdash e_2 : t \;\&\; \tau_2 \qquad \tau_1 < |\Gamma_2| \qquad \tau_2 < \sigma}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \;\&\; \rho \vee \tau_1 \vee \tau_2}$$

$$\vdash (\lambda x.x)\ 3$$

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \to^{\sigma,\rho} s \;\&\; \tau_1 \qquad \Gamma_2 \vdash e_2 : t \;\&\; \tau_2 \qquad \tau_1 < |\Gamma_2| \qquad \tau_2 < \sigma}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \;\&\; \rho \vee \tau_1 \vee \tau_2}$$

$$\vdash (\lambda x.x)\ 3 \qquad \oplus$$

$$a : {}^{1,0}[t]^{n} \vdash (\lambda x.x)\ (\texttt{recv}\ a) \qquad \oplus$$

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \rightarrow^{\sigma,\rho} s \mathbin{\&} \tau_1 \qquad \Gamma_2 \vdash e_2 : t \mathbin{\&} \tau_2 \qquad \tau_1 < |\Gamma_2| \qquad \tau_2 < \sigma}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \mathbin{\&} \rho \vee \tau_1 \vee \tau_2}$$

$\vdash (\lambda x.x)\ 3$ ⊕

$a : {}^{1,0}[t]^{n} \vdash (\lambda x.x)\ (\texttt{recv}\ a)$ ⊕

$a : {}^{1,0}[t]^{n} \vdash (\lambda x.(x, a))\ (\texttt{recv}\ a)$ ⊖

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \to^{\sigma,\rho} s \mathbin{\&} \tau_1 \qquad \Gamma_2 \vdash e_2 : t \mathbin{\&} \tau_2 \qquad \tau_1 < |\Gamma_2| \qquad \tau_2 < \sigma}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \mathbin{\&} \rho \vee \tau_1 \vee \tau_2}$$

$\vdash (\lambda x.x)\ 3$ +

$a : {}^{1,0}[t]^{n} \vdash (\lambda x.x)\ (\texttt{recv}\ a)$ +

$a : {}^{1,0}[t]^{n} \vdash (\lambda x.(x, a))\ (\texttt{recv}\ a)$ −

$a : {}^{1,0}[t \to t]^{0}, b : {}^{1,0}[t]^{1} \vdash (\texttt{recv}\ a)\ (\texttt{recv}\ b)$ +

Outline

Essential bibliography

Linear π-calculus

- Kobayashi, Pierce, Turner, **Linearity and the Pi Calculus** (TOPLAS 1999)

Deadlock freedom for the π-calculus

- Kobayashi, **A Type System for Lock-Free Processes** (I&C 2002)

Type and effect systems

- Amtoft, Nielson, Nielson, **Type and Effect Systems: Behaviours for Concurrency** (Imperial College Press 1999)

Paperware and software

- Padovani, **Deadlock and Lock Freedom in the Linear π-Calculus** (LICS 2014)
- Padovani, **Type Reconstruction for the Linear π-Calculus with Composite and Equi-Recursive Types** (FoSSaCS 2014)
- Padovani, Chen, Tosatto, **Type Reconstruction Algorithms for Deadlock-Free and Lock-Free Linear π-Calculi** (submitted)
- Padovani and Novara, **Types and Effects for Deadlock-Free Higher-Order Programs** (submitted)
- Padovani and Tosatto, **Hypha** (available at `http://di.unito.it/hypha`)

`Slides, papers, links on my home page`