Deadlock and lock freedom in the linear π-calculus

Luca Padovani – Torino

Objective

Assuring properties of communicating processes

- no deadlocks
- no locks

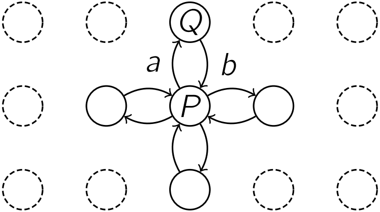
Problem undecidable in general

- Brand, Zafiropulo, **On communicating finite-state machines**, Journal of ACM, 1983
- Cécé, Finkel, **Verification of programs with half-duplex communication**, Inf. & Comp., 2005

Method

- types

Example: full-duplex communication




```
while true do
   send(a, s_P)
   s_Q := receive(b)
   s_P := update(s_P, s_Q)
```

Difficult to handle with current type systems

- recursion
- cyclic network
- several channels

Outline

1. The linear π-calculus
2. Types for deadlock prevention
3. Results
4. Concluding remarks

Outline

1. The linear π-calculus
2. Types for deadlock prevention
3. Results
4. Concluding remarks

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

Why the linear π-calculus?

"communications on linear channels [...] account for up to 50% of communications in a typical program"

Kobayashi, Pierce, Turner 1999

Why the linear π-calculus?

"communications on linear channels [. . .] account for up to 50% of communications in a typical program"

Kobayashi, Pierce, Turner 1999

Linear vs "linearized" channels

$a!\langle 3\rangle.a!\langle 4\rangle \dots$ $\qquad$ $(\nu b)a!\langle 3, b\rangle.(\nu c)b!\langle 4, c\rangle \dots$

- Kobayashi, **Type systems for concurrent programs**, 2002
- Dardha, Giachino, Sangiorgi, **Session types revisited**, 2012

Formulating the problem

Theorem (Kobayashi, Pierce, Turner, 1999)

In a well-typed process, each linear channel is used for communication at most once

Formulating the problem

Theorem (Kobayashi, Pierce, Turner, 1999)

In a well-typed process, each linear channel is used for communication at most once

$$a?(x).b!\langle x\rangle \mid b?(y).a!\langle y\rangle$$

- ⊕ the process is well typed (a and b are linear)
- ⊖ no communication is possible

Formulating the problem

Theorem (Kobayashi, Pierce, Turner, 1999)

In a well-typed process, each linear channel is used for communication ~~at most once~~ **exactly once**

$$a?(x).b!\langle x\rangle \mid b?(y).a!\langle y\rangle$$

- ⊕ the process is well typed (a and b are linear)
- ⊖ no communication is possible

Outline

Strategy

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

$$p[t]^{h}$$

2. make sure that channels are used in strict order

$$a\ ?(x).b\ !\langle x\rangle \mid b\ ?(y).a\ !\langle y\rangle$$

Strategy

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

$$p[t]^{h}$$

2. make sure that channels are used in strict order

$$a^{m}?(x).b^{n}!\langle x\rangle \mid b^{n}?(y).a^{m}!\langle y\rangle$$

Strategy

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

$$p[t]^{h}$$

2. make sure that channels are used in strict order

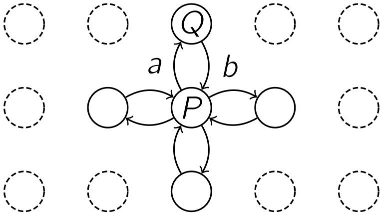
$$a^m?(x).b^n!\langle x\rangle \mid b^n?(y).a^m!\langle y\rangle$$

Typing rules

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

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

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



while **true** **do**

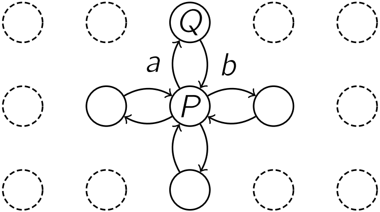
$\quad \mathtt{send}(a, s_P)$

$\quad s_Q := \mathtt{receive}(b)$

$\quad s_P := \mathtt{update}(s_P, s_Q)$

$$*node?(a^{0}, b^{0}).(\nu c^{1})(a^{0}!\langle c^{1}\rangle \mid b^{0}?(d^{1}).node!\langle c^{1}, d^{1}\rangle)$$

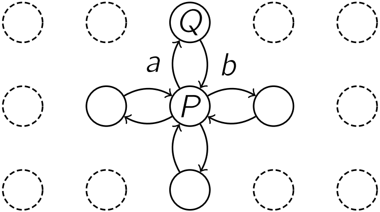
Problem: most recursive processes are **ill typed**




```
while true do
   send(a, s_P)
   s_Q := receive(b)
   s_P := update(s_P, s_Q)
```

$$*node?(a^0, b^0).(\nu c^1)(a^0!\langle c^1\rangle \mid b^0?(d^1).node!\langle c^1, d^1\rangle)$$

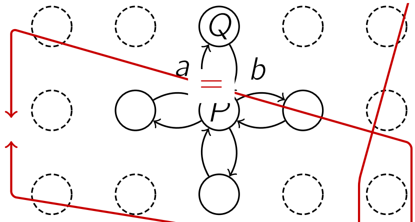
Problem: most recursive processes are **ill typed**




```
while true do
   send(a, s_P)
   s_Q := receive(b)
   s_P := update(s_P, s_Q)
```

$$*node?(a^0, b^0).(\nu c^1)(a^0!\langle c^1\rangle \mid b^0?(d^1).node!\langle c^1, d^1\rangle)$$

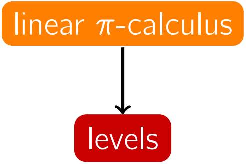
Problem: most recursive processes are **ill typed**




```
while true do
   send(a, s_P)
   s_Q := receive(b)
   s_P := update(s_P, s_Q)
```

$$*node?(a^0, b^0).(\nu c^1)(a^0!\langle c^1\rangle \mid b^0?(d^1).node!\langle c^1, d^1\rangle)$$

Giving up linearity



Giving up linearity

linear π-calculus

levels

more structured types $\Rightarrow$
fewer constraints on levels

Usages

- Kobayashi, *Inf. & Comp.*, 2002
- ...

Session types

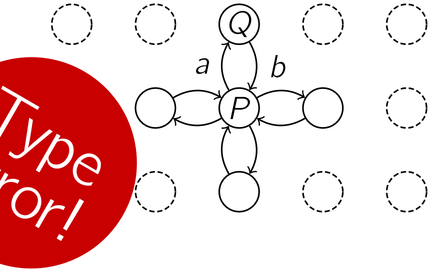
- Dezani *et al.* CONCUR 2008
- ...many more

Example: full-duplex communication

- ⊕ recursive processes are well typed. . .
- ⊖ . . . if they use **one** channel only

Example: full-duplex communication

- ⊕ recursive processes are well typed. . .
- ⊖ . . . if they use **one** channel only



$$
\begin{array}{l}
\texttt{while true do} \\
\quad \texttt{send}(a, s_P) \\
\quad s_Q := \texttt{receive}(b) \\
\quad s_P := \texttt{update}(s_P, s_Q)
\end{array}
$$

recursion + cycles = mutual channel dependencies

Giving up linearity

linear π-calculus

levels

more structured types $\Rightarrow$
fewer constraints on levels

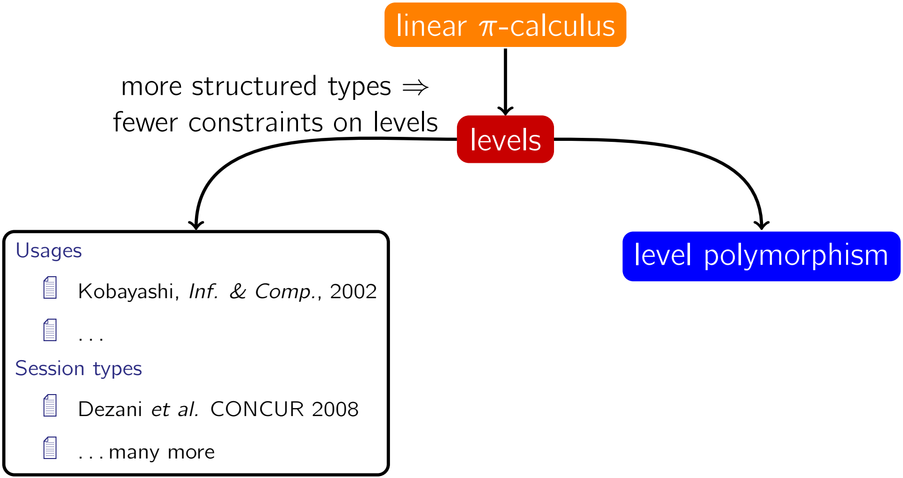
Usages

- Kobayashi, *Inf. & Comp.*, 2002
- ...

Session types

- Dezani *et al.* CONCUR 2008
- ... many more

Giving up linearity



Level polymorphism

$$a?(b^{7}).b?(x).c^{8}!\langle x\rangle \qquad a?(b^{7}, c^{8}).b?(x).c!\langle x\rangle$$

Level polymorphism

$a?(b^{7}).b?(x).c^{8}!\langle x\rangle \qquad a?(b^{7}, c^{8}).b?(x).c!\langle x\rangle$

c free

Level polymorphism

5

6

6

c free

5

$$a?(b^{7}).b?(x).c^{8}!\langle x\rangle \qquad a?(b^{7}, c^{8}).b?(x).c!\langle x\rangle$$

bound

Level polymorphism

$a?(b^7).b?(x).c^8!\langle x\rangle$ $\qquad$ $a?(b^7, c^8).b?(x).c!\langle x\rangle$

c bound

c free

5

When is a channel level-polymorphic?

It depends on the continuation after an **input** on the channel

$a?(x).P$

some free linear channel in $P \Rightarrow a$ monomorphic

When is a channel level-polymorphic?

It depends on the continuation after an **input** on the channel

$a?(x).P$

no free linear channel in $P \Rightarrow a$ polymorphic

When is a channel level-polymorphic?

It depends on the continuation after an **input** on the channel

$a?(x).P$

Typing of linear π-calculus (Kobayashi, Pierce, Sangiorgi, 1999)

$$\frac{\vdash P \qquad \text{no free linear channel in } P}{\vdash *a?(x).P}$$

no free linear

When is a channel level-polymorphic?

It depends on the continuation after an **input** on the channel

$a?(x).P$

Typing of linear π-calculus (Kobayashi, Pierce, Sangiorgi, 1999)

$$\frac{\vdash P \qquad \text{no free linear channel in } P}{\vdash *a?(x).P}$$

Key observation

All channels used for replicated inputs are level-polymorphic
This is great news! recursion $\Rightarrow$ replication

Typing rules (revised)

Linear output

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

Unlimited output

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

Typing rules (revised)

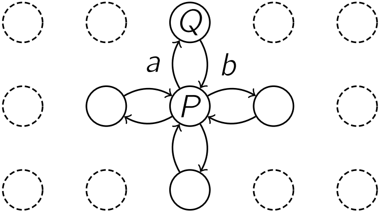
Linear output

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

Unlimited output

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

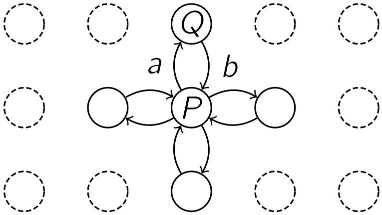
Example: full-duplex communication



while true do
  send(a, s_P)
  $s_Q :=$ receive(b)
  $s_P :=$ update(s_P, s_Q)

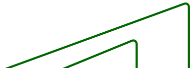
$$*node?(a^0, b^0).(\nu c^1)(a^0!\langle c^1\rangle \mid b^0?(d^1).node!\langle c^1, d^1\rangle)$$

Example: full-duplex communication




```
while true do
    send(a, s_P)
    s_Q := receive(b)
    s_P := update(s_P, s_Q)
```

$$*\mathit{node}?(a^0, b^0).(\nu c^1)(a^0!\langle c^1\rangle \mid b^0?(d^1).\mathit{node}!\langle c^1, d^1\rangle)$$



Outline

Result 1: deadlock freedom

Definition

P is **deadlock free** if $P \rightarrow^* (\nu \tilde{a})Q \not\rightarrow$ implies $\neg\mathrm{wait}(a, Q)$ for all a

Result 1: deadlock freedom

Definition

P is **deadlock free** if $P \rightarrow^* (\nu\tilde{a})Q \nrightarrow$ implies $\neg\mathsf{wait}(a, Q)$ for all a

Theorem

P closed and well typed implies P deadlock free

Result 1: deadlock freedom

Definition

P is **deadlock free** if $P \rightarrow^* (\nu \tilde{a})Q \not\rightarrow$ implies $\neg\mathsf{wait}(a, Q)$ for all a

Theorem

P closed and well typed implies P deadlock free

- but this is well typed...

$$c!\langle a\rangle \mid *c?(x).c!\langle x\rangle \mid a!\langle 1984\rangle$$

Result 2: lock freedom

Definition

P is **lock free** if $P \rightarrow^* (\nu \tilde{a})Q$ and $\mathrm{wait}(a, Q)$ implies

$$Q \rightarrow^* R \text{ such that } \mathrm{sync}(a, R)$$

Result 2: lock freedom

Definition

P is **lock free** if $P \rightarrow^* (\nu\tilde{a})Q$ and $\mathsf{wait}(a, Q)$ implies

$$Q \rightarrow^* R \text{ such that } \mathsf{sync}(a, R)$$

on on a is possible

$$\#[t]^{h}_{k}$$

Result 2: lock freedom

Definition

P is **lock free** if $P \to^* (\nu\tilde{a})Q$ and $\mathrm{wait}(a, Q)$ implies

$$Q \to^* R \text{ such that } \mathrm{sync}(a, R)$$

$$\#[t]^{h}$$

tickets = max no. of travels

Theorem

P closed and well typed with levels $\in \mathbb{N}$ implies P lock free

Result 3: expressiveness

- characterizing well-typed processes is an open problem, but...

Result 3: expressiveness

- characterizing well-typed processes is an open problem, but...

Theorem (not in proceedings, see long version)

If you can draw a **message sequence chart** for your protocol, then you can realize the protocol with a well-typed process

Result 3: expressiveness

- ⊖ characterizing well-typed processes is an open problem, but. . .

Theorem (not in proceedings, see long version)

If you can draw a **message sequence chart** for your protocol, then you can realize the protocol with a well-typed process

- ⊕ variable number of channels
- ⊕ variable number of processes
- ⊕ variable network topology

Outline

Some shortcomings

- Global ordering of events

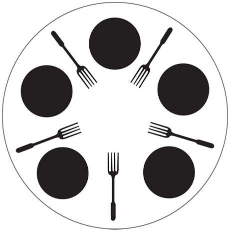
$$a!\langle 3\rangle \mid b!\langle 4\rangle \mid \texttt{if}\ \cdots\ \begin{array}{l}\texttt{then}\ a?(x).b?(y)\cdots\\ \texttt{else}\ b?(y).a?(x)\cdots\end{array}$$

Some shortcomings

- Global ordering of events

$$a!\langle 3\rangle \mid b!\langle 4\rangle \mid \texttt{if}\ \cdots\ \texttt{then}\ a?(x).b?(y)\cdots$$
$$\texttt{else}\ b?(y).a?(x)\cdots$$

- Linear/unlimited dichotomy



Giachino, Kobayashi, Laneve, **Deadlock analysis of unbounded process networks**, CONCUR 2014 (to appear)

Further work

- Type reconstruction (with Tzu-Chun Chen and Andrea Tosatto)
  - Igarashi, Kobayashi, **Type Reconstruction for Linear π-Calculus with I/O Subtyping**, Inf. & Comp. 2000
  - Padovani, **Type Reconstruction for the Linear π-Calculus with Composite and Equi-Recursive Types**, FoSSaCS 2014

$$\left.\begin{array}{r}\text{level/ticket variables}\\ \text{linear constraints}\end{array}\right\}\text{linear (integer) programming problem}$$

- Extension to higher-order languages (with Luca Novara)
  - Padovani, Novara, **Types for Deadlock-Free Higher-Order Concurrent Programs**, 2014 (tech report on my homepage)