Deadlock and Lock Freedom in the Linear π-calculus

LICS 2014 paper of Luca Padovani

Mariangiola Dezani – Dipartimento di Informatica – Torino

Outline

Lock freedom in interleaved sessions

Lock freedom = each communication can be completed

- global types assure lock freedom inside a single session
- interleaved sessions are needed (for example for delegation)
- global types do not assure lock freedom in presence of interleaved sessions

$$a[1](y).b[1](z).y?(2,x).z!\langle 2,x\rangle \mid \bar{a}[2](y).\bar{b}[2](z).z?(1,x').y!\langle 1,x'\rangle$$

$$\downarrow$$

$$\cdots$$

$$\downarrow$$

$$(\nu s)(\nu s')(s[1]?(2,x).s'[1]!\langle 2,x\rangle \mid s'[2]?(1,x').s[2]!\langle 1,x'\rangle)$$

Lock freedom in interleaved sessions

Lock freedom = each communication can be completed

- global types assure lock freedom inside a single session
- interleaved sessions are needed (for example for delegation)
- global types do not assure lock freedom in presence of interleaved sessions

$$a[1](y).b[1](z).y?(2,x).z!\langle 2,x\rangle \mid \bar{a}[2](y).\bar{b}[2](z).z?(1,x').y!\langle 1,x'\rangle$$

$$\downarrow$$

$$\cdots$$

$$\downarrow$$

$$(\nu s)(\nu s')(s[1]?(2,x).s'[1]!\langle 2,x\rangle \mid s'[2]?(1,x').s[2]!\langle 1,x'\rangle)$$

Lock freedom in interleaved sessions

Lock freedom = each communication can be completed

- global types assure lock freedom inside a single session
- interleaved sessions are needed (for example for delegation)
- global types do not assure lock freedom in presence of interleaved sessions

$$a[1](y).b[1](z).y?(2,x).z!\langle 2,x\rangle \mid \bar{a}[2](y).\bar{b}[2](z).z?(1,x').y!\langle 1,x'\rangle$$

$$\downarrow$$

$$\cdots$$

$$\downarrow$$

$$(\nu s)(\nu s')(s[1]?(2,x).s'[1]!\langle 2,x\rangle \mid s'[2]?(1,x').s[2]!\langle 1,x'\rangle)$$

Lock freedom in interleaved sessions

Lock freedom = each communication can be completed

- global types assure lock freedom inside a single session
- interleaved sessions are needed (for example for delegation)
- global types do not assure lock freedom in presence of interleaved sessions

$$a[1](y).b[1](z).y?(2,x).z!\langle 2,x\rangle \mid \bar{a}[2](y).\bar{b}[2](z).z?(1,x').y!\langle 1,x'\rangle$$

$$\downarrow$$

$$\cdots$$

$$\downarrow$$

$$(\nu s)(\nu s')(s[1]?(2,x).s'[1]!\langle 2,x\rangle \mid s'[2]?(1,x').s[2]!\langle 1,x'\rangle)$$

Lock freedom in interleaved sessions

Lock freedom = each communication can be completed

- global types assure lock freedom inside a single session
- interleaved sessions are needed (for example for delegation)
- global types do not assure lock freedom in presence of interleaved sessions

$$a[1](y).b[1](z).y?(2,x).z!\langle 2,x\rangle \mid \bar{a}[2](y).\bar{b}[2](z).z?(1,x').y!\langle 1,x'\rangle$$

$$\downarrow$$

$$\cdots$$

$$\downarrow$$

$$(\nu s)(\nu s')(s[1]?(2,x).s'[1]!\langle 2,x\rangle \mid s'[2]?(1,x').s[2]!\langle 1,x'\rangle)$$

Lock freedom in interleaved sessions

Lock freedom = each communication can be completed

- global types assure lock freedom inside a single session
- interleaved sessions are needed (for example for delegation)
- global types do not assure lock freedom in presence of interleaved sessions

$$a[1](y).b[1](z).y?(2,x).z!\langle 2,x\rangle \mid \bar{a}[2](y).\bar{b}[2](z).z?(1,x').y!\langle 1,x'\rangle$$

$$\downarrow$$
$$\dots$$
$$\downarrow$$

$$(\nu s)(\nu s')(s[1]?(2,x).s'[1]!\langle 2,x\rangle \mid s'[2]?(1,x').s[2]!\langle 1,x'\rangle)$$

Lock freedom in interleaved sessions

Lock freedom = each communication can be completed

- global types assure lock freedom inside a single session
- interleaved sessions are needed (for example for delegation)
- global types do not assure lock freedom in presence of interleaved sessions

$$a[1](y).b[1](z).y?(2,x).z!\langle 2,x\rangle \mid \bar{a}[2](y).\bar{b}[2](z).z?(1,x').y!\langle 1,x'\rangle$$

$$\downarrow$$
$$\dots$$
$$\downarrow$$

$$(\nu s)(\nu s')(s[1]?(2,x).s'[1]!\langle 2,x\rangle \mid s'[2]?(1,x').s[2]!\langle 1,x'\rangle)$$

From binary sessions to the linear π-calculus

- Dardha, Giachino, Sangiorgi, **Session types revisited**, PPDP 2012

Session	**Linear π-calculus**
$y!\langle 45\rangle.y?(x)$	$(\nu b)a!\langle 45, b\rangle.b?(x)$
$y : !\texttt{int}.?\texttt{bool}$	$a : ![\texttt{int} \times ?[\texttt{bool}]]$

The linear π-calculus

Each linear channel is used exactly once

Each unlimited channel is used without restrictions

benefits:

- specialised behavioural equivalences for reasoning about communication optimisations
- efficient implementation of linear channels
- communications on linear channels are deterministic and confluent

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

The linear π-calculus

Each linear channel is used exactly once

Each unlimited channel is used without restrictions

benefits:

- specialised behavioural equivalences for reasoning about communication optimisations
- efficient implementation of linear channels
- communications on linear channels are deterministic and confluent

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

The linear π-calculus

Each linear channel is used exactly once

Each unlimited channel is used without restrictions

benefits:

- specialised behavioural equivalences for reasoning about communication optimisations
- efficient implementation of linear channels
- communications on linear channels are deterministic and confluent

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

The linear π-calculus

Each linear channel is used exactly once

Each unlimited channel is used without restrictions

benefits:

- specialised behavioural equivalences for reasoning about communication optimisations
- efficient implementation of linear channels
- communications on linear channels are deterministic and confluent

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

The linear π-calculus

Each linear channel is used exactly once

Each unlimited channel is used without restrictions

benefits:

- specialised behavioural equivalences for reasoning about communication optimisations
- efficient implementation of linear channels
- communications on linear channels are deterministic and confluent

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

The linear π-calculus

Each linear channel is used exactly once

Each unlimited channel is used without restrictions

benefits:

- specialised behavioural equivalences for reasoning about communication optimisations
- efficient implementation of linear channels
- communications on linear channels are deterministic and confluent

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

The linear π-calculus

Each linear channel is used exactly once

Each unlimited channel is used without restrictions

benefits:

- specialised behavioural equivalences for reasoning about communication optimisations
- efficient implementation of linear channels
- communications on linear channels are deterministic and confluent

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

Deadlock and lock freedom

Definition (deadlock freedom)

no pending communications on linear channels in irreducible states

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

Definition (lock freedom)

each pending communication on a linear channel can be completed

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

Deadlock and lock freedom

Definition (deadlock freedom)

no pending communications on linear channels in irreducible states

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

Definition (lock freedom)

each pending communication on a linear channel can be completed

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

Deadlock and lock freedom

Definition (deadlock freedom)

no pending communications on linear channels in irreducible states

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

Definition (lock freedom)

each pending communication on a linear channel can be completed

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

Deadlock and lock freedom

Definition (deadlock freedom)

no pending communications on linear channels in irreducible states

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

Definition (lock freedom)

each pending communication on a linear channel can be completed

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

Types of channels

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

p is the polarity: ? or ! or both or nothing
ι is the multiplicity: 1 or ω
t is the type of the message

Types of channels

$$p^{1}[t]^{m}_{\downarrow}$$

$+$ level ($\Rightarrow$ deadlock freedom)

levels m enforce an order on the use of channels: channels with lower levels must be used before channels with higher levels

p is the polarity: ? or ! or both or nothing
ι is the multiplicity: 1 or ω
t is the type of the message

Types of channels

$$p^{\iota}[t]^{m}_{n}$$

$\downarrow$

$+$ tickets ($\Rightarrow$ lock freedom)

tickets n limit the number of travels that channels can do: each time a channel is sent as a message, one ticket is removed from its type; a channel with no ticket cannot travel and may be used only for performing a communication

p is the polarity: ? or ! or both or nothing
ι is the multiplicity: 1 or ω
t is the type of the message

Types of linear channels and unlimited channels

$$p[t]^{m}_{n}$$

linear channel

$$p[t]$$

unlimited channel

p is the polarity: ? or ! or both or nothing

t is the type of the message

m is the level

n is the tickets

no multiplicity

Types of linear channels and unlimited channels

$$p[t]^{m}_{n}$$

linear channel

$$p[t]$$

unlimited channel

p is the polarity: ? or ! or both or nothing
t is the type of the message
m is the level
n is the tickets
no multiplicity

Types of linear channels and unlimited channels

$$p[t]^{m}_{n}$$

linear channel

$$p[t]$$

unlimited channel

p is the polarity: ? or ! or both or nothing
t is the type of the message
m is the level
n is the tickets
no multiplicity

Types of linear channels and unlimited channels

$$p[t]^{m}_{n}$$

linear channel

$$p[t]$$

unlimited channel

p is the polarity: ? or ! or both or nothing
t is the type of the message
m is the level
n is the tickets
no multiplicity

Types of linear channels and unlimited channels

$$p[t]^{m}_{n}$$

linear channel

$$p[t]$$

unlimited channel

p is the polarity: ? or ! or both or nothing
t is the type of the message
m is the level
n is the tickets
no multiplicity

Conditions for deadlock freedom

Inputs

$$\frac{u \text{ has lower level than all the channels in } P}{u?(x).P \text{ is well typed}}$$

Outputs

$$\frac{u \text{ has lower level than } v}{u!\langle v \rangle \text{ is well typed}}$$

Conditions for deadlock freedom

Inputs

$$\frac{u \text{ has lower level than all the channels in } P}{u?(x).P \text{ is well typed}}$$

Outputs

$$\frac{u \text{ has lower level than } v}{u!\langle v \rangle \text{ is well typed}}$$

Examples of deadlock freedom

$$a : \{?, !\}[\texttt{int}]^{m}, b : \{?, !\}[\texttt{bool}]^{h} \vdash a^{m}?(x).b^{h}!\langle x\rangle \mid b^{h}?(y).a^{m}!\langle y\rangle$$

Examples of deadlock freedom

$$a : \{?, !\}[\texttt{int}]^{m}, b : \{?, !\}[\texttt{bool}]^{h} \vdash a^{m}?(x).b^{h}!\langle x\rangle \mid b^{h}?(y).a^{m}!\langle y\rangle$$

Examples of deadlock freedom

$$a : \{?, !\}[\texttt{int}]^m, b : \{?, !\}[\texttt{bool}]^h \vdash a^m?(x).b^h!\langle x\rangle \mid b^h?(y).a^m!\langle y\rangle$$

$<$ $\not<$

$$a : \{?, !\}[\texttt{int}]^m \vdash a^m?(x).a^m!\langle x\rangle$$

Examples of deadlock freedom

$$a : \{?, !\}[\texttt{int}]^m, b : \{?, !\}[\texttt{bool}]^h \vdash a^m?(x).b^h!\langle x\rangle \mid b^h?(y).a^m!\langle y\rangle$$

$$a : \{?, !\}[\texttt{int}]^m \vdash a^m?(x).a^m!\langle x\rangle$$

Examples of deadlock freedom

$$a : \{?, !\}[\texttt{int}]^m, b : \{?, !\}[\texttt{bool}]^h \vdash a^m?(x).b^h!\langle x\rangle \mid b^h?(y).a^m!\langle y\rangle$$

$$a : \{?, !\}[\texttt{int}]^m \vdash a^m?(x).a^m!\langle x\rangle$$

$$a : \{?, !\}[\mu\alpha.?[\alpha]^m]^m \vdash a^m!\langle a^m\rangle$$

Examples of deadlock freedom

$$a : \{?, !\}[\texttt{int}]^m, b : \{?, !\}[\texttt{bool}]^h \vdash a^m?(x).b^h!\langle x\rangle \mid b^h?(y).a^m!\langle y\rangle$$

$$a : \{?, !\}[\texttt{int}]^m \vdash a^m?(x).a^m!\langle x\rangle$$

$$a : \{?, !\}[\mu\alpha.?[\alpha]^m]^m \vdash a^m!\langle a^m\rangle$$

Lock freedom

Condition

$$\frac{v\text{'s tickets} > 0 \text{ (and 1 unit is consumed)}}{u!\langle v\rangle \text{ is well typed}}$$

Example

$$c : \{?,!\}^{\omega}[?[\texttt{int}]^{m}{}_{n}] \vdash c!\langle a^{m}{}_{n+1}\rangle \mid *c?(x^{m}{}_{n}).c!\langle x^{m}{}_{n}\rangle \mid a^{m}{}_{0}!\langle 1984\rangle$$

Lock freedom

Condition

$$\frac{v\text{'s tickets} > 0 \text{ (and 1 unit is consumed)}}{u!\langle v\rangle \text{ is well typed}}$$

Example

$$c : \{?, !\}^{\omega}[?[\texttt{int}]^{m}{}_{n}] \vdash c!\langle a^{m}{}_{n+1}\rangle \mid *c?(x^{m}{}_{n}).c!\langle x^{m}{}_{n}\rangle \mid a^{m}{}_{0}!\langle 1984\rangle$$

Lock freedom

Condition

$$\frac{v\text{'s tickets} > 0 \text{ (and 1 unit is consumed)}}{u!\langle v\rangle \text{ is well typed}}$$

Example

$\hookrightarrow$ any channel sent on c must have $n+1$ tickets

$$c : \{?,!\}^{\omega}[?[\texttt{int}]^{m}{}_{n}] \vdash c!\langle a^{m}{}_{n+1}\rangle \mid *c?(x^{m}{}_{n}).c!\langle x^{m}{}_{n}\rangle \mid a^{m}{}_{0}!\langle 1984\rangle$$

Lock freedom

Condition

$$\frac{v\text{'s tickets} > 0 \text{ (and 1 unit is consumed)}}{u!\langle v\rangle \text{ is well typed}}$$

Example

$\hookrightarrow$ any channel sent on c must have $n+1$ tickets

$$c : \{?,!\}^{\omega}[?[\texttt{int}]^{m}{}_{n}] \vdash c!\langle a^{m}{}_{n+1}\rangle \mid *c?(x^{m}{}_{n}).c!\langle x^{m}{}_{n}\rangle \mid a^{m}{}_{0}!\langle 1984\rangle$$

1 unit consumed $\hookleftarrow$

Lock freedom

Condition

$$\frac{v\text{'s tickets} > 0 \text{ (and 1 unit is consumed)}}{u!\langle v\rangle \text{ is well typed}}$$

Example

$c : \{?, !\}^{\omega}[?[\texttt{int}]^{m}{}_{n}] \vdash c!\langle a^{m}{}_{n+1}\rangle \mid *c?(x^{m}{}_{n}).c!\langle x^{m}{}_{n}\rangle \mid a^{m}{}_{0}!\langle 1984\rangle$

→ any channel sent on c must have $n+1$ tickets

1 unit consumed ←

a is sent on c infinitely many times and so it would ideally need infinitely many tickets

Recursive processes

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle$$
$$\texttt{else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

Recursive processes

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle$$
$$\texttt{else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

Recursive processes

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

the input on a block the output on y, so the level of a should be lower than the level of y

Recursive processes

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

the input on a block the output on y, so the level of a should be lower than the level of y

a and y are used in the same position and should have the same level

Recursive processes

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

Recursive processes

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

Recursive processes

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

the input on y block the output on a, so the level of y should be lower than the level of a

Recursive processes

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

the input on y block the output on a, so the level of y should be lower than the level of a

a and y are used in the same position and should have the same level

Spot the differences

$c?(x\ ).y\ !\langle x\ \rangle$

- y free
- x's level $\succ 7$
- c monomorphic

$c?(x\ , y\ ).y\ !\langle x\ \rangle$

- y bound
- x's level $\succ$ y's level
- c **polymorphic**

Spot the differences

$c?(x\ ).y\ !\langle x\ \rangle$

- y free
- x's level $\succ 7$
- c monomorphic

$c?(x\ , y\ ).y\ !\langle x\ \rangle$

- y bound
- x's level $\succ$ y's level
- c **polymorphic**

Spot the differences

$c?(x^8).y^7!\langle x^8\rangle$

- y free
- x's level $\succ 7$
- c monomorphic

$c?(x^8, y^7).y^7!\langle x^8\rangle$

- y bound
- x's level $\succ$ y's level
- c **polymorphic**

Spot the differences

$c?(x^8).y^7!\langle x^8\rangle$

- y free
- x's level $\succ 7$
- c monomorphic

$c?(x^8, y^7).y^7!\langle x^8\rangle$

- y bound
- x's level $\succ$ y's level
- c **polymorphic**

Spot the differences

$c?(x^8).y^7!\langle x^8\rangle$

- y free
- x's level $\succ 7$
- c monomorphic

$c?(x^8, y^7).y^7!\langle x^8\rangle$

- y bound
- x's level $\succ$ y's level
- c **polymorphic**

When is a channel polymorphic?

- input on c has free linear channels in continuation
  $\Rightarrow$ c monomorphic
- input on c has no free linear channels in continuation
  $\Rightarrow$ c polymorphic

When is a channel polymorphic?

- input on c has free linear channels in continuation
  $\Rightarrow$ c monomorphic
- input on c has no free linear channels in continuation
  $\Rightarrow$ c polymorphic

$$\frac{\Gamma, x : t \vdash P \qquad \mathsf{un}(\Gamma)}{\Gamma \vdash *u?(x).P}$$

When is a channel polymorphic?

- input on c has free linear channels in continuation
  $\Rightarrow$ c monomorphic
- input on c has no free linear channels in continuation
  $\Rightarrow$ c polymorphic

$$\frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma \vdash *u?(x).P}$$

Fact

Every replicated channel is polymorphic in the linear π-calculus

Back to *fact* and *stream*

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

Back to *fact* and *stream*

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

Back to *fact* and *stream*

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

Back to *fact* and *stream*

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

Back to *fact* and *stream*

$$*fact?(x, y^{0}).\texttt{if } x = 0 \texttt{ then } y^{0}!\langle 1\rangle \texttt{ else } (\nu a^{-1})(fact!\langle x-1, a^{-1}\rangle \mid a^{-1}?(z).y^{0}!\langle x \times z\rangle)$$

$$*stream?(x, y^{0}).(\nu a^{1})(y^{0}!\langle x, a^{1}\rangle \mid stream!\langle x+1, a^{1}\rangle)$$

these processes can be declared well typed since *fact* and *stream* are replicated channels, i.e. polymorphic channels

Syntax

$$u ::= a, b, \ldots, x, y, \ldots$$

names channels variables

Syntax

$$u ::= a, b, \ldots, x, y, \ldots$$

$$e ::= n \mid u \mid (e, e) \mid \texttt{inl}\ e \mid \texttt{inr}\ e$$

expressions

Syntax

$$u ::= a, b, \ldots, x, y, \ldots$$

$$e ::= n \mid u \mid (e, e) \mid \texttt{inl}\ e \mid \texttt{inr}\ e$$

$$P ::= \mathbf{0} \mid u?(x).P \mid *u?(x).P \mid u!\langle e \rangle \mid (P \mid Q) \mid (\nu a)P \mid$$
$$\texttt{let}\ x, y = e\ \texttt{in}\ P \mid \texttt{case}\ e\ \{i\ x_i \Rightarrow P_i\}_{i=\texttt{inl},\texttt{inr}}$$

processes

Syntax

$$u ::= a, b, \ldots, x, y, \ldots$$

$$e ::= n \mid u \mid (e, e) \mid \texttt{inl}\ e \mid \texttt{inr}\ e$$

$$P ::= \mathbf{0} \mid u?(x).P \mid *u?(x).P \mid u!\langle e \rangle \mid (P \mid Q) \mid (\nu a)P \mid$$
$$\texttt{let}\ x, y = e\ \texttt{in}\ P \mid \texttt{case}\ e\ \{i\ x_i \Rightarrow P_i\}_{i=\texttt{inl},\texttt{inr}}$$

$u?(x, y).P$ abbreviates $u?(z).\texttt{let}\ x, y = z\ \texttt{in}\ P$ for some fresh z

Semantics

$$a!\langle \mathrm{v} \rangle \mid a?(x).P \rightarrow P\{\mathrm{v}/x\}$$

Semantics

$$a!\langle \mathrm{v} \rangle \mid a?(x).P \rightarrow P\{\mathrm{v}/x\}$$

$$a!\langle \mathrm{v} \rangle \mid *a?(x).P \rightarrow P\{\mathrm{v}/x\} \mid *a?(x).P$$

Semantics

$$a!\langle \mathsf{v} \rangle \mid a?(x).P \to P\{\mathsf{v}/x\}$$

$$a!\langle \mathsf{v} \rangle \mid *a?(x).P \to P\{\mathsf{v}/x\} \mid *a?(x).P$$

$$\texttt{let}\ x, y = \mathsf{v}, \mathsf{w}\ \texttt{in}\ P \to P\{\mathsf{v}, \mathsf{w}/x, y\}$$

Semantics

$$a!\langle \mathsf{v} \rangle \mid a?(x).P \to P\{\mathsf{v}/x\}$$

$$a!\langle \mathsf{v} \rangle \mid *a?(x).P \to P\{\mathsf{v}/x\} \mid *a?(x).P$$

$$\texttt{let}\ x, y = \mathsf{v}, \mathsf{w}\ \texttt{in}\ P \to P\{\mathsf{v}, \mathsf{w}/x, y\}$$

$$\texttt{case}\ k\ \mathsf{v}\ \{i\ x_i \Rightarrow P_i\}_{i=\texttt{inl},\texttt{inr}} \to P_k\{\mathsf{v}/x_k\}$$

Semantics

$$a!\langle \mathsf{v} \rangle \mid a?(x).P \to P\{\mathsf{v}/x\}$$

$$a!\langle \mathsf{v} \rangle \mid *a?(x).P \to P\{\mathsf{v}/x\} \mid *a?(x).P$$

$$\texttt{let}\ x, y = \mathsf{v}, \mathsf{w}\ \texttt{in}\ P \to P\{\mathsf{v}, \mathsf{w}/x, y\}$$

$$\texttt{case}\ k\ \mathsf{v}\ \{i\ x_i \Rightarrow P_i\}_{i=\texttt{inl},\texttt{inr}} \to P_k\{\mathsf{v}/x_k\}$$

closed by reduction contexts

$$\mathcal{C} ::= [\,] \mid (\mathcal{C} \mid P) \mid (\nu a)\mathcal{C}$$

and structural congruence

Pending communications

$$\mathsf{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

input on a

Pending communications

$$\mathsf{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

$$*\mathsf{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[*a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

replicated input on a

Pending communications

$$\mathsf{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

$$*\mathsf{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[*a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

$$\mathsf{out}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[a!\langle e \rangle] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

output on a

Pending communications

$$\mathsf{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

$$*\mathsf{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[*a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

$$\mathsf{out}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[a!\langle e \rangle] \wedge a \notin \mathsf{bn}(\mathcal{C})$$

$$\mathsf{sync}(a, P) \overset{\text{def}}{\iff} (\mathsf{in}(a, P) \vee *\mathsf{in}(a, P)) \wedge \mathsf{out}(a, P)$$

immediate synchronisation on a

Pending communications

$$\mathrm{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[a?(x).Q] \wedge a \notin \mathrm{bn}(\mathcal{C})$$

$$*\mathrm{in}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[*a?(x).Q] \wedge a \notin \mathrm{bn}(\mathcal{C})$$

$$\mathrm{out}(a, P) \overset{\text{def}}{\iff} P \equiv \mathcal{C}[a!\langle e \rangle] \wedge a \notin \mathrm{bn}(\mathcal{C})$$

$$\mathrm{sync}(a, P) \overset{\text{def}}{\iff} (\mathrm{in}(a, P) \vee *\mathrm{in}(a, P)) \wedge \mathrm{out}(a, P)$$

$$\mathrm{wait}(a, P) \overset{\text{def}}{\iff} (\mathrm{in}(a, P) \vee \mathrm{out}(a, P)) \wedge \neg\mathrm{sync}(a, P)$$

communication but not

immediate synchronisation on a

Pending communications

$$\begin{aligned}
\mathsf{in}(a, P) &\overset{\text{def}}{\iff} P \equiv \mathcal{C}[a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C}) \\
*\mathsf{in}(a, P) &\overset{\text{def}}{\iff} P \equiv \mathcal{C}[*a?(x).Q] \wedge a \notin \mathsf{bn}(\mathcal{C}) \\
\mathsf{out}(a, P) &\overset{\text{def}}{\iff} P \equiv \mathcal{C}[a!\langle e \rangle] \wedge a \notin \mathsf{bn}(\mathcal{C}) \\
\mathsf{sync}(a, P) &\overset{\text{def}}{\iff} (\mathsf{in}(a, P) \vee *\mathsf{in}(a, P)) \wedge \mathsf{out}(a, P) \\
\mathsf{wait}(a, P) &\overset{\text{def}}{\iff} (\mathsf{in}(a, P) \vee \mathsf{out}(a, P)) \wedge \neg\mathsf{sync}(a, P)
\end{aligned}$$

for deadlock freedom irreducible processes cannot contain pending communications, but for replicated inputs (they play the role of persistently available servers)

Pending communications

$$\begin{aligned}
\mathrm{in}(a, P) &\overset{\text{def}}{\iff} P \equiv \mathcal{C}[a?(x).Q] \wedge a \notin \mathrm{bn}(\mathcal{C}) \\
*\mathrm{in}(a, P) &\overset{\text{def}}{\iff} P \equiv \mathcal{C}[*a?(x).Q] \wedge a \notin \mathrm{bn}(\mathcal{C}) \\
\mathrm{out}(a, P) &\overset{\text{def}}{\iff} P \equiv \mathcal{C}[a!\langle e \rangle] \wedge a \notin \mathrm{bn}(\mathcal{C}) \\
\mathrm{sync}(a, P) &\overset{\text{def}}{\iff} (\mathrm{in}(a, P) \vee *\mathrm{in}(a, P)) \wedge \mathrm{out}(a, P) \\
\mathrm{wait}(a, P) &\overset{\text{def}}{\iff} (\mathrm{in}(a, P) \vee \mathrm{out}(a, P)) \wedge \neg\mathrm{sync}(a, P)
\end{aligned}$$

for deadlock freedom irreducible processes cannot contain pending communications, but for replicated inputs (they play the role of persistently available servers)

for lock freedom

- all non-replicated inputs should be able to receive their messages
- all outputs should be able to deliver their messages

Deadlock and lock freedom

Definition (deadlock freedom)

P is deadlock free if, for every Q such that $P \to^* (\nu \tilde{a})Q \not\to$, we have $\neg\mathrm{wait}(a, Q)$ for every a

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Definition (lock freedom)

P is lock free if, for every Q such that $P \to^* (\nu \tilde{a})Q$ and $\mathrm{wait}(a, Q)$, there exists R such that $Q \to^* R$ and $\mathrm{sync}(a, R)$

- Kobayashi, Sangiorgi, **A Hybrid Type System for Lock-Freedom of Mobile Processes**, TOPLAS 2010

$(\nu a)a!\langle 1984\rangle$ and $(\nu a)a?(x).P$ are deadlocks ✖

$(\nu a)*a?(x).P$ is lock free ✔

Deadlock and lock freedom

Definition (deadlock freedom)

P is deadlock free if, for every Q such that $P \to^* (\nu \tilde{a})Q \not\to$, we have $\neg\mathrm{wait}(a, Q)$ for every a

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Definition (lock freedom)

P is lock free if, for every Q such that $P \to^* (\nu \tilde{a})Q$ and $\mathrm{wait}(a, Q)$, there exists R such that $Q \to^* R$ and $\mathrm{sync}(a, R)$

- Kobayashi, Sangiorgi, **A Hybrid Type System for Lock-Freedom of Mobile Processes**, TOPLAS 2010

$(\nu a)a!\langle 1984\rangle$ and $(\nu a)a?(x).P$ are deadlocks ✖

$(\nu a)*a?(x).P$ is lock free ✔

Deadlock and lock freedom

Definition (deadlock freedom)

P is deadlock free if, for every Q such that $P \to^* (\nu \tilde{a})Q \not\to$, we have $\neg\mathrm{wait}(a, Q)$ for every a

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Definition (lock freedom)

P is lock free if, for every Q such that $P \to^* (\nu \tilde{a})Q$ and $\mathrm{wait}(a, Q)$, there exists R such that $Q \to^* R$ and $\mathrm{sync}(a, R)$

- Kobayashi, Sangiorgi, **A Hybrid Type System for Lock-Freedom of Mobile Processes**, TOPLAS 2010

$(\nu a)a!\langle 1984\rangle$ and $(\nu a)a?(x).P$ are deadlocks ✖

$(\nu a)*a?(x).P$ is lock free ✔

Deadlock and lock freedom

Definition (deadlock freedom)

P is deadlock free if, for every Q such that $P \to^* (\nu \tilde{a})Q \not\to$, we have $\neg\mathrm{wait}(a, Q)$ for every a

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Definition (lock freedom)

P is lock free if, for every Q such that $P \to^* (\nu \tilde{a})Q$ and $\mathrm{wait}(a, Q)$, there exists R such that $Q \to^* R$ and $\mathrm{sync}(a, R)$

- Kobayashi, Sangiorgi, **A Hybrid Type System for Lock-Freedom of Mobile Processes**, TOPLAS 2010

$(\nu a)a!\langle 1984\rangle$ and $(\nu a)a?(x).P$ are deadlocks ✖

$(\nu a)*a?(x).P$ is lock free ✔

Deadlock and lock freedom

Definition (deadlock freedom)

P is deadlock free if, for every Q such that $P \rightarrow^* (\nu \tilde{a})Q \not\rightarrow$, we have $\neg\mathrm{wait}(a, Q)$ for every a

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Definition (lock freedom)

P is lock free if, for every Q such that $P \rightarrow^* (\nu \tilde{a})Q$ and $\mathrm{wait}(a, Q)$, there exists R such that $Q \rightarrow^* R$ and $\mathrm{sync}(a, R)$

- Kobayashi, Sangiorgi, **A Hybrid Type System for Lock-Freedom of Mobile Processes**, TOPLAS 2010

$(\nu a)a!\langle 1984\rangle$ and $(\nu a)a?(x).P$ are deadlocks ✖

$(\nu a)*a?(x).P$ is lock free ✔

Deadlock and lock freedom

Definition (deadlock freedom)

P is deadlock free if, for every Q such that $P \to^* (\nu \tilde{a})Q \not\to$, we have $\neg\mathrm{wait}(a, Q)$ for every a

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Definition (lock freedom)

P is lock free if, for every Q such that $P \to^* (\nu \tilde{a})Q$ and $\mathrm{wait}(a, Q)$, there exists R such that $Q \to^* R$ and $\mathrm{sync}(a, R)$

- Kobayashi, Sangiorgi, **A Hybrid Type System for Lock-Freedom of Mobile Processes**, TOPLAS 2010

$(\nu a)a!\langle 1984\rangle$ and $(\nu a)a?(x).P$ are deadlocks ✖

$(\nu a)*a?(x).P$ is lock free ✔

Types and type environments

$$t ::=$$

Types and type environments

$$t ::= \texttt{int}$$

integers

Types and type environments

$$t ::= \texttt{int} \mid \alpha$$

type variable

Types and type environments

$$t ::= \texttt{int} \mid \alpha \mid t \times s$$

pairs inhabited by values (v, w)

Types and type environments

$$t ::= \texttt{int} \mid \alpha \mid t \times s \mid t \oplus s$$

disjoint sum of t and s inhabited by values:

(`inl` v) when v has type t or
(`inr` w) when w has type s

Types and type environments

$$t ::= \texttt{int} \mid \alpha \mid t \times s \mid t \oplus s \mid p[t]^{m}_{n}$$

linear channel with polarity p, message type t, level m and tickets n

Types and type environments

$$t ::= \texttt{int} \mid \alpha \mid t \times s \mid t \oplus s \mid p[t]^m_n \mid p[t]$$

unlimited channel with polarity p and message type t

Types and type environments

$$t ::= \texttt{int} \mid \alpha \mid t \times s \mid t \oplus s \mid p[t]_n^m \mid p[t] \mid \mu\alpha.t$$

recursive type

Types and type environments

$$t ::= \texttt{int} \mid \alpha \mid t \times s \mid t \oplus s \mid p[t]_n^m \mid p[t] \mid \mu\alpha.t$$

type environments Γ are finite maps from names to types

$$u_1 : t_1, \ldots, u_n : t_n$$

Typing rules for expressions

Typing rules for expressions

$$\frac{\text{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \quad [\text{t-const}]$$

$\text{un}(\Gamma)$ if all the types in the range of Γ are unlimited

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

level of types

$$|t| \stackrel{\text{def}}{=} \begin{cases}$$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

level of types

$$|t| \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } t = p[s] \text{ and } ? \in p \end{cases}$$

unlimited channels with input polarity have the lowest level $\bot$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

level of types

$$|t| \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } t = p[s] \text{ and } ? \in p \\ m & \text{if } t = p[s]^m_n \text{ and } p \neq \emptyset \end{cases}$$

linear channels with pending operations must be used according to their level

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \mathtt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

level of types

$$|t| \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } t = p[s] \text{ and } ? \in p \\ m & \text{if } t = p[s]^m_n \text{ and } p \neq \emptyset \\ \min\{|t_1|, |t_2|\} & \text{if } t = t_1 \times t_2 \text{ or } t = t_1 \oplus t_2 \end{cases}$$

the level of compound types is the minimum of the levels of the component types

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \mathtt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

level of types

$$|t| \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } t = p[s] \text{ and } ? \in p \\ m & \text{if } t = p[s]^m_n \text{ and } p \neq \emptyset \\ \min\{|t_1|, |t_2|\} & \text{if } t = t_1 \times t_2 \text{ or } t = t_1 \oplus t_2 \\ \top & \text{otherwise} \end{cases}$$

numbers, unlimited channels with only output polarity, and linear channels with empty polarity have the highest level $\top$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \mathtt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

level of types

$$|t| \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } t = p[s] \text{ and } ? \in p \\ m & \text{if } t = p[s]^m_n \text{ and } p \neq \emptyset \\ \min\{|t_1|, |t_2|\} & \text{if } t = t_1 \times t_2 \text{ or } t = t_1 \oplus t_2 \\ \top & \text{otherwise} \end{cases}$$

$$|\,!\,[t]^3_1 \times ?[s]^2_4| = \min\{|\,!\,[t]^3_1|, |?[s]^2_4|\} = \min\{3, 2\} = 2$$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of types

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of types

$$t + t = t \qquad \text{if } |t| = \top$$

types with $\top$ level compose with themselves without restrictions

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of types

$$t + t = t \qquad \text{if } |t| = \top$$
$$p[t] + q[t] = (p \cup q)[t]$$

the composition of two unlimited channel types has the union of their polarities

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of types

$$\begin{aligned} t + t &= t && \text{if } |t| = \top \\ p[t] + q[t] &= (p \cup q)[t] \\ p[s]^m_h + q[s]^m_k &= (p \cup q)[s]^m_{h+k} && \text{if } p \cap q = \emptyset \end{aligned}$$

two linear channel types can be composed only if they have the same level and disjoint polarities, and the composition has the union of their polarities and the sum of their tickets

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of types

$$\begin{aligned} t + t &= t && \text{if } |t| = \top \\ p[t] + q[t] &= (p \cup q)[t] \\ p[s]^m_h + q[s]^m_k &= (p \cup q)[s]^m_{h+k} && \text{if } p \cap q = \emptyset \end{aligned}$$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of type environments

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of type environments

$$\Gamma + \Gamma' \stackrel{\text{def}}{=} \Gamma, \Gamma' \qquad \text{if } \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \emptyset$$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of type environments

$$\Gamma + \Gamma' \stackrel{\text{def}}{=} \Gamma, \Gamma' \qquad \text{if } \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \emptyset$$

$$(\Gamma, u : t) + (\Gamma', u : s) \stackrel{\text{def}}{=} (\Gamma + \Gamma'), u : t + s$$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t}$$

composition of type environments

$$\Gamma + \Gamma' \stackrel{\text{def}}{=} \Gamma, \Gamma' \qquad \text{if } \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \emptyset$$
$$(\Gamma, u : t) + (\Gamma', u : s) \stackrel{\text{def}}{=} (\Gamma + \Gamma'), u : t + s$$

$\Gamma + \Gamma'$ is undefined if there is $u \in \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma')$ such that $\Gamma(u) + \Gamma'(u)$ is undefined

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t} \qquad \text{[t-pair]}\ \frac{\Gamma \vdash e : t \qquad \Gamma' \vdash e' : s}{\Gamma + \Gamma' \vdash (e, e') : t \times s}$$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t} \qquad \text{[t-pair]}\ \frac{\Gamma \vdash e : t \qquad \Gamma' \vdash e' : s}{\Gamma + \Gamma' \vdash (e, e') : t \times s}$$

$$\text{[t-inl]}\ \frac{\Gamma \vdash e : t}{\Gamma \vdash \texttt{inl}\ e : t \oplus s}$$

Typing rules for expressions

$$\text{[t-const]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash n : \texttt{int}} \qquad \text{[t-name]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma, u : t \vdash u : t} \qquad \text{[t-pair]}\ \frac{\Gamma \vdash e : t \qquad \Gamma' \vdash e' : s}{\Gamma + \Gamma' \vdash (e, e') : t \times s}$$

$$\text{[t-inl]}\ \frac{\Gamma \vdash e : t}{\Gamma \vdash \texttt{inl}\ e : t \oplus s} \qquad \text{[t-inr]}\ \frac{\Gamma \vdash e : s}{\Gamma \vdash \texttt{inr}\ e : t \oplus s}$$

Typing rules for processes I

Typing rules for processes I

$$\frac{\text{un}(\Gamma)}{\Gamma \vdash \mathbf{0}} \quad [\text{t-idle}]$$

Typing rules for processes I

$$\text{[t-idle]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash \mathbf{0}} \qquad \text{[t-par]}\ \frac{\Gamma_1 \vdash P_1 \quad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2}$$

Typing rules for processes I

$$\text{[t-idle]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash \mathbf{0}} \qquad \text{[t-par]}\ \frac{\Gamma_1 \vdash P_1 \quad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2}$$

$$\text{[t-let]}\ \frac{\Gamma \vdash e : t \times s \qquad \Gamma', x : t, y : s \vdash P}{\Gamma + \Gamma' \vdash \texttt{let}\ x, y = e\ \texttt{in}\ P}$$

Typing rules for processes I

$$\text{[t-idle]}\ \frac{\mathrm{un}(\Gamma)}{\Gamma \vdash \mathbf{0}} \qquad \text{[t-par]}\ \frac{\Gamma_1 \vdash P_1 \quad \Gamma_2 \vdash P_2}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2}$$

$$\text{[t-let]}\ \frac{\Gamma \vdash e : t \times s \qquad \Gamma', x : t, y : s \vdash P}{\Gamma + \Gamma' \vdash \texttt{let}\ x, y = e\ \texttt{in}\ P}$$

$$\text{[t-case]}\ \frac{\Gamma \vdash e : t \oplus s \qquad \Gamma', x : t \vdash P \qquad \Gamma', y : s \vdash Q}{\Gamma + \Gamma' \vdash \texttt{case}\ e\ \{\texttt{inl}\ x \Rightarrow P, \texttt{inr}\ y \Rightarrow Q\}}$$

Typing rules for processes II

[t-newL]

$$\frac{\Gamma, a : p[t]^{m}_{n} \vdash P \qquad p = \{?, !\} \text{ or } p = \emptyset}{\Gamma \vdash (\nu a)P}$$

[t-newU]

$$\frac{\Gamma, a : p[t] \vdash P \qquad p = \{?, !\} \text{ or } p = \emptyset}{\Gamma \vdash (\nu a)P}$$

Typing rules for processes III

[t-in*]

$$\frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P}$$

Typing rules for processes III

$$\text{[t-in*]}\quad \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P}$$

- u must be an unlimited channel with input polarity

Typing rules for processes III

$$\text{[t-in*]}\quad \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P}$$

- u must be an unlimited channel with input polarity
- the residual environment Γ must be unlimited, because the continuation P can be spawned an arbitrary number of times

Typing rules for processes III

$$\text{[t-in*]}\quad \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P}$$

- u must be an unlimited channel with input polarity
- the residual environment Γ must be unlimited, because the continuation P can be spawned an arbitrary number of times
- $?[t] + p[t] = \{?\} \cup p[t]$ allows $u \in \mathrm{dom}(\Gamma)$

Typing rules for processes III

[t-in*]

$$\frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P}$$

shift

Typing rules for processes III

[t-in*]

$$\frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P}$$

shift

$$\$_k^h\, t \stackrel{\text{def}}{=} \begin{cases} p[s]_{n+k}^{m+h} & \text{if } t = p[s]_n^m \text{ and } p \neq \emptyset \\ (\$_k^h\, t_1) \times (\$_k^h\, t_2) & \text{if } t = t_1 \times t_2 \\ (\$_k^h\, t_1) \oplus (\$_k^h\, t_2) & \text{if } t = t_1 \oplus t_2 \\ t & \text{otherwise} \end{cases}$$

$\$_k^h\, t$ is t with all levels/tickets in the topmost linear channel types transposed by h and k, respectively

Typing rules for processes III

[t-in*]

$$\frac{\Gamma, x : t \vdash P \qquad \mathsf{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P}$$

shift

$$\$_k^h\, t \stackrel{\text{def}}{=} \begin{cases} p[s]_{n+k}^{m+h} & \text{if } t = p[s]_n^m \text{ and } p \neq \emptyset \\ (\$_k^h\, t_1) \times (\$_k^h\, t_2) & \text{if } t = t_1 \times t_2 \\ (\$_k^h\, t_1) \oplus (\$_k^h\, t_2) & \text{if } t = t_1 \oplus t_2 \\ t & \text{otherwise} \end{cases}$$

$$\$_2^1\,(\mathtt{int} \times ?[\mathtt{int}]_0^3) = (\$_2^1\,\mathtt{int}) \times (\$_1^1\,?[\mathtt{int}]_0^3) = \mathtt{int} \times ?[\mathtt{int}]_2^4$$

Typing rules for processes III

$$\text{[t-in*]}\quad \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P}$$

$$\text{[t-out*]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad \bot < |t|}{\Gamma + u : ![t] \vdash u!\langle e \rangle}$$

Typing rules for processes III

$$\text{[t-in*]}\ \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P} \qquad \text{[t-out*]}\ \frac{\Gamma \vdash e : \$_1^m\, t \qquad \bot < |t|}{\Gamma + u : ![t] \vdash u!\langle e \rangle}$$

- an unlimited channel with input polarity cannot be communicated

Typing rules for processes III

$$\text{[t-in*]}\quad \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P} \qquad \text{[t-out*]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad \bot < |t|}{\Gamma + u : ![t] \vdash u!\langle e \rangle}$$

- an unlimited channel with input polarity cannot be communicated
- the level of the channel can be shifted by an arbitrary amount m

Typing rules for processes III

$$\text{[t-in*]}\quad \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P} \qquad \text{[t-out*]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad \bot < |t|}{\Gamma + u : ![t] \vdash u!\langle e \rangle}$$

- an unlimited channel with input polarity cannot be communicated
- the level of the channel can be shifted by an arbitrary amount m
- one ticket is consumed

Typing rules for processes III

$$\text{[t-in*]}\quad \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P} \qquad \text{[t-out*]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad \bot < |t|}{\Gamma + u : ![t] \vdash u!\langle e \rangle}$$

$$a : \mu\alpha.![\alpha] \vdash a!\langle a \rangle$$

Typing rules for processes III

$$\text{[t-in*]}\quad \frac{\Gamma, x : t \vdash P \qquad \mathrm{un}(\Gamma)}{\Gamma + u : ?[t] \vdash *u?(x).P} \qquad \text{[t-out*]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad \bot < |t|}{\Gamma + u : ![t] \vdash u!\langle e \rangle}$$

shift implements polymorphism: an unlimited channel of type $![t]$ accepts messages of type $\$_-^n\, t$ for any n

Typing rules for processes IV

[t-in]

$$\frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}$$

Typing rules for processes IV

$$\frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}\ [\text{t-in}]$$

$$|\Gamma| = \min\{|\Gamma(u)| \mid u \in \mathrm{dom}(\Gamma)\}$$

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}$$

- the type of x is not just t, but t with level shifted by m

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}$$

- the type of x is not just t, but t with level shifted by m
- the input on u does not block operations on other channels with equal or lower level

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}$$

$a : ?[\texttt{int}]_0^1, b : ![\texttt{int}]_0^0 \not\vdash a?(x).b!\langle x\rangle$

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}$$

$a : \{?, !\}[\texttt{int}]_0^h \not\vdash a?(x).a!\langle x\rangle$

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}$$

$a : ?[\texttt{int}]_0^h, c : ?[\texttt{int}] \not\vdash a?(x).{*}c?(y)$

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}$$

$$\text{[t-out]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e \rangle}$$

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P} \qquad \text{[t-out]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e \rangle}$$

- the type of e is not just t, but t shifted by m

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P} \qquad \text{[t-out]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e \rangle}$$

- the type of e is not just t, but t shifted by m
- the tickets are shifted by 1

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P} \qquad \text{[t-out]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e \rangle}$$

- the type of e is not just t, but t shifted by m
- the tickets are shifted by 1
- the level of the message is greater than that of the channel on which it travels

Typing rules for processes IV

$$\frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}\ [\text{t-in}] \qquad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e\rangle}\ [\text{t-out}]$$

$a : ![?[\texttt{int}]_0^1]_0^2, b : ?[\texttt{int}]_1^3 \vdash a!\langle b\rangle\ (?[\texttt{int}]_1^3 = \$_1^2\, ?[\texttt{int}]_0^1)$

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P} \qquad \text{[t-out]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e\rangle}$$

$a : ?[![\texttt{int}]_0^2]_0^0, b : ![![\texttt{int}]_0^1]_0^1 \not\vdash a?(x).b!\langle x\rangle$

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}$$

$$\text{[t-out]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e \rangle}$$

$a : \{?, !\}[\mu\alpha.?[\alpha]_1^0]_1^1 \not\vdash a!\langle a \rangle$

Typing rules for processes IV

$$\frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P}\ [\text{t-in}] \qquad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e \rangle}\ [\text{t-out}]$$

- a linear channel of type $![t]_-^h$ accepts messages of type $\$_-^h\, t$

Typing rules for processes IV

$$\text{[t-in]}\quad \frac{\Gamma, x : \$_0^m\, t \vdash P \qquad m < |\Gamma|}{\Gamma + u : ?[t]_n^m \vdash u?(x).P} \qquad \text{[t-out]}\quad \frac{\Gamma \vdash e : \$_1^m\, t \qquad 0 < |t|}{\Gamma + u : ![t]_n^m \vdash u!\langle e \rangle}$$

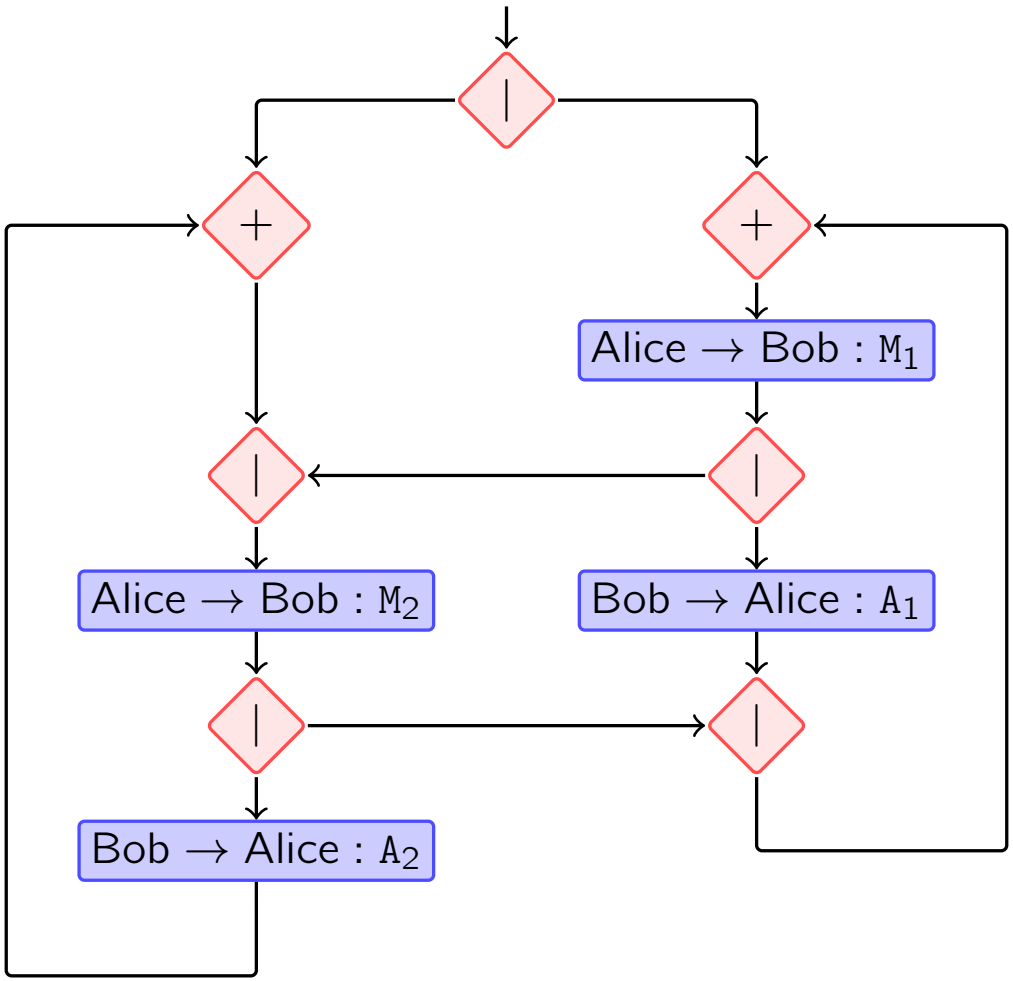
- a linear channel of type $![t]_-^h$ accepts messages of type $\$_-^h\, t$
- message types are "relative" to the level of the channels on which they travel

Soundness

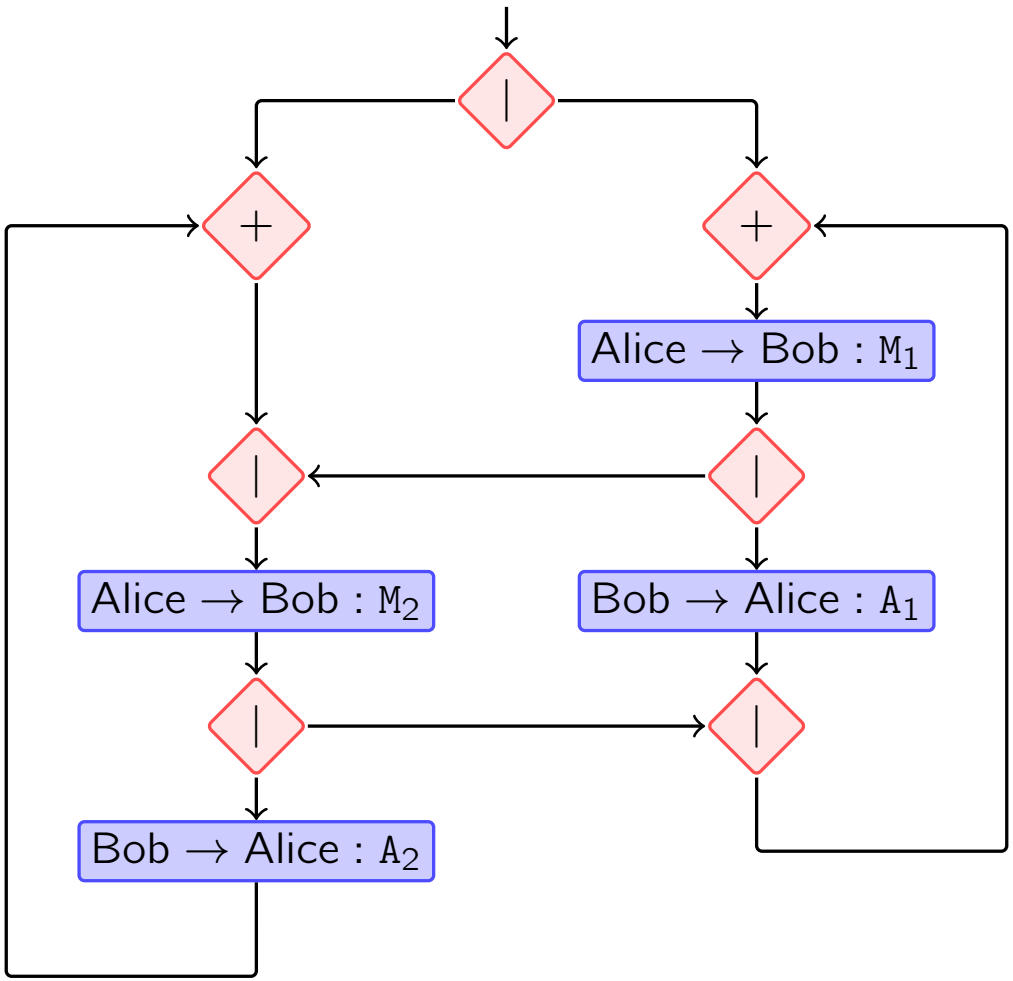
Theorem

- *If $\vdash P$ without taking into account the conditions on tickets, then P is deadlock free*
- *If $\vdash P$, then P is lock free*

Lock-free alternating bit protocol

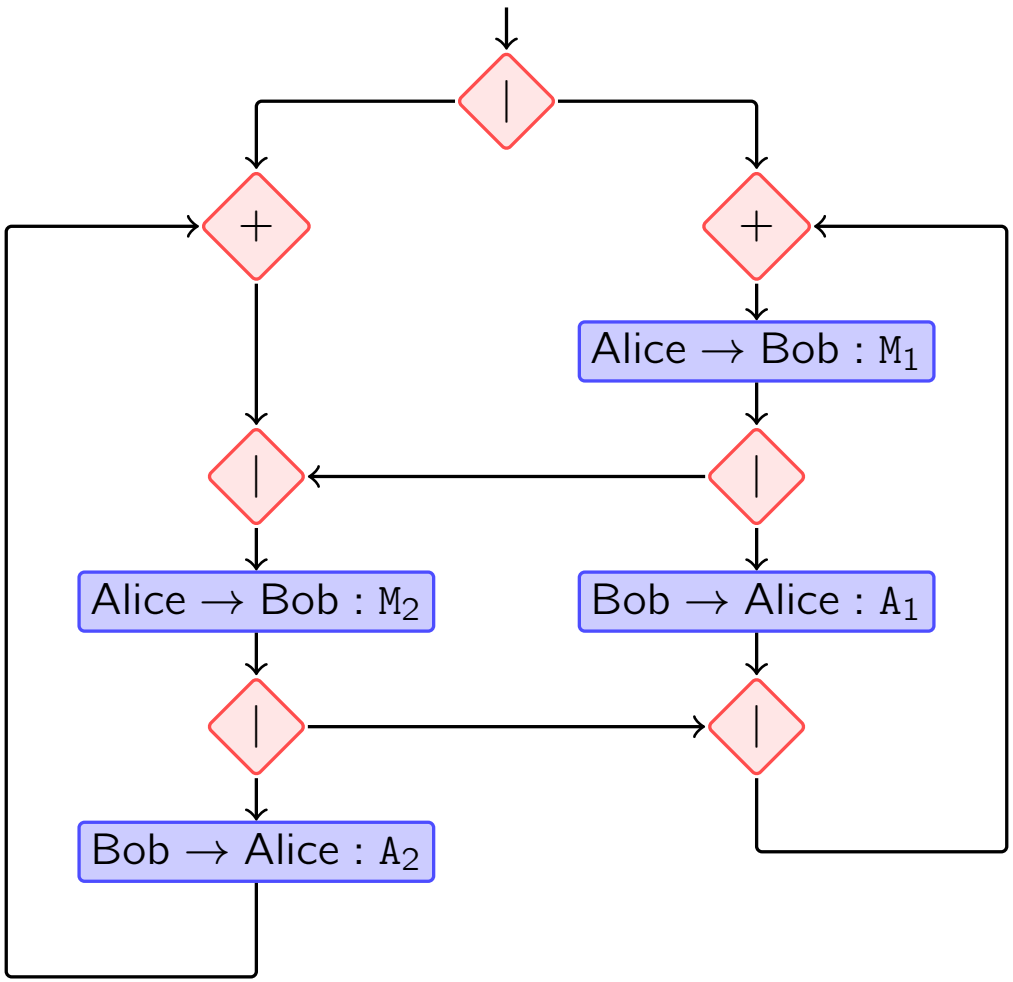


Lock-free alternating bit protocol



merging points (+ nodes) forks (| nodes with two outgoing arcs) joins (| nodes with two incoming arcs)
Alice sends alternated messages $\mathtt{M}_1$ and $\mathtt{M}_2$ to Bob, and Bob answers with corresponding acknowledgments $\mathtt{A}_1$ and $\mathtt{A}_2$
Alice waits for both $\mathtt{A}_1$ and $\mathtt{A}_2$ before sending the next message

Lock-free alternating bit protocol



- Deniélou, Yoshida, **Multiparty Session Types Meet Communicating Automata**, ESOP 2012

Lock-free alternating bit protocol

$$\mathrm{Bob} \stackrel{\text{def}}{=} *bob?(x_0^0, y_0^2).(\nu a_3^3 b_3^5)\big(x_0^0?(\bar{x}_0^1).(\bar{x}_0^1!\langle a_2^3\rangle \mid y_0^2?(\bar{y}_0^3).(\bar{y}_0^3!\langle b_2^5\rangle \mid bob!\langle a_1^3, b_1^5\rangle))\big)$$

$$\mathrm{Alice}_1 \stackrel{\text{def}}{=} *alice_1?(x_0^0, z_0^1).(\nu a_1^1 c_1^2)\big(x_0^0!\langle a_1^1\rangle \mid z_0^1!\langle c_1^2\rangle \mid a_0^1?(\bar{x}_1^3).c_0^2?(\bar{z}_1^4).alice_1!\langle \bar{x}_1^3, \bar{z}_1^4\rangle\big)$$

$$\mathrm{Alice}_2 \stackrel{\text{def}}{=} *alice_2?(y_0^2, z_0^1).(\nu b_1^3 c_3^4)\big(z_0^1?(\bar{z}_0^2).(y_0^2!\langle b_1^3\rangle \mid \bar{z}_0^2!\langle c_2^4\rangle \mid b_0^3?(\bar{y}_1^5).alice_2!\langle \bar{y}_1^5, c_1^4\rangle)\big)$$

$$(\nu ab)(\mathrm{Bob} \mid \mathrm{Alice}_1 \mid \mathrm{Alice}_2 \mid bob!\langle a, b\rangle \mid (\nu c)(alice_1!\langle a, c\rangle \mid alice_2!\langle b, c\rangle))$$

Global types

global types guarantee lock freedom by design within sessions

Global types

global types guarantee lock freedom by design within sessions

$$\varepsilon ::= \mathtt{p} \xrightarrow{\ell} \mathtt{q}$$

$\mathtt{p} \xrightarrow{\ell} \mathtt{q}$ represents the communication of a message from participant $\mathtt{p}$ (the sender) to participant $\mathtt{q}$ (the receiver)

Global types

global types guarantee lock freedom by design within sessions

$$\varepsilon ::= \mathtt{p} \xrightarrow{\ell} \mathtt{q} \qquad G ::=$$

$\mathtt{p} \xrightarrow{\ell} \mathtt{q}$ represents the communication of a message from participant $\mathtt{p}$ (the sender) to participant $\mathtt{q}$ (the receiver)

Global types

global types guarantee lock freedom by design within sessions

$$\varepsilon ::= \mathtt{p} \xrightarrow{\ell} \mathtt{q} \qquad G ::= \varepsilon.G$$

$\mathtt{p} \xrightarrow{\ell} \mathtt{q}$ represents the communication of a message from participant $\mathtt{p}$ (the sender) to participant $\mathtt{q}$ (the receiver)
ε may occur before all the other events in G

Global types

global types guarantee lock freedom by design within sessions

$$\varepsilon ::= \mathtt{p} \xrightarrow{\ell} \mathtt{q} \qquad G ::= \varepsilon.G \mid \varepsilon.\{G, G\}$$

$\mathtt{p} \xrightarrow{\ell} \mathtt{q}$ represents the communication of a message from participant $\mathtt{p}$ (the sender) to participant $\mathtt{q}$ (the receiver)
ε may occur before all the other events in G
the sender decides between G_1 and G_2 in $\varepsilon.\{G_1, G_2\}$

Global types

global types guarantee lock freedom by design within sessions

$$\varepsilon ::= \mathtt{p} \xrightarrow{\ell} \mathtt{q} \qquad G ::= \varepsilon.G \mid \varepsilon.\{G, G\} \mid \alpha \mid \mu\alpha.G$$

$\mathtt{p} \xrightarrow{\ell} \mathtt{q}$ represents the communication of a message from participant $\mathtt{p}$ (the sender) to participant $\mathtt{q}$ (the receiver)
ε may occur before all the other events in G
the sender decides between G_1 and G_2 in $\varepsilon.\{G_1, G_2\}$
recursive protocols `end` is short for $\mu\alpha.\alpha$

Global types

global types guarantee lock freedom by design within sessions

$$\varepsilon ::= \mathtt{p} \xrightarrow{\ell} \mathtt{q} \qquad G ::= \varepsilon.G \mid \varepsilon.\{G, G\} \mid \alpha \mid \mu\alpha.G$$

$\mathtt{p} \xrightarrow{\ell} \mathtt{q}$ represents the communication of a message from participant $\mathtt{p}$ (the sender) to participant $\mathtt{q}$ (the receiver)
ε may occur before all the other events in G
the sender decides between G_1 and G_2 in $\varepsilon.\{G_1, G_2\}$
recursive protocols $\qquad$ $\mathtt{end}$ is short for $\mu\alpha.\alpha$

Unsensible protocol

$$G_a \stackrel{\text{def}}{=} \mathtt{A} \to \mathtt{B}.\{\mathtt{C} \to \mathtt{D}.\mathtt{end}, \mathtt{D} \to \mathtt{C}.\mathtt{end}\}$$

Global types

global types guarantee lock freedom by design within sessions

$$\varepsilon ::= \mathtt{p} \xrightarrow{\ell} \mathtt{q} \qquad G ::= \varepsilon.G \mid \varepsilon.\{G, G\} \mid \alpha \mid \mu\alpha.G$$

$\mathtt{p} \xrightarrow{\ell} \mathtt{q}$ represents the communication of a message from participant $\mathtt{p}$ (the sender) to participant $\mathtt{q}$ (the receiver)
ε may occur before all the other events in G
the sender decides between G_1 and G_2 in $\varepsilon.\{G_1, G_2\}$
recursive protocols $\qquad$ $\mathtt{end}$ is short for $\mu\alpha.\alpha$

Unsensible protocol

$$G_a \stackrel{\text{def}}{=} \mathtt{A} \to \mathtt{B}.\{\mathtt{C} \to \mathtt{D}.\mathtt{end}, \mathtt{D} \to \mathtt{C}.\mathtt{end}\}$$

neither $\mathtt{C}$ nor $\mathtt{D}$ are aware of the decision of $\mathtt{A}$

Realisable global types

$$G_b \stackrel{\text{def}}{=} \mathtt{A} \xrightarrow{\ell_1} \mathtt{B}. \left\{ \begin{array}{l} \mathtt{B} \xrightarrow{\ell_2} \mathtt{A.C} \xrightarrow{\ell_3} \mathtt{D.end} \\ \mathtt{C} \xrightarrow{\ell_3} \mathtt{D.B} \xrightarrow{\ell_4} \mathtt{A.end} \end{array} \right\}$$

Realisable global types

$$G_b \stackrel{\text{def}}{=} \mathtt{A} \xrightarrow{\ell_1} \mathtt{B}. \left\{ \begin{array}{l} \mathtt{B} \xrightarrow{\ell_2} \mathtt{A.C} \xrightarrow{\ell_3} \mathtt{D.end} \\ \mathtt{C} \xrightarrow{\ell_3} \mathtt{D.B} \xrightarrow{\ell_4} \mathtt{A.end} \end{array} \right\}$$

OK: `B` is aware of the choice taken by `A`, `C` and `D` are unaware of the choice taken by `A`, but behave in the same way in the two branches

Realisable global types

$$G_b \stackrel{\text{def}}{=} \mathtt{A} \xrightarrow{\ell_1} \mathtt{B}. \begin{Bmatrix} \mathtt{B} \xrightarrow{\ell_2} \mathtt{A.C} \xrightarrow{\ell_3} \mathtt{D.end} \\ \mathtt{C} \xrightarrow{\ell_3} \mathtt{D.B} \xrightarrow{\ell_4} \mathtt{A.end} \end{Bmatrix}$$

OK: B is aware of the choice taken by A, C and D are unaware of the choice taken by A, but behave in the same way in the two branches

$$G_c \stackrel{\text{def}}{=} \mathtt{A} \xrightarrow{\ell_1} \mathtt{B}. \begin{Bmatrix} \mathtt{E} \xrightarrow{\ell_2} \mathtt{F.C} \xrightarrow{\ell_3} \mathtt{D.end} \\ \mathtt{C} \xrightarrow{\ell_3} \mathtt{D.E} \xrightarrow{\ell_2} \mathtt{F.end} \end{Bmatrix}$$

Realisable global types

$$G_b \stackrel{\text{def}}{=} \mathtt{A} \xrightarrow{\ell_1} \mathtt{B}. \begin{cases} \mathtt{B} \xrightarrow{\ell_2} \mathtt{A.C} \xrightarrow{\ell_3} \mathtt{D.end} \\ \mathtt{C} \xrightarrow{\ell_3} \mathtt{D.B} \xrightarrow{\ell_4} \mathtt{A.end} \end{cases}$$

OK: B is aware of the choice taken by A, C and D are unaware of the choice taken by A, but behave in the same way in the two branches

$$G_c \stackrel{\text{def}}{=} \mathtt{A} \xrightarrow{\ell_1} \mathtt{B}. \begin{cases} \mathtt{E} \xrightarrow{\ell_2} \mathtt{F.C} \xrightarrow{\ell_3} \mathtt{D.end} \\ \mathtt{C} \xrightarrow{\ell_3} \mathtt{D.E} \xrightarrow{\ell_2} \mathtt{F.end} \end{cases}$$

NO: there is a circular precedence between the labels ℓ_2 and ℓ_3 (we cannot take different labels since C, D, E and F are unaware of the choice taken by A)

Realisable global types

$$G_b \stackrel{\text{def}}{=} \mathtt{A} \xrightarrow{\ell_1} \mathtt{B}. \begin{cases} \mathtt{B} \xrightarrow{\ell_2} \mathtt{A.C} \xrightarrow{\ell_3} \mathtt{D.end} \\ \mathtt{C} \xrightarrow{\ell_3} \mathtt{D.B} \xrightarrow{\ell_4} \mathtt{A.end} \end{cases}$$

OK: B is aware of the choice taken by A, C and D are unaware of the choice taken by A, but behave in the same way in the two branches

$$G_c \stackrel{\text{def}}{=} \mathtt{A} \xrightarrow{\ell_1} \mathtt{B}. \begin{cases} \mathtt{E} \xrightarrow{\ell_2} \mathtt{F.C} \xrightarrow{\ell_3} \mathtt{D.end} \\ \mathtt{C} \xrightarrow{\ell_3} \mathtt{D.E} \xrightarrow{\ell_2} \mathtt{F.end} \end{cases}$$

NO: there is a circular precedence between the labels ℓ_2 and ℓ_3 (we cannot take different labels since C, D, E and F are unaware of the choice taken by A)

All global types of dyadic sessions are realisable

Characteristic participants

$$G_d \stackrel{\text{def}}{=} \mu\alpha.\mathtt{A} \to \mathtt{B}.\mathtt{B} \to \mathtt{C}.\mathtt{C} \to \mathtt{A}.\alpha$$

Characteristic participants

$$G_d \stackrel{\text{def}}{=} \mu\alpha.\mathtt{A} \to \mathtt{B}.\mathtt{B} \to \mathtt{C}.\mathtt{C} \to \mathtt{A}.\alpha$$

$$\mathbf{P}(G_d, \mathtt{A}) =$$

Characteristic participants

$$G_d \stackrel{\text{def}}{=} \mu\alpha.\mathtt{A} \to \mathtt{B}.\mathtt{B} \to \mathtt{C}.\mathtt{C} \to \mathtt{A}.\alpha$$

$$\begin{aligned}&\mathbf{P}(G_d, \mathtt{A}) = \\ &\quad (\nu c_\alpha)(c_\alpha!\langle x_{\mathtt{AB}}, x_{\mathtt{AC}}\rangle \mid \\ &\qquad *c_\alpha?(x_{\mathtt{AB}}, x_{\mathtt{AC}}).\\ &\qquad\qquad c_\alpha!\langle x_{\mathtt{AB}}, x_{\mathtt{AC}}\rangle))\end{aligned}$$

Characteristic participants

$$G_d \stackrel{\text{def}}{=} \mu\alpha.\mathtt{A} \to \mathtt{B}.\mathtt{B} \to \mathtt{C}.\mathtt{C} \to \mathtt{A}.\alpha$$

$$\begin{aligned}&\mathbf{P}(G_d, \mathtt{A}) = \\ &\quad (\nu c_\alpha)(c_\alpha!\langle x_{\mathtt{AB}}, x_{\mathtt{AC}}\rangle \mid \\ &\qquad *c_\alpha?(x_{\mathtt{AB}}, x_{\mathtt{AC}}).(\nu a)(x_{\mathtt{AB}}!\langle a\rangle \mid \mathtt{let}\ x_{\mathtt{AB}} = a\ \mathtt{in} \\ &\qquad\qquad c_\alpha!\langle x_{\mathtt{AB}}, x_{\mathtt{AC}}\rangle))\end{aligned}$$

Characteristic participants

$$G_d \stackrel{\text{def}}{=} \mu\alpha.\mathtt{A} \to \mathtt{B}.\mathtt{B} \to \mathtt{C}.\mathtt{C} \to \mathtt{A}.\alpha$$

$$\begin{aligned}&\mathbf{P}(G_d, \mathtt{A}) = \\ &\quad (\nu c_\alpha)(c_\alpha!\langle x_{\mathtt{AB}}, x_{\mathtt{AC}}\rangle \mid \\ &\qquad *c_\alpha?(x_{\mathtt{AB}}, x_{\mathtt{AC}}).(\nu a)(x_{\mathtt{AB}}!\langle a\rangle \mid \mathtt{let}\ x_{\mathtt{AB}} = a\ \mathtt{in} \\ &\qquad\qquad x_{\mathtt{AC}}?(x_{\mathtt{AC}}).c_\alpha!\langle x_{\mathtt{AB}}, x_{\mathtt{AC}}\rangle)))\end{aligned}$$

Characteristic participants

$$G_d \stackrel{\text{def}}{=} \mu\alpha.\mathtt{A} \to \mathtt{B}.\mathtt{B} \to \mathtt{C}.\mathtt{C} \to \mathtt{A}.\alpha$$

$$\begin{aligned}
&\mathbf{P}(G_d, \mathtt{A}) = \\
&\quad (\nu c_\alpha)(c_\alpha!\langle x_{\mathtt{AB}}, x_{\mathtt{AC}}\rangle \mid \\
&\qquad *c_\alpha?(x_{\mathtt{AB}}, x_{\mathtt{AC}}).(\nu a)(x_{\mathtt{AB}}!\langle a\rangle \mid \mathtt{let}\ x_{\mathtt{AB}} = a\ \mathtt{in} \\
&\qquad\qquad x_{\mathtt{AC}}?(x_{\mathtt{AC}}).c_\alpha!\langle x_{\mathtt{AB}}, x_{\mathtt{AC}}\rangle)))
\end{aligned}$$

Main result

Theorem

Each characteristic participant of a realisable global type is well typed.

Tracking dependencies between **sessions**

1. associate processes with dependencies $a \prec b$

"an action of service a blocks an action of service b"

2. a process is well typed if it yields no circular dependencies

Tracking dependencies between **sessions**

1. associate processes with dependencies $a \prec b$

"an action of service a blocks an action of service b"

2. a process is well typed if it yields no circular dependencies

Tracking dependencies between **sessions**

1. associate processes with dependencies $a \prec b$

"an action of service a blocks an action of service b"

2. a process is well typed if it yields no circular dependencies

Computing service dependencies

$$a(y).b(z).y?(x).z!\langle x\rangle \qquad a \prec b$$

$$\overline{a}(y).\overline{b}(z).z?(x).y!\langle x\rangle \qquad b \prec a$$

Service names as messages

$$a(y).b(z).y?(x).z!\langle x\rangle \qquad a \prec b$$

$$\overline{c}(t).t?(x).\overline{x}(y).\overline{b}(z).z?(x).y!\langle x\rangle$$

$$c(t).t!\langle a\rangle$$

Service names as messages

$$a(y).b(z).y?(x).z!\langle x\rangle \qquad a \prec b$$

$$t?(x).\overline{x}(y).\overline{b}(z).z?(x).y!\langle x\rangle$$

$$t!\langle a\rangle$$

Service names as messages

$$a(y).b(z).y?(x).z!\langle x\rangle \qquad a \prec b$$

$$\overline{a}(y).\overline{b}(z).z?(x).y!\langle x\rangle \qquad b \prec a$$

Service names as messages

$$a(y).b(z).y?(x).z!\langle x\rangle \qquad a \prec b$$

$$\overline{a}(y).\overline{b}(z).z?(x).y!\langle x\rangle$$

Idea

- identify a class of safe services even if mutually dependent
- restrict messages to services in this class

Nested services

Definition

a is a nested service if $\lambda \prec a$ implies that λ is a nested service

		Nested?
$\overline{a}(y).\overline{a}(z).z?(x).y?(x')$	$a \prec a$	✔
$\overline{a}(y).\overline{b}(z).z?(x).y?(x')$ $\mid \overline{b}(z).\overline{a}(y).y?(x).z?(x')$	$b \prec a$ $a \prec b$	✔
$\overline{a}(y).\overline{b}(z).y?(x).z?(x')$	$y \prec b$	✖

Boundable services

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

- no parallel process can help starting the session on b

Boundable services

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

- no parallel process can help starting the session on b

Definition

a is boundable if it is never followed by free channels

- b is nested but not boundable

Interaction type system

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad a\in\mathscr{R}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \tilde{a}[\mathrm{p}](y).P \blacktriangleright \mathscr{D}\{a/y\}^{+}}\ \{\textsc{InitR}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad a\in\mathscr{N}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \tilde{a}[\mathrm{p}](y).P \blacktriangleright \mathscr{D}\setminus\!\setminus y}\ \{\textsc{InitN}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad a\in\mathscr{B} \quad \mathsf{fc}(P)\subseteq\{y\}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \tilde{a}[\mathrm{p}](y).P \blacktriangleright \mathscr{D}\setminus\!\setminus y}\ \{\textsc{InitB}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad \mathsf{fc}(P)\subseteq\{y\}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \tilde{x}[\mathrm{p}](y).P \blacktriangleright \mathscr{D}\setminus\!\setminus y}\ \{\textsc{InitV}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad e\in\mathscr{S}\Rightarrow e\in\mathscr{N}\cup\mathscr{B}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c!\langle\Pi,e\rangle.P \blacktriangleright \mathscr{D}}\ \{\textsc{Send}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c?(\mathrm{q},x).P \blacktriangleright (\mathsf{pre}(c,\mathsf{fc}(P))\cup\mathscr{D})^{+}}\ \{\textsc{Rcv}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c!\langle\langle\mathrm{p},c'\rangle\rangle.P \blacktriangleright (\{\curlywedge(c)\prec\curlywedge(c')\}\cup\mathscr{D})^{+}}\ \{\textsc{Deleg}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad \mathscr{D}\setminus\mathscr{S}\subseteq\{\curlywedge(c)\prec y\}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c?((\mathrm{q},y)).P \blacktriangleright \mathscr{D}\setminus\{y\}}\ \{\textsc{SRcv}\}$$

$$\frac{}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \mathbf{0} \blacktriangleright \emptyset}\ \{\textsc{Inact}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad a\in\mathscr{B}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\setminus\{a\}\vdash (\nu a:\mathsf{G})P \blacktriangleright \mathscr{D}\setminus\{a\}}\ \{\textsc{NRes}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_1 \blacktriangleright \mathscr{D}_1 \quad \Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_2 \blacktriangleright \mathscr{D}_2}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_1 \mid P_2 \blacktriangleright (\mathscr{D}_1\cup\mathscr{D}_2)^{+}}\ \{\textsc{Par}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_1 \blacktriangleright \mathscr{D}_1 \quad \Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_2 \blacktriangleright \mathscr{D}_2}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \blacktriangleright (\mathscr{D}_1\cup\mathscr{D}_2)^{+}}\ \{\textsc{If}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c\oplus\langle\Pi,l\rangle.P \blacktriangleright \mathscr{D}}\ \{\textsc{Sel}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_i \blacktriangleright \mathscr{D}_i \quad \forall i\in I}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c\&(\mathrm{p},\{l_i:P_i\}_{i\in I}) \blacktriangleright (\mathsf{pre}(c,\bigcup_{i\in I}\mathsf{fc}(P_i))\cup\bigcup_{i\in I}\mathscr{D}_i)^{+}}\ \{\textsc{Branch}\}$$

$$\frac{e\in\mathscr{S}\Rightarrow e\in\mathscr{N}\cup\mathscr{B}}{\Theta,X[y] \blacktriangleright \mathscr{D};\mathscr{R};\mathscr{N};\mathscr{B}\vdash X\langle e,c\rangle \blacktriangleright \mathscr{D}\{\curlywedge(c)/y\}}\ \{\textsc{Var}\}$$

$$\frac{\Theta,X[y] \blacktriangleright \mathscr{D};\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \qquad \Theta,X[y] \blacktriangleright \mathscr{D};\mathscr{R};\mathscr{N};\mathscr{B}\vdash Q \blacktriangleright \mathscr{D}'}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \mathsf{def}\ X(x,y)=P\ \mathsf{in}\ Q \blacktriangleright \mathscr{D}'}\ \{\textsc{Def}\}$$

Summary

Before: two ways of assuring (dead)lock freedom:

- refined type systems for process calculi
  - + are compositional
  - - do not deal well with recursive processes and cyclic network topologies
- global types
  - + deal well with recursive processes and cyclic network topologies
  - - are not compositional

Now: the new form of polymorphic typing:

- + deal well with recursive processes and cyclic network topologies
- + is compositional

anyway

$$*c?(x,y).x?(z).y?(z) \mid *c?(x,y).y?(z).x?(z)$$

is not typable here but it is typable in

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Summary

Before: two ways of assuring (dead)lock freedom:

- refined type systems for process calculi
  - + are compositional
  - - do not deal well with recursive processes and cyclic network topologies
- global types
  - + deal well with recursive processes and cyclic network topologies
  - - are not compositional

Now: the new form of polymorphic typing:

- + deal well with recursive processes and cyclic network topologies
- + is compositional

anyway

$$*c?(x,y).x?(z).y?(z) \mid *c?(x,y).y?(z).x?(z)$$

is not typable here but it is typable in

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Summary

Before: two ways of assuring (dead)lock freedom:

- refined type systems for process calculi
  - + are compositional
  - - do not deal well with recursive processes and cyclic network topologies
- global types
  - + deal well with recursive processes and cyclic network topologies
  - - are not compositional

Now: the new form of polymorphic typing:

- + deal well with recursive processes and cyclic network topologies
- + is compositional

anyway

$$*c?(x,y).x?(z).y?(z) \mid *c?(x,y).y?(z).x?(z)$$

is not typable here but it is typable in

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Summary

Before: two ways of assuring (dead)lock freedom:

- refined type systems for process calculi
  - + are compositional
  - - do not deal well with recursive processes and cyclic network topologies
- global types
  - + deal well with recursive processes and cyclic network topologies
  - - are not compositional

Now: the new form of polymorphic typing:

- + deal well with recursive processes and cyclic network topologies
- + is compositional

anyway

$$*c?(x,y).x?(z).y?(z) \mid *c?(x,y).y?(z).x?(z)$$

is not typable here but it is typable in

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Summary

Before: two ways of assuring (dead)lock freedom:

- refined type systems for process calculi
  - + are compositional
  - - do not deal well with recursive processes and cyclic network topologies
- global types
  - + deal well with recursive processes and cyclic network topologies
  - - are not compositional

Now: the new form of polymorphic typing:

- + deal well with recursive processes and cyclic network topologies
- + is compositional

anyway

$$*c?(x,y).x?(z).y?(z) \mid *c?(x,y).y?(z).x?(z)$$

is not typable here but it is typable in

- Kobayashi, **A New Type System for Deadlock-Free Processes**, CONCUR 2006

Thanks


obrigado
Dank U
Merci
mahalo
Köszi
спасибо
Grazie
Thank you
mauruuru
Takk
Gracias
Dziękuję
Děkuju
danke
Kiitos