# Types and Effects for Deadlock-Free Higher-Order Concurrent Programs

Luca Padovani and Luca Novara

Dipartimento di Informatica, Torino

BETTY Meeting, Grenoble, 2014

# On the structure of programs

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

# On the structure of programs

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

# Ingredients

- $\lambda$-calculus
- linear channels
- `open`, `send`, `recv`, `fork`

- Reppy, *Concurrent programming in ML*, Cambridge University Press, 99

# A deadlock in CML [Reppy, 99]

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

# A deadlock in CML [Reppy, 99]

$![\texttt{int}]$

$\{\ \underbrace{\texttt{send}}_{}\ \overbrace{a}^{}\ (\texttt{recv}\ b)\ \} \mid \{\ \texttt{send}\ b\ (\texttt{recv}\ a)\ \}$

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

# A deadlock in CML [Reppy, 99]

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

`int` $\rightarrow$ `unit`

# A deadlock in CML [Reppy, 99]

?[int] → int

{ send a (recv b) } | { send b (recv a) }

int → unit

?[int]

# A deadlock in CML [Reppy, 99]

int

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

int → unit

# Outline

1. Motivation
2. Technique
3. Challenges
4. Conclusion

# Detecting deadlocks with priorities

$$\{\ \texttt{send}\ \overbrace{a^n\ (\texttt{recv}\ b^m}^{>})\ \} \mid \{\ \texttt{send}\ \overbrace{b^m\ (\texttt{recv}\ a^n}^{>})\ \}$$

# Detecting deadlocks with priorities **in types**

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

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

# Detecting deadlocks with priorities **in types**

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

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

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

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

# Detecting deadlocks with priorities **in types**

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

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

# Detecting deadlocks with priorities **in types**

$?[\texttt{int}]^m \rightarrow \texttt{int}$

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

$\texttt{int} \rightarrow \texttt{unit}$ $?[\texttt{int}]^m$

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

# Detecting deadlocks with priorities **in types**

int

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

int → unit

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

# Detecting deadlocks with priorities **in types**

int

{ send *a* (recv *b*) }|{ send *b* (recv *a*) }

int → unit

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

# One more try

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

# One more try

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

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

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

# One more try

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

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

# One more try

$?[\mathtt{int}]^m \rightarrow^m \mathtt{int} \mathbin{\&} \bot$

$\{\ \mathtt{send}\ a\ (\overbrace{\mathtt{recv}}\ \underbrace{b})\ \} \mid \{\ \mathtt{send}\ b\ (\mathtt{recv}\ a)\ \}$

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

# One more try

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

# One more try

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

# Priorities vs effects

$$<$$

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

Which type for $f$?

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

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

# Priorities vs effects

$<$

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

Which type for $f$?

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

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

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

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

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

# Priorities vs effects

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

Which type for $f$?

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

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

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

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

# Typing abstractions

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

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

# Typing abstractions

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

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

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

# Typing abstractions

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

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

# Typing abstractions

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

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

# Typing abstractions

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

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

# Typing applications

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

# Typing applications

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

$\vdash (\lambda x . x)\ 3$ ☺

# Typing applications

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

$$\vdash (\lambda x\,.\,x)\ 3 \qquad ☺$$

$$a : p[t]^n \vdash (\lambda x\,.\,x)\ a \qquad ☺$$

# Typing applications

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

$\vdash (\lambda x.x)\ 3$ ☺

$a : p[t]^n \vdash (\lambda x.x)\ a$ ☺

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

# Typing applications

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

$\vdash (\lambda x.x)\ 3$ ☺

$a : p[t]^n \vdash (\lambda x.x)\ a$ ☺

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

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

# Typing applications

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

$\vdash (\lambda x.x)\ 3$ ☺

$a : p[t]^n \vdash (\lambda x.x)\ a$ ☺

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

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

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

# Typing forks

$$\texttt{fork} : \forall i.\forall j.(\texttt{unit} \rightarrow^{i,j} \texttt{unit}) \rightarrow \texttt{unit}$$

- effect masking [Amtoft, Nielson, Nielson, 99]

# Example: parallel Fibonacci

```
let rec fibo n =
  if n ≤ 1 then n
  else let (a, b) = (open(), open()) in
       fork λ_.(send a (fibo (n - 1)));
       fork λ_.(send b (fibo (n - 2)));
       (recv a) + (recv b)
```

# Properties of well-typed programs

**Theorem**

1. *typing is* **preserved** *by reductions*
2. *computations are* **confluent**
3. *well-typed, closed programs are* **deadlock free**
4. *well-typed, convergent programs typed with discrete priorities eventually* **use** *all of their channels*

$$\texttt{send}\ a\ (\texttt{rec}\ x\ x)$$

*(some sensible programs require dense priorities)*

# Polymorphic effects and recursion

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

$$\mathtt{fibo} : \forall i.\mathtt{int} \to {!}[\mathtt{int}]^{i} \to^{\top,i} \mathtt{unit}$$

# Polymorphic effects and recursion

```
let rec fibo n c^i =
  if n ≤ 1 then n
  else let (a^{i−2}, b^{i−1}) = (open(), open()) in
       fork λ_.(fibo (n - 1) a^{i−2});
       fork λ_.(fibo (n - 2) b^{i−1});
       send c^i (recv a^{i−2} + recv b^{i−1})
```

$$\texttt{fibo} : \forall i.\texttt{int} \rightarrow \,!\,[\texttt{int}]^{i} \rightarrow^{\top,i} \texttt{unit}$$

# Polymorphic effects and recursion

```
let rec fibo n c^i =
  if n ≤ 1 then n
  else let (a^{i−2}, b^{i−1}) = (open(), open()) in
       fork λ_.(fibo (n - 1) a^{i−2});
       fork λ_.(fibo (n - 2) b^{i−1});
       send c^i (recv a^{i−2} + recv b^{i−1})
```

$$\texttt{fibo} : \forall i.\texttt{int} \to \texttt{!}\,[\texttt{int}]^{i} \to^{\top,i} \texttt{unit}$$

- type inference for polymorphic recursion is **undecidable**
- ... but is **decidable** when limited to effects [Amtoft, Nielson, Nielson, 99]

# The priority of type variables

$$\lambda x\,.\,\lambda y\,.\,(x,\ y) : \qquad \forall\alpha\,.\forall\beta\,.\alpha \to \beta \to \quad \alpha \times \beta$$

# The priority of type variables

$$\lambda x\,.\,\lambda y\,.\,(x,\ y) : \forall i.\forall j.\forall \alpha^{i}.\forall \beta^{j}.\alpha \to \beta \to^{i,\bot} \alpha \times \beta$$

# Constraints on priority variables

Parallel `send`s

$$\lambda x\,.\,\lambda y\,.\,\mathtt{fork}\ \lambda\_.\,(\mathtt{send}\ x\ 1)\,;\ \mathtt{fork}\ \lambda\_.\,(\mathtt{send}\ y\ 2)$$

$$\forall i.\forall j.\,!\,[\mathtt{int}]^{i} \rightarrow\ !\,[\mathtt{int}]^{j} \rightarrow^{i} \mathtt{unit}$$

# Constraints on priority variables

Parallel sends

$$\lambda x.\lambda y.\texttt{fork}\ \lambda\_.(\texttt{send}\ x\ 1);\ \texttt{fork}\ \lambda\_.(\texttt{send}\ y\ 2)$$

$$\forall i.\forall j.\,!\,[\texttt{int}]^{i} \to\ !\,[\texttt{int}]^{j} \to^{i} \texttt{unit}$$

Sequential sends

$$\lambda x.\lambda y.\texttt{send}\ x\ 1;\ \texttt{send}\ y\ 2$$

$$\forall i.\forall j.(i < j) \Rightarrow\ !\,[\texttt{int}]^{i} \to\ !\,[\texttt{int}]^{j} \to^{i} \texttt{unit}$$

# Recursive types

```
let rec forward x  y  =
  let (v, c   ) = recv x  in
  let d     = open () in
  send y  (v, d   );
  forward c    d
```

```
type  α In  = ?[α × α In   ]     type of x
type α Out  = ![α × α In   ]     type of y
```

$\forall\alpha\ .\qquad \alpha\ \texttt{In} \rightarrow \alpha\ \texttt{Out} \rightarrow \quad \texttt{unit}$

# Recursive types

```
let rec forward x^i y^j =
  let (v, c   ) = recv x^i in
  let d     = open () in
  send y^j (v, d   );
  forward c    d
```

$$\texttt{type}\ \ \alpha\ \texttt{In}^{i}\ \texttt{=}\ \texttt{?}[\alpha \times \alpha\ \texttt{In}^{i}\ \ ]^{i}$$

$$\texttt{type}\ \alpha\ \texttt{Out}^{j}\ \texttt{=}\ \texttt{!}[\alpha \times \alpha\ \texttt{In}^{j}\ \ ]^{j}$$

$$\forall i.\forall j.\forall \alpha\ \ .\qquad \alpha\ \texttt{In}^{i} \rightarrow \alpha\ \texttt{Out}^{j} \rightarrow^{i,}\ \ \texttt{unit}$$

# Recursive types

```
let rec forward x^i y^j =
  let (v, c  ) = recv x^i in
  let d    = open () in
  send y^j (v, d  );
  forward c   d
```

receive from $x$...

...then send on $y$

$$\texttt{type}\ \alpha\ \texttt{In}^i = ?[\alpha \times \alpha\ \texttt{In}^i\ ]^i$$

$$\texttt{type}\ \alpha\ \texttt{Out}^j = ![\alpha \times \alpha\ \texttt{In}^j\ ]^j$$

$$\forall i.\forall j.\forall \alpha\ .(i < j) \Rightarrow \alpha\ \texttt{In}^i \rightarrow \alpha\ \texttt{Out}^j \rightarrow^{i,}\ \texttt{unit}$$

# Recursive types

```
let rec forward x^i y^j =
  let (v, c^{i+1}) = recv x^i in      c received from x
  let d      = open () in
  send y^j (v, d   );
  forward c^{i+1} d
```

```
type  α In^i  = ?[α × α In^{i+1}]^i      non-regular type
type α Out^j = ![α × α In^j  ]^j
```

$$\forall i.\forall j.\forall \alpha\ .(i < j) \Rightarrow \alpha\ \mathtt{In}^{i} \rightarrow \alpha\ \mathtt{Out}^{j} \rightarrow^{i,}\ \ \mathtt{unit}$$

# Recursive types

$$
\begin{array}{l}
\texttt{let rec forward } x^i\ y^j\ \texttt{=} \\
\quad \texttt{let } (v,\ c^{i+1})\ \texttt{= recv } x^i\ \texttt{in} \\
\quad \texttt{let } d^{j+1}\ \texttt{= open () in} \\
\quad \texttt{send } y^j\ (v,\ d^{j+1})\texttt{;} \\
\quad \texttt{forward } c^{i+1}\ d^{j+1}
\end{array}
$$

*d* sent on *y*

$$
\begin{array}{l}
\texttt{type}\ \ \alpha\ \texttt{In}^{i}\ \texttt{=}\ \texttt{?}[\alpha \times \alpha\ \texttt{In}^{i+1}]^{i} \\
\texttt{type}\ \alpha\ \texttt{Out}^{j}\ \texttt{=}\ \texttt{!}[\alpha \times \alpha\ \texttt{In}^{j+1}]^{j}
\end{array}
$$

non-regular type

$$\forall i.\forall j.\forall \alpha\ .(i < j) \Rightarrow \alpha\ \texttt{In}^{i} \rightarrow \alpha\ \texttt{Out}^{j} \rightarrow^{i,}\ \ \texttt{unit}$$

# Recursive types

$\texttt{let rec forward } x^i\ y^j\ \texttt{=}$

$\quad \texttt{let } (v,\ c^{i+1}) \texttt{ = recv } x^i \texttt{ in}$

$\quad \texttt{let } d^{j+1} \texttt{ = open () in}$

$\quad \texttt{send } y^j\ (v,\ d^{j+1});$

$\quad \texttt{forward } c^{i+1}\ d^{j+1}$

$\texttt{type}\ \ \alpha\ \texttt{In}^{i} \texttt{ = ?}[\alpha \times \alpha\ \texttt{In}^{i+1}]^{i}$

$\texttt{type}\ \ \alpha\ \texttt{Out}^{j} \texttt{ = !}[\alpha \times \alpha\ \texttt{In}^{j+1}]^{j}$

unlimited messages only!

$$\forall i.\forall j.\forall \alpha^{\top}.(i < j) \Rightarrow \alpha\ \texttt{In}^{i} \to \alpha\ \texttt{Out}^{j} \to^{i,\top} \texttt{unit}$$

# Recursive types

```
let rec forward x^i y^j =
  let (v, c^{i+1}) = recv x^i in
  let d^{j+1} = open () in
  send y^j (v, d^{j+1});
  forward c^{i+1} d^{j+1}
```

$$\texttt{type}\ \ \alpha\ \texttt{In}^{i} = ?[\alpha \times \alpha\ \texttt{In}^{i+1}]^{i}$$

$$\texttt{type}\ \alpha\ \texttt{Out}^{j} = ![\alpha \times \alpha\ \texttt{In}^{j+1}]^{j}$$

tail applications only!

$$\forall i. \forall j. \forall \alpha^{\top}. (i < j) \Rightarrow \alpha\ \texttt{In}^{i} \rightarrow \alpha\ \texttt{Out}^{j} \rightarrow^{i,\top}\ \texttt{unit}$$

# Example: filter

```
let rec filter p x y =
  let (v, c) = recv x in
  if p v then
    let d = open () in
    fork λ_.(send y (v, d));
    filter p c d
  else
    filter p c y
```

# Concluding remarks

## Question

How hard is it to adapt a type system for deadlock freedom to a real-world programming language?

## Answer

Simple mechanism, but full integration requires advanced features

- priority polymorphism
- priority constraints
- polymorphic recursion (doable)
- higher-rank polymorphism (see [Reppy, 99])
- non-regular types (regular representation possible)

# Variations and ongoing work

## Lazy evaluation

- Monadic I/O
  - Luca Novara

## Type reconstruction for the linear $\pi$-calculus

- Preliminary results
  (monomorphic types, polymorphism on priorities)
  - Tzu-Chun Chen
  - Andrea Tosatto