Types for the π-calculus

Typing processes

- we now want to express some properties of π-calculus terms

- we have seen some cases where the usage of names should obey a certain discipline

▷ polyadic π: runtime errors like $\overline{a}\langle v\rangle.P \mid a(x,y).Q \quad \longrightarrow \quad ??$

▷ booleans: *trigger* channels are not used like channels acting as boolean locations

▷ . . .

- *types:*

▷ less knowledge in a concurrent setting than for sequential languages

▷ original w.r.t. CCS

Simply typed π-calculus: types for channels

T	$=$		types
		$\# \tilde{T}$	medium

Simply typed π-calculus: types for channels

T	$=$		types
		$\#\tilde{T}$	medium
P	$=$		processes
		$\overline{n}\langle v\rangle.P$	
	$\mid$	$n(v).P$	

Simply typed π-calculus: types for channels

$$
\begin{array}{lcll}
T & = & & \text{types} \\
 & & \#\tilde{T} & \text{medium} \\
P & = & & \text{processes} \\
 & & \overline{n}\langle v\rangle.P & \\
 & | & n(v).P & \\
 & | & (\boldsymbol{\nu} x : T)\, P & \\
 & | & \ldots &
\end{array}
$$

Simply typed π-calculus: types for channels

T	$=$		types
		$\#\tilde{T}$	medium
P	$=$		processes
		$\overline{n}\langle v\rangle.P$	
	$\mid$	$n(v).P$	
	$\mid$	$(\boldsymbol{\nu} x : T)\, P$	
	$\mid$	$\ldots$	
Γ	$=$	$\emptyset \mid \Gamma, x : T$	typing contexts

Simply typed π-calculus: types for channels

$$
\begin{array}{rcll}
T & = & & \text{types} \\
 & & \#\tilde{T} & \text{medium} \\
P & = & & \text{processes} \\
 & & \overline{n}\langle v\rangle.P & \\
 & \mid & n(v).P & \\
 & \mid & (\boldsymbol{\nu} x : T)\, P & \\
 & \mid & \ldots & \\
\Gamma & = & \emptyset \mid \Gamma, x : T & \text{typing contexts}
\end{array}
$$

- $\#\langle\rangle$ is the base type

Simply typed π-calculus: types for channels

$$
\begin{array}{rcll}
T & = & & \text{types} \\
 & & \#\tilde{T} & \text{medium} \\
P & = & & \text{processes} \\
 & & \overline{n}\langle v\rangle.P & \\
 & | & n(v).P & \\
 & | & (\boldsymbol{\nu} x : T)\, P & \\
 & | & \ldots & \\
\Gamma & = & \emptyset \;\;|\;\; \Gamma, x : T & \text{typing contexts}
\end{array}
$$

- $\#\langle\rangle$ is the base type
- restriction *(name allocation)* is typed, input is not

Simply typed π-calculus: types for channels

$$
\begin{array}{rcll}
T & = & & \text{types} \\
 & & \#\tilde{T} & \text{medium} \\
P & = & & \text{processes} \\
 & & \overline{n}\langle v\rangle.P & \\
 & | & n(v).P & \\
 & | & (\boldsymbol{\nu} x : T)\, P & \\
 & | & \ldots & \\
\Gamma & = & \emptyset \;\; | \;\; \Gamma, x : T & \text{typing contexts}
\end{array}
$$

- $\#\langle\rangle$ is the base type
- restriction *(name allocation)* is typed, input is not
- we often write $\#T$ instead of $\#\langle T\rangle$

Types in π – examples

- $\overline{a}\langle t\rangle \mid a(x).\overline{x}.T$

Types in π – examples

- $\overline{a}\langle t\rangle \mid a(x).\overline{x}.T$ $\qquad a : \#\#\langle\rangle, \qquad x : \#\langle\rangle$

Types in π – examples

- $\overline{a}\langle t\rangle \mid a(x).\overline{x}.T$ $\quad a : \#\#\langle\rangle, \quad x : \#\langle\rangle$
- $b(x, y).\left(\overline{y}\langle x\rangle \mid x.P\right)$

Types in π – examples

- $\overline{a}\langle t\rangle \mid a(x).\overline{x}.T$ $\quad a : \#\#\langle\rangle, \quad x : \#\langle\rangle$
- $b(x,y).\Big(\overline{y}\langle x\rangle \mid x.P\Big)$ $\quad b : \#\langle\#\langle\rangle, \#\#\langle\rangle\rangle$

Types in π – examples

- $\overline{a}\langle t\rangle \mid a(x).\overline{x}.T$ $\quad a : \#\#\langle\rangle, \quad x : \#\langle\rangle$
- $b(x,y).\Big(\overline{y}\langle x\rangle \mid x.P\Big)$ $\quad b : \#\langle\#\langle\rangle, \#\#\langle\rangle\rangle$
- $\overline{a}\langle x,x\rangle \mid a(y).P$

Types in π – examples

- $\overline{a}\langle t\rangle \mid a(x).\overline{x}.T$ $\quad a : \#\#\langle\rangle, \quad x : \#\langle\rangle$
- $b(x,y).\Big(\overline{y}\langle x\rangle \mid x.P\Big)$ $\quad b : \#\langle\#\langle\rangle, \#\#\langle\rangle\rangle$
- $\overline{a}\langle x,x\rangle \mid a(y).P$ $\quad$ *not typeable*

Types in π – examples

- $\overline{a}\langle t \rangle \mid a(x).\overline{x}.T$ $\quad a : \#\#\langle\rangle, \quad x : \#\langle\rangle$
- $b(x,y).\Big(\overline{y}\langle x \rangle \mid x.P\Big)$ $\quad b : \#\langle \#\langle\rangle, \#\#\langle\rangle\rangle$
- $\overline{a}\langle x, x \rangle \mid a(y).P$ $\quad$ *not typeable*
- $\overline{a}\langle a \rangle$

Types in π – examples

- $\overline{a}\langle t\rangle \mid a(x).\overline{x}.T$ $\quad a : \#\#\langle\rangle, \quad x : \#\langle\rangle$
- $b(x,y).\Big(\overline{y}\langle x\rangle \mid x.P\Big)$ $\quad b : \#\langle\#\langle\rangle, \#\#\langle\rangle\rangle$
- $\overline{a}\langle x,x\rangle \mid a(y).P$ *not typeable*
- $\overline{a}\langle a\rangle$ *recursive types?*

Types in π – examples

- $\overline{a}\langle t\rangle \mid a(x).\overline{x}.T$ $\quad a : \#\#\langle\rangle, \quad x : \#\langle\rangle$
- $b(x,y).\left(\overline{y}\langle x\rangle \mid x.P\right)$ $\quad b : \#\langle\#\langle\rangle, \#\#\langle\rangle\rangle$
- $\overline{a}\langle x,x\rangle \mid a(y).P$ *not typeable*
- $\overline{a}\langle a\rangle$ *recursive types?*

typing judgments:

$\Gamma \vdash n : T$: n has type T $\quad$ $\Gamma \vdash P$: P obeys Γ

(compare with $\Gamma \vdash M : T$: as usual, we observe*)*

Typing rules

- values

$$\frac{\Gamma(n) = T}{\Gamma \vdash n : T} \qquad \frac{\Gamma \vdash n_i : T_i}{\Gamma \vdash (n_1, \ldots, n_k) : \langle T_1, \ldots, T_k \rangle}$$

Typing rules

- values

$$\frac{\Gamma(n) = T}{\Gamma \vdash n : T} \qquad \frac{\Gamma \vdash n_i : T_i}{\Gamma \vdash (n_1, \ldots, n_k) : \langle T_1, \ldots, T_k \rangle}$$

- processes

$$\frac{\Gamma \vdash P \quad \Gamma \vdash n : \#\tilde{T} \quad \Gamma \vdash \tilde{m} : \tilde{T}}{\Gamma \vdash \overline{n}\langle \tilde{m} \rangle.P} \qquad \frac{\Gamma, \tilde{x} : \tilde{T} \vdash P \quad \Gamma \vdash n : \#\tilde{T}}{\Gamma \vdash n(\tilde{x}).P}$$

Typing rules

- values

$$\frac{\Gamma(n) = T}{\Gamma \vdash n : T} \qquad \frac{\Gamma \vdash n_i : T_i}{\Gamma \vdash (n_1, \ldots, n_k) : \langle T_1, \ldots, T_k \rangle}$$

- processes

$$\frac{\Gamma \vdash P \quad \Gamma \vdash n : \#\tilde{T} \quad \Gamma \vdash \tilde{m} : \tilde{T}}{\Gamma \vdash \overline{n}\langle \tilde{m} \rangle.P} \qquad \frac{\Gamma, \tilde{x} : \tilde{T} \vdash P \quad \Gamma \vdash n : \#\tilde{T}}{\Gamma \vdash n(\tilde{x}).P}$$

$$\frac{\Gamma, x : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} x : T)\, P} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \,|\, Q} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P} \qquad \Gamma \vdash \mathbf{0}$$

Typing rules

- values

$$\frac{\Gamma(n) = T}{\Gamma \vdash n : T} \qquad \frac{\Gamma \vdash n_i : T_i}{\Gamma \vdash (n_1, \ldots, n_k) : \langle T_1, \ldots, T_k \rangle}$$

- processes

$$\frac{\Gamma \vdash P \quad \Gamma \vdash n : \sharp\tilde{T} \quad \Gamma \vdash \tilde{m} : \tilde{T}}{\Gamma \vdash \overline{n}\langle\tilde{m}\rangle.P} \qquad \frac{\Gamma, \tilde{x} : \tilde{T} \vdash P \quad \Gamma \vdash n : \sharp\tilde{T}}{\Gamma \vdash n(\tilde{x}).P}$$

$$\frac{\Gamma, x : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} x : T)\, P} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \,|\, Q} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash\, !P} \qquad \Gamma \vdash \mathbf{0}$$

Remark: in the rule for $|$, P and Q must agree on a typing
– there exists a more ‘algorithmical’ presentation

Type system – metatheoretical properties

Proposition [strengthening]

If $\Gamma, x : T \vdash P$ and $x \notin \mathsf{fn}(P)$, then $\Gamma \vdash P$.

Proposition [weakening]

If $\Gamma \vdash P$ and Γ is not defined on x, then $\Gamma, x : T \vdash P$.

Lemma [substitution lemma]

If $\Gamma \vdash P$, $\Gamma \vdash x : T$ and $\Gamma \vdash v : T$, then $\Gamma \vdash P_{\{x \leftarrow v\}}$.

Type system – metatheoretical properties

Proposition [strengthening]

If $\Gamma, x : T \vdash P$ and $x \notin \mathsf{fn}(P)$, then $\Gamma \vdash P$.

Proposition [weakening]

If $\Gamma \vdash P$ and Γ is not defined on x, then $\Gamma, x : T \vdash P$.

Lemma [substitution lemma]

If $\Gamma \vdash P$, $\Gamma \vdash x : T$ and $\Gamma \vdash v : T$, then $\Gamma \vdash P_{\{x \leftarrow v\}}$.

Theorem [arity checking]

If $\Gamma \vdash (\boldsymbol{\nu} \tilde{n}) \left(\overline{a}\langle m_1, \ldots, m_k\rangle.P_1 \mid a(x_1, \ldots, x_l).P_2 \mid P_3\right)$, then $k = l$.

Type system – metatheoretical properties

Proposition [strengthening]

If $\Gamma, x : T \vdash P$ and $x \notin \mathsf{fn}(P)$, then $\Gamma \vdash P$.

Proposition [weakening]

If $\Gamma \vdash P$ and Γ is not defined on x, then $\Gamma, x : T \vdash P$.

Lemma [substitution lemma]

If $\Gamma \vdash P$, $\Gamma \vdash x : T$ and $\Gamma \vdash v : T$, then $\Gamma \vdash P_{\{x \leftarrow v\}}$.

Theorem [arity checking]

If $\Gamma \vdash (\boldsymbol{\nu}\tilde{n})\left(\overline{a}\langle m_1, \ldots, m_k\rangle.P_1 \mid a(x_1, \ldots, x_l).P_2 \mid P_3\right)$, then $k = l$.

Theorem [subject reduction]

If $\Gamma \vdash P$ and $P \rightarrow P'$, then $\Gamma \vdash P'$.

Type system – metatheoretical properties

Proposition [strengthening]

If $\Gamma, x : T \vdash P$ and $x \notin \mathsf{fn}(P)$, then $\Gamma \vdash P$.

Proposition [weakening]

If $\Gamma \vdash P$ and Γ is not defined on x, then $\Gamma, x : T \vdash P$.

Lemma [substitution lemma]

If $\Gamma \vdash P$, $\Gamma \vdash x : T$ and $\Gamma \vdash v : T$, then $\Gamma \vdash P_{\{x \leftarrow v\}}$.

Theorem [arity checking]

If $\Gamma \vdash (\boldsymbol{\nu}\tilde{n}) \left(\overline{a}\langle m_1, \ldots, m_k\rangle.P_1 \mid a(x_1, \ldots, x_l).P_2 \mid P_3\right)$, then $k = l$.

Theorem [subject reduction]

If $\Gamma \vdash P$ and $P \rightarrow P'$, then $\Gamma \vdash P'$.

Corollary [soundness] No run-time error.

Examples – exercises

- suppose $\Gamma \vdash P$, and $a, b \notin \mathsf{fn}(P)$; then

$$\Gamma, a : \#\langle\rangle, b : \#\#\langle\rangle \;\; \vdash \;\; \overline{a}\langle\rangle \mid a.\overline{b}\langle a\rangle \mid b(x).P$$

Examples – exercises

- suppose $\Gamma \vdash P$, and $a, b \notin \mathsf{fn}(P)$; then

$$\Gamma, a : \#\langle\rangle, b : \#\#\langle\rangle \;\vdash\; \overline{a}\langle\rangle \mid a.\overline{b}\langle a\rangle \mid b(x).P$$

- can the following terms be typed?

$$P \;=\; b(y).\overline{y}\langle\rangle \mid \overline{a}\langle b\rangle \mid a(x).\overline{x}\langle c\rangle \qquad\qquad Q \;=\; \overline{a}\langle b\rangle \mid a(x).\overline{x}\langle\rangle \mid b(y).\overline{y}\langle\rangle$$

Examples – exercises

- suppose $\Gamma \vdash P$, and $a, b \notin \mathsf{fn}(P)$; then

$$\Gamma, a : \#\langle\rangle, b : \#\#\langle\rangle \;\vdash\; \overline{a}\langle\rangle \mid a.\overline{b}\langle a\rangle \mid b(x).P$$

- can the following terms be typed?

$$P \;=\; b(y).\overline{y}\langle\rangle \mid \overline{a}\langle b\rangle \mid a(x).\overline{x}\langle c\rangle \qquad\qquad Q \;=\; \overline{a}\langle b\rangle \mid a(x).\overline{x}\langle\rangle \mid b(y).\overline{y}\langle\rangle$$

- type the booleans:

$$\Gamma \vdash !b(x,y).\overline{x}\langle\rangle \qquad\qquad \Gamma ?$$

Types and finiteness

- the simply typed λ-calculus is terminating

Types and finiteness

- the simply typed λ-calculus is terminating
- an example of a typeable non terminating process?

Types and finiteness

- the simply typed λ-calculus is terminating
- an example of a typeable non terminating process?
- logical threads:

▷ labelled transitions: $P \xrightarrow{\mu} P'$, μ: input, output, τ

Types and finiteness

- the simply typed λ-calculus is terminating

- an example of a typeable non terminating process?

- logical threads:

▷ labelled transitions: $P \xrightarrow{\mu} P'$, μ: input, output, τ

▷ P exhibits *trace* $\alpha_1, \ldots \alpha_n, \ldots$ (possibly infinite):
$P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} P_2 \ldots$

Types and finiteness

- the simply typed λ-calculus is terminating
- an example of a typeable non terminating process?
- logical threads:

▷ labelled transitions: $P \xrightarrow{\mu} P'$, μ: input, output, τ

▷ P exhibits *trace* $\alpha_1, \dots \alpha_n, \dots$ (possibly infinite):
$P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} P_2 \dots$

▷ $\alpha_1, \dots, \alpha_n, \dots$ is a *logical thread*:
the object of α_i is the subject of α_{i+1}

Types and finiteness

- the simply typed λ-calculus is terminating

- an example of a typeable non terminating process?

- logical threads:

▷ labelled transitions: $P \xrightarrow{\mu} P'$, μ: input, output, τ

▷ P exhibits *trace* $\alpha_1, \ldots \alpha_n, \ldots$ (possibly infinite):
$P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} P_2 \ldots$

▷ $\alpha_1, \ldots, \alpha_n, \ldots$ is a *logical thread*:
the object of α_i is the subject of α_{i+1}

- **Theorem:** *If* $\Gamma \vdash P$*, then any logical thread* extracted from *a trace of* P *is of finite length.*

Subtyping

A more accurate analysis of channel usage

- channel types: $a : \#T$ $\quad$ a is used to carry Ts

A more accurate analysis of channel usage

- channel types: $a : \# T$ $\quad$ a is used to carry Ts
- motivating example: factorial server

$$SPEC \stackrel{\text{def}}{=} !f(x,r).\overline{r}\langle \mathtt{fact}(x) \rangle$$

A more accurate analysis of channel usage

- channel types: $a : \#T$ a is used to carry Ts
- motivating example: factorial server

$$SPEC \stackrel{\text{def}}{=} !f(x,r).\overline{r}\langle \texttt{fact}(x)\rangle$$

$$IMPL \stackrel{\text{def}}{=} !f(x,r).\texttt{if } x = 0 \texttt{ then } \overline{r}\,1 \quad \texttt{else } (\boldsymbol{\nu} r')\,(\overline{f}\langle x-1\rangle \mid r'(m).\overline{r}\langle x * m\rangle)$$

A more accurate analysis of channel usage

- channel types: $a : \#T$ $\qquad$ a is used to carry Ts
- motivating example: factorial server

$$SPEC \stackrel{\text{def}}{=} \,!f(x,r).\overline{r}\langle \mathtt{fact}(x)\rangle$$

$$IMPL \stackrel{\text{def}}{=} \,!f(x,r).\mathtt{if}\ x = 0\ \mathtt{then}\ \overline{r}\,1 \quad \mathtt{else}\ (\boldsymbol{\nu} r')\,(\overline{f}\langle x-1\rangle \mid r'(m).\overline{r}\langle x * m\rangle)$$

$IMPL$ is ok as long as no evil agent tries to receive on f

N.B.: f has to be made public!

A more accurate analysis of channel usage

- channel types: $a : \#T$ $\quad$ a is used to carry Ts
- motivating example: factorial server

$$SPEC \stackrel{\text{def}}{=} !f(x,r).\overline{r}\langle \texttt{fact}(x)\rangle$$

$$IMPL \stackrel{\text{def}}{=} !f(x,r).\texttt{if } x = 0 \texttt{ then } \overline{r}\,1 \\ \texttt{else } (\boldsymbol{\nu} r')\,(\overline{f}\langle x-1\rangle \mid r'(m).\overline{r}\langle x * m\rangle)$$

$IMPL$ is ok as long as no evil agent tries to receive on f

N.B.: f has to be made public!

- idea: separate *input* and *output capabilities* on f

. . . and pass only output capability

input/output types

- enrich the grammar for types:

$$
\begin{array}{rcll}
T & = & & \text{types} \\
& | & \#\tilde{T} & \text{medium (bidirectional communication)} \\
& | & \mathsf{i}\tilde{T} & \text{reception} \\
& | & \mathsf{o}\tilde{T} & \text{emission}
\end{array}
$$

input/output types

- enrich the grammar for types:

$$
\begin{array}{rcll}
T & = & & \text{types} \\
 & | & \#\tilde{T} & \text{medium (bidirectional communication)} \\
 & | & \mathsf{i}\tilde{T} & \text{reception} \\
 & | & \mathsf{o}\tilde{T} & \text{emission}
\end{array}
$$

- judgment: $\Gamma \vdash P$ $\quad$ Γ captures P's point of view

Ex: $\Gamma, c : \mathsf{oi}S \vdash P$

P may only *emit* on c, and the receiver will only be able to listen on the transmitted channel

input/output types

- enrich the grammar for types:

$$
\begin{array}{rcll}
T & = & & \text{types} \\
 & | & \#\tilde{T} & \text{medium (bidirectional communication)} \\
 & | & \mathsf{i}\tilde{T} & \text{reception} \\
 & | & \mathsf{o}\tilde{T} & \text{emission}
\end{array}
$$

- judgment: $\Gamma \vdash P$ Γ captures P's point of view

Ex: $\Gamma, c : \mathsf{oi}S \vdash P$

P may only *emit* on c, and the receiver will only be able to listen on the transmitted channel

localised π:

input/output types

- enrich the grammar for types:

$$
\begin{array}{rcll}
T & = & & \text{types} \\
 & | & \#\tilde{T} & \text{medium (bidirectional communication)} \\
 & | & \mathsf{i}\tilde{T} & \text{reception} \\
 & | & \mathsf{o}\tilde{T} & \text{emission}
\end{array}
$$

- judgment: $\Gamma \vdash P$ $\qquad$ Γ captures P's point of view

Ex: $\qquad \Gamma, c : \mathsf{oi}S \vdash P$

P may only *emit* on c, and the receiver will only be able to listen on the transmitted channel

localised π: $\qquad \Gamma, n : \#\mathsf{o}T \vdash n(x).P$

Typing rules

- more precise inference rules for emission/reception:

$$\frac{\Gamma \vdash n : \mathsf{i}\tilde{T} \qquad \Gamma, \tilde{x} : \tilde{T} \vdash P}{\Gamma \vdash n(\tilde{x}).P}$$

$$\frac{\Gamma \vdash n : \mathsf{o}\tilde{T} \qquad \Gamma \vdash \tilde{m} : \tilde{T} \qquad \Gamma \vdash P}{\Gamma \vdash \overline{n}\langle m\rangle.P}$$

Typing rules

- more precise inference rules for emission/reception:

$$\frac{\Gamma \vdash n : \mathsf{i}\tilde{T} \qquad \Gamma, \tilde{x} : \tilde{T} \vdash P}{\Gamma \vdash n(\tilde{x}).P}$$

$$\frac{\Gamma \vdash n : \mathsf{o}\tilde{T} \qquad \Gamma \vdash \tilde{m} : \tilde{T} \qquad \Gamma \vdash P}{\Gamma \vdash \overline{n}\langle m \rangle.P}$$

- *if I can do both, I can do one*

in other words, $\Gamma, a : \#T \vdash a : \mathsf{i}T$ and $\Gamma, a : \#T \vdash a : \mathsf{i}T$

Typing rules

- more precise inference rules for emission/reception:

$$\frac{\Gamma \vdash n : \mathsf{i}\tilde{T} \qquad \Gamma, \tilde{x} : \tilde{T} \vdash P}{\Gamma \vdash n(\tilde{x}).P}$$

$$\frac{\Gamma \vdash n : \mathsf{o}\tilde{T} \qquad \Gamma \vdash \tilde{m} : \tilde{T} \qquad \Gamma \vdash P}{\Gamma \vdash \overline{n}\langle m\rangle.P}$$

- *if I can do both, I can do one*

in other words, $\Gamma, a : \#T \vdash a : \mathsf{i}T$ and $\Gamma, a : \#T \vdash a : \mathsf{i}T$

$\longrightarrow$ introduce *subtyping*

Subtyping

- $T \leq U$: whenever a U is requested, I may use a T

Subtyping

- $T \leq U$: whenever a U is requested, I may use a T
- *subsumption* rule:

$$\frac{\Gamma \vdash v : T \qquad T \leq U}{\Gamma \vdash v : U}$$

Subtyping

- $T \leq U$: whenever a U is requested, I may use a T
- *subsumption* rule:

$$\frac{\Gamma \vdash v : T \qquad T \leq U}{\Gamma \vdash v : U}$$

- in our case, $\#T \leq \mathsf{i}T$ and $\#T \leq \mathsf{o}T$

Subtyping

- $T \leq U$: whenever a U is requested, I may use a T
- *subsumption* rule:

$$\frac{\Gamma \vdash v : T \qquad T \leq U}{\Gamma \vdash v : U}$$

- in our case, $\#T \leq \mathsf{i}T$ and $\#T \leq \mathsf{o}T$
- subtyping brings *flexibility:*

  in a situation where P is needed s.t. $\Gamma, c : U \vdash P$,
  knowing $\Gamma, c : T \vdash P_0$, we can safely plug P_0 *(Narrowing)*

Subtyping and capabilities

- suppose $T \leq U$; should we deduce $\#T \leq \#U$?

Subtyping and capabilities

- suppose $T \leq U$; should we deduce $\#T \leq \#U$?
- consider $\mathtt{Int} \leq \mathtt{Real}$, *(additional base types)*

  i.e. any time a **Real** is requested, I can use an **Int**

Subtyping and capabilities

- suppose $T \leq U$; should we deduce $\#T \leq \#U$?
- consider $\texttt{Int} \leq \texttt{Real}$, *(additional base types)*

  i.e. any time a **Real** is requested, I can use an **Int**
- the answer is *No* because type soundness would crash:

$$a : \#\texttt{Int}, b : \#\texttt{Int} \vdash a(x).\bar{b}\langle x \ \texttt{mod}\ 2\rangle$$

Subtyping and capabilities

- suppose $T \leq U$; should we deduce $\#T \leq \#U$?
- consider $\texttt{Int} \leq \texttt{Real}$, *(additional base types)*

  i.e. any time a $\texttt{Real}$ is requested, I can use an $\texttt{Int}$
- the answer is *No* because type soundness would crash:

$$a : \#\texttt{Int}, b : \#\texttt{Int} \vdash a(x).\overline{b}\langle x \texttt{ mod } 2\rangle$$
$$a : \#\texttt{Int} \vdash a : \#\texttt{Real}, \text{ hence } a : \#\texttt{Int} \vdash \overline{a}\langle \texttt{3.02}\rangle$$

Subtyping and capabilities

- suppose $T \leq U$; should we deduce $\#T \leq \#U$?
- consider $\texttt{Int} \leq \texttt{Real}$, *(additional base types)*

  i.e. any time a $\texttt{Real}$ is requested, I can use an $\texttt{Int}$
- the answer is *No* because type soundness would crash:

$$a : \#\texttt{Int}, b : \#\texttt{Int} \vdash a(x).\bar{b}\langle x \texttt{ mod } 2\rangle$$

$$a : \#\texttt{Int} \vdash a : \#\texttt{Real}, \text{ hence } a : \#\texttt{Int} \vdash \overline{a}\langle \texttt{3.02}\rangle$$

$$\text{hence } a : \#\texttt{Int}, b : \#\texttt{Int} \vdash a(x).\bar{b}\langle x \texttt{ mod } 2\rangle \mid \overline{a}\langle \texttt{3.02}\rangle$$

Subtyping and capabilities

- suppose $T \leq U$; should we deduce $\#T \leq \#U$?
- consider $\texttt{Int} \leq \texttt{Real}$, *(additional base types)*

  i.e. any time a $\texttt{Real}$ is requested, I can use an $\texttt{Int}$
- the answer is **No** because type soundness would crash:

$$a : \#\texttt{Int}, b : \#\texttt{Int} \vdash a(x).\bar{b}\langle x \ \texttt{mod} \ 2\rangle$$
$$a : \#\texttt{Int} \vdash a : \#\texttt{Real}, \text{ hence } a : \#\texttt{Int} \vdash \overline{a}\langle \texttt{3.02}\rangle$$

hence $a : \#\texttt{Int}, b : \#\texttt{Int} \vdash a(x).\bar{b}\langle x \ \texttt{mod} \ 2\rangle \mid \overline{a}\langle \texttt{3.02}\rangle$

but $a(x).\bar{b}\langle x \ \texttt{mod} \ 2\rangle \mid \overline{a}\langle \texttt{3.02}\rangle \longrightarrow \bar{b}\langle \texttt{3.02 mod 2}\rangle$

Subtyping and i/o

- then if $T \leq U$,
  - $\star$ $\mathsf{i}T \leq \mathsf{i}U$?
  - $\star$ $\mathsf{o}T \leq \mathsf{o}U$?

- I am P, I have capabilities $c : \mathsf{i}T, d : \mathsf{o}T$
  can I use c as $\mathsf{i}U$? d as $\mathsf{o}U$?

Subtyping and i/o

- then if $T \leq U$,
  - $\star$ i$T \leq$ iU ?
  - $\star$ o$T \leq$ oU ?

- I am P, I have capabilities c : iT, d : oT
  can I use c as iU? d as oU?

  - answer: $$\frac{T \leq U}{\mathsf{i}T \leq \mathsf{i}U} \qquad \frac{T \leq U}{\mathsf{o}U \leq \mathsf{o}T}$$

Subtyping relation $\dfrac{\Gamma \vdash v : T \quad T \leq U}{\Gamma \vdash v : U}$

$$T \leq T \qquad \frac{T \leq U \quad U \leq V}{T \leq V} \qquad \text{preorder}$$

$$\#T \leq \mathsf{i}T \qquad \#T \leq \mathsf{o}T \qquad \text{giving up a capability}$$

$$\frac{T \leq U}{\mathsf{i}T \leq \mathsf{i}U} \qquad \frac{T \leq U}{\mathsf{o}U \leq \mathsf{o}T} \qquad \text{covariance / contravariance}$$

$$\frac{T \leq U \quad U \leq T}{\#T \leq \#U} \qquad \text{invariance}$$

- if $a : \mathsf{i}T$, consider what is arriving on a as "bigger"
- if $a : \mathsf{o}T$, one can use a to emit "smaller" values

Subtyping – examples $\frac{T \leq U}{\mathsf{i}T \leq \mathsf{i}U} \quad \frac{T \leq U}{\mathsf{o}U \leq \mathsf{o}T}$

- back to the example seen before:

$$a : \texttt{iInt}, b : \texttt{oInt} \vdash a(x).\bar{b}\langle x \texttt{ mod } 2\rangle \qquad a : \texttt{oReal} \vdash \overline{a}\langle \texttt{3.02}\rangle$$

$$\rightarrow \quad a : ?, b : \texttt{oInt} \vdash a(x).\bar{b}\langle x \texttt{ mod } 2\rangle \mid \overline{a}\langle 3.02\rangle\,,$$

no way: $a : \#\texttt{Int}$ or $a : \#\texttt{Real}$

Subtyping – examples

$$\frac{T \leq U}{\mathsf{i}T \leq \mathsf{i}U} \qquad \frac{T \leq U}{\mathsf{o}U \leq \mathsf{o}T}$$

- back to the example seen before:

$$a : \texttt{iInt}, b : \texttt{oInt} \vdash a(x).\bar{b}\langle x \texttt{ mod } 2\rangle \qquad a : \texttt{oReal} \vdash \overline{a}\langle \texttt{3.02}\rangle$$

$$\rightarrow \quad a : ?, b : \texttt{oInt} \vdash a(x).\bar{b}\langle x \texttt{ mod } 2\rangle \mid \overline{a}\langle 3.02\rangle \,,$$

no way: $a : \#\texttt{Int}$ or $a : \#\texttt{Real}$

- one possible derivation:

$$x : \mathsf{oo}T, z : \mathsf{oi}T \ \vdash \ (\boldsymbol{\nu} y : \#T)\,(\overline{x}\langle y\rangle \mid \overline{z}\langle y\rangle)$$

Back to the factorial

$$
\begin{aligned}
SPEC &\stackrel{\text{def}}{=} \; !f(x,r).\overline{r}\langle \texttt{fact}(x)\rangle \\
IMPL &\stackrel{\text{def}}{=} \; !f(x,r).\texttt{if}\ x = 0\ \texttt{then}\ \overline{r}\langle 1\rangle \\
&\qquad\qquad\qquad \texttt{else}\ (\boldsymbol{\nu} r')\,(\overline{f}\langle x-1\rangle \mid r'(m).\overline{r}\langle x * m\rangle)
\end{aligned}
$$

Back to the factorial

$$SPEC \stackrel{\text{def}}{=} !f(x,r).\overline{r}\langle \texttt{fact}(x)\rangle$$

$$IMPL \stackrel{\text{def}}{=} !f(x,r).\texttt{if } x = 0 \texttt{ then } \overline{r}\langle 1\rangle$$
$$\texttt{else } (\boldsymbol{\nu} r')\,(\overline{f}\langle x-1\rangle \mid r'(m).\overline{r}\langle x * m\rangle)$$

- we have now means to limit the usage of f:

$$a : \#\texttt{o}(\texttt{Int}, \texttt{oInt}) \vdash (\boldsymbol{\nu} f)\,(\overline{a}f \mid SPEC) \cong^c (\boldsymbol{\nu} f)\,(\overline{a}f \mid IMPL)$$

where $\cong^c$ is a *behavioural equivalence* (see later)

i/o types for the π-calculus – properties

- **Theorem:** *Subject Reduction*

i/o types for the π-calculus – properties

- **Theorem:** *Subject Reduction*

- **Corollary:** *suppose* $\Gamma \vdash P$ *and* $P \Rightarrow P'$ *(i.e.* $P \rightarrow^* P'$*);*
  (1) *if* $\Gamma(a) = \mathsf{i}T$ *then* a *occurs in* P'
  either in input subject or in output object position
  (2) *if* $\Gamma(a) = \mathsf{o}T$ *then* a *occurs in* P'
  in (subject or object) output position

Exercise: i/o capabilities and

- suppose $x : \mathsf{i}T, a : \#\mathsf{o}T \vdash P$, and $P \xrightarrow{a(x)} P'$

  $P \xrightarrow{a(x)} P'$: *P may receive x on a and become P'*

Exercise: i/o capabilities and

- suppose $x : \mathsf{i}T, a : \#\mathsf{o}T \vdash P$, and $P \xrightarrow{a(x)} P'$

  $P \xrightarrow{a(x)} P'$: P may receive x on a and become P'

- is P' typeable? if yes, explain informally how the typing derivation goes

- discuss on the precision of the 'i/o types' analysis

Remarks on i/o types

- types i/o and R/L-*values*: the type ref T is invariant
  distinguish two *uses* of a reference:
  R (*read value*) and L (*write value – location*)

Remarks on i/o types

- types i/o and R/L-*values*: the type ref T is invariant
  distinguish two *uses* of a reference:
  R (*read value*) and L (*write value – location*)

- using types to reason about terms

▷ more equivalences hold when imposing some typing on the context

▷ typed encodings: enforce some kind of “programming discipline”

Other type systems

- linearity
- receptiveness
- polymorphism
- . . .   (session types, graph types, types for termination)