Types for Deadlock-Free Higher-Order Programs

Luca Padovani and Luca Novara

Dipartimento di Informatica, Torino, Italy

FORTE 2015

A bit of context

Behavioural Types for Reliable Large-Scale Software Systems

- COST Action IC1201
- `http://www.behavioural-types.eu`

"...integration of behavioural types into mainstream programming languages..."

A type system for deadlock freedom

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

In a process calculus

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

A type system for deadlock freedom

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

In a process calculus

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

Continuation = what happens afterwards

Input prefix

A type system for deadlock freedom

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

In a process calculus

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

What about a structured programming language?

- I/O may happen within functions, methods, objects, ...
- ...for which we rarely know the "continuation"
- $\Rightarrow$ how do we transpose this typing rule?

Outline

Outline

A minimal programming language

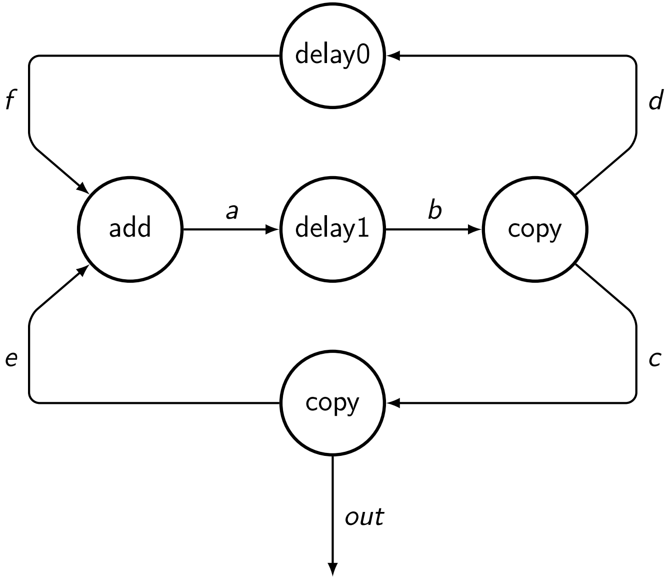
- call-by-value λ-calculus
- `open`, `send`, `recv`, `fork`
- **linear** channels

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

Example: parallel recursive function

```
let rec fibo n c =
  if n ≤ 1 then send c n
  else let a = open () in
       let b = open () in
       fork (fun _ → fibo (n - 1) a);
       fork (fun _ → fibo (n - 2) b);
       send c (recv a + recv b)
```

Example: Kahn process network



Example: Kahn process network

```
let rec link x y =
  let v, x' = recv x in
  let y' = open () in
  send y (v, y');
  link x' y'

let delay v x y =
  let y' = open () in
  send y (v, y');
  link x y'

let rec add x y z =
  let v, x' = recv x in
  let w, y' = recv y in
  let z' = open () in
  send z (v + w, z');
  add x' y' z'
```

```
let rec copy x y z =
  let v, x' = recv x in
  let y' = open () in
  let z' = open () in
  send y (v, y');
  send z (v, z');
  copy x' y' z'

let fibo out =
  let a, b = open (), open () in
  let c, d = open (), open () in
  let e, f = open (), open () in
  fork (fun _ → add e f a);
  fork (fun _ → delay 1 a b);
  fork (fun _ → copy b c d);
  fork (fun _ → copy c e out);
  fork (fun _ → delay 0 d f)
```

Outline

Channel levels in types

$$p\,[\,t\,]^{\,n}$$

$$\langle\ \texttt{send}\ a^{n}\ (\texttt{recv}\ b^{m})\ \rangle \mid \langle\ \texttt{send}\ b^{m}\ (\texttt{recv}\ a^{n})\ \rangle$$

Types are not enough

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

Amtoft, Nielson, Nielson, **Type and Effect Systems: Behaviours for Concurrency**, 1999

Types are not enough

$$\texttt{![int]}^n$$

$$\langle\ \underbrace{\texttt{send}}\ \overbrace{a}\ (\texttt{recv}\ b)\ \rangle \,|\, \langle\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \rangle$$

$$\texttt{![int]}^n \rightarrow \texttt{int} \rightarrow \texttt{unit}$$

Amtoft, Nielson, Nielson, **Type and Effect Systems: Behaviours for Concurrency**, 1999

Types are not enough

$$\langle\ \underbrace{\texttt{send}\ a}_{\texttt{int} \to \texttt{unit}}\ (\texttt{recv}\ b)\ \rangle \mid \langle\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \rangle$$

Amtoft, Nielson, Nielson, **Type and Effect Systems: Behaviours for Concurrency**, 1999

Types are not enough

$$\langle\ \underbrace{\texttt{send}\ a}_{\texttt{int} \to \texttt{unit}}\ (\overbrace{\texttt{recv}}^{?[\texttt{int}]^m \to \texttt{int}}\ \underbrace{b}_{?[\texttt{int}]^m})\ \rangle | \langle\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \rangle$$

Amtoft, Nielson, Nielson, **Type and Effect Systems: Behaviours for Concurrency**, 1999

Types are not enough

$$\overbrace{\texttt{recv}\ b}^{\texttt{int}}$$

$$\langle\ \underbrace{\texttt{send}\ a}_{\texttt{int} \rightarrow \texttt{unit}}\ (\texttt{recv}\ b)\ \rangle \mid \langle\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \rangle$$

Amtoft, Nielson, Nielson, **Type and Effect Systems: Behaviours for Concurrency**, 1999

Types are not enough

$$\langle\ \underbrace{\texttt{send}\ a}_{\texttt{int} \to \texttt{unit}}\ (\overbrace{\texttt{recv}\ b}^{\texttt{int}})\ \rangle \,|\, \langle\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \rangle$$

Amtoft, Nielson, Nielson, **Type and Effect Systems: Behaviours for Concurrency**, 1999

Channel levels in types **and effects**

$\langle$ `send` a (`recv` b) $\rangle \mid \langle$ `send` b (`recv` a) $\rangle$

Channel levels in types **and effects**

$$\texttt{![int]}^{n} \,\&\, \bot$$

$$\langle\ \underbrace{\texttt{send}}\ \overbrace{a}\ (\texttt{recv}\ b)\ \rangle \,|\, \langle\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \rangle$$

$$\texttt{![int]}^{n} \rightarrow \texttt{int} \rightarrow^{n} \texttt{unit} \,\&\, \bot$$

Channel levels in types **and effects**

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

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

Channel levels in types **and effects**

$$?[\texttt{int}]^m \to^m \texttt{int} \ \& \ \bot$$

$$\langle\ \texttt{send}\ a\ (\overbrace{\texttt{recv}}\ \underbrace{b})\ \rangle \,|\, \langle\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \rangle$$

$$?[\texttt{int}]^m \ \& \ \bot$$

Channel levels in types **and effects**

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

$$\langle\ \underbrace{\texttt{send}\ a}\ (\overbrace{\texttt{recv}\ b})\ \rangle \mid \langle\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \rangle$$

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

Channel levels in types **and effects**

$$\langle\ \underbrace{\texttt{send}\ a}_{\texttt{int} \rightarrow^{n} \texttt{unit}\ \&\ \bot}\ (\overbrace{\texttt{recv}\ b}^{\texttt{int}\ \&\ m})\ \rangle \mid \langle\ \underbrace{\texttt{send}\ b}_{\texttt{int} \rightarrow^{m} \texttt{unit}}\ (\overbrace{\texttt{recv}\ a}^{\texttt{int}\ \&\ n})\ \rangle$$

More on arrow types

$$<$$

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

Which type for f?

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

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

More on arrow types

$$<$$

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

Which type for f?

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

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

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

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

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

More on arrow types

$$f \equiv \lambda x\,.\,(\mathtt{send}\ \overset{<}{a^m\ x;\ \mathtt{send}\ b^n}\ x)$$

Which type for f?

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

$$\langle\ \underbrace{(f\ 3)}_{\mathtt{int}\ \&\ m}\ ;\ \overbrace{\mathtt{recv}\ b}^{\mathtt{int}\ \&\ n}\ \rangle \mid \langle\ \mathtt{recv}\ a\ \rangle$$

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

$$\langle\ \underbrace{f}_{\mathtt{int} \rightarrow^n \mathtt{unit}\ \&\ \bot}\ (\overbrace{\mathtt{recv}\ a}^{\mathtt{int}\ \&\ m})\ \rangle \mid \langle\ \mathtt{recv}\ b\ \rangle$$

Typing abstractions $t \to^{\rho,\sigma} s$

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

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

Typing abstractions $\quad t \to^{\rho,\sigma} s$

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

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

$$a : \texttt{!}\,\texttt{[int]}^{n} \vdash \lambda x\,.\,(x\,,\ a) \quad : \texttt{int} \to^{n,\bot} \texttt{int} \times \texttt{!}\,\texttt{[int]}^{n}$$

Typing abstractions $\quad t \to^{\rho,\sigma} s$

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

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

Typing abstractions $\quad t \rightarrow^{\rho,\sigma} s$

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

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

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \to^{\rho,\sigma} s \,\&\, \tau_1 \qquad \Gamma_2 \vdash e_2 : t \,\&\, \tau_2}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \,\&\, \sigma \sqcup \tau_1 \sqcup \tau_2} \qquad \begin{array}{l} \tau_1 < |\Gamma_2| \\ \tau_2 < \rho \end{array}$$

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \rightarrow^{\rho,\sigma} s \,\&\, \tau_1 \qquad \Gamma_2 \vdash e_2 : t \,\&\, \tau_2}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \,\&\, \sigma \sqcup \tau_1 \sqcup \tau_2} \qquad \begin{array}{l} \tau_1 < |\Gamma_2| \\ \tau_2 < \rho \end{array}$$

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

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \rightarrow^{\rho,\sigma} s \,\&\, \tau_1 \qquad \Gamma_2 \vdash e_2 : t \,\&\, \tau_2}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \,\&\, \sigma \sqcup \tau_1 \sqcup \tau_2} \qquad \begin{array}{l} \tau_1 < |\Gamma_2| \\ \tau_2 < \rho \end{array}$$

$$\vdash (\lambda x.x)\ 3$$ ⊕

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

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \rightarrow^{\rho,\sigma} s \,\&\, \tau_1 \qquad \Gamma_2 \vdash e_2 : t \,\&\, \tau_2}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \,\&\, \sigma \sqcup \tau_1 \sqcup \tau_2} \qquad \begin{array}{l} \tau_1 < |\Gamma_2| \\ \tau_2 < \rho \end{array}$$

$$\vdash \texttt{(}\lambda x\texttt{.}x\texttt{)}\ 3$$

$$a : ?[t]^n \vdash \texttt{(}\lambda x\texttt{.}x\texttt{)}\ \texttt{(recv}\ a\texttt{)}$$

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

Typing applications

$$\frac{\Gamma_1 \vdash e_1 : t \rightarrow^{\rho,\sigma} s \,\&\, \tau_1 \qquad \Gamma_2 \vdash e_2 : t \,\&\, \tau_2}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : s \,\&\, \sigma \sqcup \tau_1 \sqcup \tau_2} \quad \begin{array}{l} \tau_1 < |\Gamma_2| \\ \tau_2 < \rho \end{array}$$

$\vdash (\lambda x.x)\ 3$ ⊕

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

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

$a : ?[t \rightarrow s]^0, b : ?[t]^1 \vdash (\texttt{recv}\ a)\ (\texttt{recv}\ b)$ ⊕

Outline

Levels and recursion

```
let rec fibo n c  =
  if n ≤ 1 then send c  1
  else let a  = open () in
       let b  = open () in
       fork (fun _ → fibo (n - 1) a );
       fork (fun _ → fibo (n - 2) b );
       send c  (recv a  + recv b )
```

Levels and recursion

```
let rec fibo n c❸ =
  if n ≤ 1 then send c❸ 1
  else let a  = open () in
       let b  = open () in
       fork (fun _ → fibo (n - 1) a );
       fork (fun _ → fibo (n - 2) b );
       send c❸ (recv a  + recv b )
```

Levels and recursion

```
let rec fibo n c❸ =
  if n ≤ 1 then send c❸ 1
  else let a  = open () in
       let b❷ = open () in
       fork (fun _ → fibo (n - 1) a );
       fork (fun _ → fibo (n - 2) b❷);
       send c❸ (recv a  + recv b❷)
```

Levels and recursion

```
let rec fibo n c❸ =
  if n ≤ 1 then send c❸ 1
  else let a❶ = open () in
       let b❷ = open () in
       fork (fun _ → fibo (n - 1) a❶);
       fork (fun _ → fibo (n - 2) b❷);
       send c❸ (recv a❶ + recv b❷)
```

Different calls with different levels (types)

- `fibo` is well typed only if it is level polymorphic
- `fibo` is recursive ⇒ polymorphic recursion

Polymorphic application

level offset

$$\frac{\Gamma_1 \vdash e_1 : t \rightarrow^{\top,\sigma} s \ \&\ \tau_1 \qquad \Gamma_2 \vdash e_2 : \Uparrow^{n} t \ \&\ \tau_2}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : \Uparrow^{n} s \ \&\ (n + \sigma) \sqcup \tau_1 \sqcup \tau_2} \qquad \begin{array}{l} \tau_1 < |\Gamma_2| \\ \tau_2 < \top \end{array}$$

level offset

Polymorphic application

unlimited function

$$\frac{\Gamma_1 \vdash e_1 : t \rightarrow^{\top,\sigma} s \mathbin{\&} \tau_1 \qquad \Gamma_2 \vdash e_2 : \Uparrow^n t \mathbin{\&} \tau_2}{\Gamma_1 + \Gamma_2 \vdash e_1 e_2 : \Uparrow^n s \mathbin{\&} (n + \sigma) \sqcup \tau_1 \sqcup \tau_2} \qquad \begin{array}{l} \tau_1 < |\Gamma_2| \\ \tau_2 < \top \end{array}$$

Only unlimited functions are level polymorphic

- an unlimited function has no channels in its closure
- the absolute level of its argument does not matter

Outline

Well-typed programs are deadlock free

Definition (deadlock freedom)

P is deadlock free if $P \longrightarrow^* Q \not\longrightarrow$ implies $Q \equiv \langle () \rangle$

Theorem (soundness)

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

Well-typed programs are deadlock free

Definition (deadlock freedom)

P is deadlock free if $P \longrightarrow^* Q \not\longrightarrow$ implies $Q \equiv \langle () \rangle$

Theorem (soundness)

If $\emptyset \vdash P$*, then* P *is deadlock free*

Note

- apparently weak result
- every process P becomes deadlock free if composed with

$$\langle \texttt{fix}\ \lambda x.x \rangle$$

Well-typed programs are interactive

Definition (convergent process)

Convergence is the largest relation s.t. P convergent implies:

1. P has no infinite reduction $P \stackrel{\tau}{\longmapsto} P_1 \stackrel{\tau}{\longmapsto} P_2 \stackrel{\tau}{\longmapsto} \cdots$
2. if $P \stackrel{a!\mathsf{v}}{\longmapsto} Q$, then Q is convergent
3. if $P \stackrel{a?x}{\longmapsto} Q$, then $Q\{\mathsf{v}/x\}$ is convergent for some v

Theorem (interactivity)

Let P be a convergent process such that $a \in \mathrm{fn}(P)$. Then $P \stackrel{\mu_1}{\longmapsto} P_1 \stackrel{\mu_2}{\longmapsto} \cdots \stackrel{\mu_n}{\longmapsto} P_n$ for some $\mu_1, \ldots, \mu_n$ such that $a \notin \mathrm{fn}(P_n)$

Note

- interactivity is still weaker than lock freedom

Well-typed programs are interactive

Definition (convergent process)

Convergence is the largest relation s.t. P convergent implies:

1. P has no infinite reduction $P \stackrel{\tau}{\longmapsto} P_1 \stackrel{\tau}{\longmapsto} P_2 \stackrel{\tau}{\longmapsto} \cdots$
2. if $P \stackrel{a!\mathsf{v}}{\longmapsto} Q$, then Q is convergent
3. if $P \stackrel{a?x}{\longmapsto} Q$, then $Q\{\mathsf{v}/x\}$ is convergent for some v

Theorem (interactivity)

Let P be a convergent process such that $a \in \mathrm{fn}(P)$. Then $P \stackrel{\mu_1}{\longmapsto} P_1 \stackrel{\mu_2}{\longmapsto} \cdots \stackrel{\mu_n}{\longmapsto} P_n$ for some $\mu_1, \ldots, \mu_n$ such that $a \notin \mathrm{fn}(P_n)$

Note

- interactivity is still weaker than lock freedom

Example: Kahn process network

```
...

let fibo out =
  let a, b = open (), open () in
  let c, d = open (), open () in
  let e, f = open (), open () in
  fork (fun _ → add e f a);
  fork (fun _ → delay 1 a b);
  fork (fun _ → copy b c d);
  fork (fun _ → copy c e out);
  fork (fun _ → delay 0 d f)
```

- `fibo` is well typed, hence deadlock free
- the whole network is interactive

⇒ each Fibonacci number is produced in finite time

Outline

Wrap-up

- use effects for tracking levels of used channels
- arrow types need two decorations
  - latent effect
  - static information about channels in the closure
- non-trivial programs require some non-trivial features
  - polymorphic recursion
  - non-regular types (not discussed, see paper)
- the approach scales to call-by-need languages
  - no effects, but annotations on the `IO` monad

Related work

Deadlock freedom and higher-order session calculi

- Wadler, **Propositions as sessions**, ICFP 2012
- Toninho, Caires, Pfenning, **Higher-Order Processes, Functions, and Sessions: A Monadic Integration**, ESOP 2013

- simpler type systems (no channel levels $\Rightarrow$ no effects)
- acyclic network topologies only

Type reconstruction

- tomorrow at COORDINATION, for π calculus