Fundamentals of Session Types

Based on the paper by Vasco T. Vasconcelos

Bas van den Heuvel and Peter Thiemann

2024-07-10

Introduction

- The π-calculus allows all sorts of unexpected, wild behavior.
- We will use *types* to "tame" processes.
- The idea is that every channel is assigned a communication protocol expressed as a *session type*.
- Typing ensures that well-typed processes satisfy...
  - *Protocol fidelity*: a process uses every channel according to its session type.
  - *Communication safety*: no communication errors can occur.

Process syntax

Processes:

$P, Q ::=$	$\overline{x}v.P$	send
	$x(y).P$	receive
	$P \mid Q$	parallel composition
	$\text{if } v \text{ then } P \text{ else } Q$	conditional
	0	inaction
	$(\nu xy)P$	scope restriction

Values:

$v ::=$	x	variable
	true $\mid$ false	boolean values

- Channels consist of two *ends*: in $(\nu xy)P$, x is the server end and y the client end. We call x and y *co-variables*.
- Variable y occurs *bound* in $x(y).P$ and $(\nu xy)P$. Variable x occurs *bound* in $(\nu xy)P$.
- Non-bound variables are *free*: $\text{fv}(P)$.
- Capture-free substitution of x by v in P: $P[v/x]$.
- Process equality holds up to alpha conversion.
- Barendregt's variable convention: bound variables are pairwise distinct and distinct from free variables.
- We omit trailing ".0".

Operational semantics

Structural congruence $[P \equiv Q]$:

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid 0 \equiv P$$

$$(\nu xy)P \mid Q \equiv (\nu xy)(P \mid Q)^1 \qquad (\nu xy)0 \equiv 0 \qquad (\nu wx)(\nu yz)P \equiv (\nu yz)(\nu wx)P$$

Reduction rules $(P \rightarrow P)$:

$$\text{[R-Com]} \quad \frac{}{(\nu xy)(\overline{x}v.P \mid y(z).Q \mid R) \rightarrow (\nu xy)(P \mid Q[v/z] \mid R)}$$

$$\text{[R-IfT]} \quad \frac{}{\text{if true then } P \text{ else } Q \rightarrow P} \qquad \text{[R-IfF]} \quad \frac{}{\text{if false then } P \text{ else } Q \rightarrow Q}$$

$$\text{[R-Res]} \quad \frac{P \rightarrow Q}{(\nu xy)P \rightarrow (\nu xy)Q} \qquad \text{[R-Par]} \quad \frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R} \qquad \text{[R-Struct]} \quad \frac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q}$$

[1] Condition $x, y \notin \mathrm{fv}(Q)$ not necessary by variable convention.

Undefined behavior

- $(\nu x_1 x_2)(\overline{x_1}\,\mathsf{true} \mid x_2(y).\overline{y}\,\mathsf{false})$
- $(\nu x_1 x_2)\ \mathsf{if}\ x_1\ \mathsf{then}\ 0\ \mathsf{else}\ 0$
- $\overline{x}\,\mathsf{true} \mid x(y)$

Undefined behavior

- $(\nu x_1 x_2)(\overline{x_1}\,\mathsf{true} \mid x_2(y).\overline{y}\,\mathsf{false})$
  Substitution $[\mathsf{true}/y]$ yields a stuck process: cannot send on a boolean value.
- $(\nu x_1 x_2)\ \mathsf{if}\ x_1\ \mathsf{then}\ 0\ \mathsf{else}\ 0$
- $\overline{x}\,\mathsf{true} \mid x(y)$

Undefined behavior

- $(\nu x_1 x_2)(\overline{x_1}\,\mathsf{true} \mid x_2(y).\overline{y}\,\mathsf{false})$
  Substitution $[\mathsf{true}/y]$ yields a stuck process: cannot send on a boolean value.
- $(\nu x_1 x_2)\,\mathsf{if}\,x_1\,\mathsf{then}\,0\,\mathsf{else}\,0$
  Cannot test a channel end rather than a boolean value.
- $\overline{x}\,\mathsf{true} \mid x(y)$

Undefined behavior

- $(\nu x_1 x_2)(\overline{x_1}\,\mathsf{true} \mid x_2(y).\overline{y}\,\mathsf{false})$
  Substitution $[\mathsf{true}\,/y]$ yields a stuck process: cannot send on a boolean value.
- $(\nu x_1 x_2)\,\mathsf{if}\,x_1\,\mathsf{then}\,0\,\mathsf{else}\,0$
  Cannot test a channel end rather than a boolean value.
- $\overline{x}\,\mathsf{true} \mid x(y)$
  Both threads try to write and read simultaneously on the same channel end.

Types

Qualifiers:

$q ::=$	lin	linear
	un	unrestricted

Pretypes:

$p ::=$	$?T.S$	receive
	$!T.S$	send

Types:

$S, T, U ::=$	bool	boolean
	end	termination
	qp	qualified pretype

Contexts:

$\Gamma ::=$	$\emptyset$	empty context
	$\Gamma, x : T$	assumption

- Linearity: value of type lin T must be used *exactly once*.
- Unrestricted: value of type un T can be used *zero or more times*.

Types

Qualifiers:		
$q ::=$	lin	linear
	un	unrestricted

Pretypes:		
$p ::=$	$?T.S$	receive
	$!T.S$	send

Types:		
$S,T,U ::=$	bool	boolean
	end	termination
	qp	qualified pretype

Contexts:		
$\Gamma ::=$	$\emptyset$	empty context
	$\Gamma, x : T$	assumption

- Linearity: value of type lin T must be used *exactly once*.
- Unrestricted: value of type un T can be used *zero or more times*.
- We can freely reorder assumptions in a context.
- We omit all linear type qualifier and only annotate unrestricted types.
- We omit trailing ". end".
- Co-variables, even if not explicitly under restriction, are annotated, e.g., (x_1, x_2).
- Variable x for (arbitrarily) qualified type, a for unrestricted type, c for linear type.

Types

- Linearity: value of type $\mathsf{lin}\, T$ must be used *exactly once*.
- Unrestricted: value of type $\mathsf{un}\, T$ can be used *zero or more times*.
- We can freely reorder assumptions in a context.
- We omit all linear type qualifier and only annotate unrestricted types.
- We omit trailing ". end".
- Co-variables, even if not explicitly under restriction, are annotated, e.g., (x_1, x_2).
- Variable x for (arbitrarily) qualified type, a for unrestricted type, c for linear type.

Well formed?

- $\overline{a}\,\mathsf{true} \mid \overline{a}\,\mathsf{true} \mid \overline{a}\,\mathsf{false}$
- $\overline{c}\,\mathsf{true} \mid \overline{c}\,\mathsf{false}$

Types

- Linearity: value of type $\mathsf{lin}\, T$ must be used *exactly once*.
- Unrestricted: value of type $\mathsf{un}\, T$ can be used *zero or more times*.
- We can freely reorder assumptions in a context.
- We omit all linear type qualifier and only annotate unrestricted types.
- We omit trailing ". end".
- Co-variables, even if not explicitly under restriction, are annotated, e.g., (x_1, x_2).
- Variable x for (arbitrarily) qualified type, a for unrestricted type, c for linear type.

Well formed?

- $\overline{a}\,\mathsf{true} \mid \overline{a}\,\mathsf{true} \mid \overline{a}\,\mathsf{false}$

  Yes.

- $\overline{c}\,\mathsf{true} \mid \overline{c}\,\mathsf{false}$

Types

- Linearity: value of type lin T must be used *exactly once*.
- Unrestricted: value of type un T can be used *zero or more times*.
- We can freely reorder assumptions in a context.
- We omit all linear type qualifier and only annotate unrestricted types.
- We omit trailing ". end".
- Co-variables, even if not explicitly under restriction, are annotated, e.g., (x_1, x_2).
- Variable x for (arbitrarily) qualified type, a for unrestricted type, c for linear type.

Well formed?

- $\overline{a}$ true | $\overline{a}$ true | $\overline{a}$ false

  Yes.

- $\overline{c}$ true | $\overline{c}$ false

  No.

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Should we accept these processes?

- $\overline{x_1}\,\mathsf{true} \mid x_2(z)$
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).\overline{c_2}\,\mathsf{false}$
- $\overline{x_1}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false}$
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).c_2(t)$

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Should we accept these processes?

- $\overline{x_1}\,\mathsf{true} \mid x_2(z)$
  Yes: $x_1 : q!\,\mathsf{bool}\,,\, x_2 : q?\,\mathsf{bool} = \overline{q!\,\mathsf{bool}}$.
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).\overline{c_2}\,\mathsf{false}$
- $\overline{x_1}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false}$
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).c_2(t)$

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Should we accept these processes?

- $\overline{x_1}\,\mathsf{true} \mid x_2(z)$
  Yes: $x_1 : q!\,\mathsf{bool}\,, x_2 : q?\,\mathsf{bool} = \overline{q!\,\mathsf{bool}}$.
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).\overline{c_2}\,\mathsf{false}$
  Yes: $c_1 : \mathsf{lin}\,!\,\mathsf{bool}\,.\,\mathsf{lin}\,?\,\mathsf{bool}\,, c_2 : \mathsf{lin}\,?\,\mathsf{bool}\,.\,\mathsf{lin}\,!\,\mathsf{bool} = \overline{\mathsf{lin}\,!\,\mathsf{bool}\,.\,\mathsf{lin}\,?\,\mathsf{bool}}$.
- $\overline{x_1}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false}$
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).c_2(t)$

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Should we accept these processes?

- $\overline{x_1}\,\mathsf{true} \mid x_2(z)$
  Yes: $x_1 : q!\,\mathsf{bool}\,, x_2 : q?\,\mathsf{bool} = \overline{q!\,\mathsf{bool}}$.
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).\overline{c_2}\,\mathsf{false}$
  Yes: $c_1 : \mathsf{lin}\,!\,\mathsf{bool}\,.\,\mathsf{lin}\,?\,\mathsf{bool}\,, c_2 : \mathsf{lin}\,?\,\mathsf{bool}\,.\,\mathsf{lin}\,!\,\mathsf{bool} = \overline{\mathsf{lin}\,!\,\mathsf{bool}\,.\,\mathsf{lin}\,?\,\mathsf{bool}}$.
- $\overline{x_1}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false}$
  No: $x_1 : q!\,\mathsf{bool}\,, x_2 : q!\,\mathsf{bool} \neq \overline{q!\,\mathsf{bool}}$.
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).c_2(t)$

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Should we accept these processes?

- $\overline{x_1}\,\mathsf{true} \mid x_2(z)$
  Yes: $x_1 : q!\,\mathsf{bool}\,, x_2 : q?\,\mathsf{bool} = \overline{q!\,\mathsf{bool}}$.
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).\overline{c_2}\,\mathsf{false}$
  Yes: $c_1 : \mathsf{lin}\,!\,\mathsf{bool}\,.\,\mathsf{lin}\,?\,\mathsf{bool}\,, c_2 : \mathsf{lin}\,?\,\mathsf{bool}\,.\,\mathsf{lin}\,!\,\mathsf{bool} = \overline{\mathsf{lin}\,!\,\mathsf{bool}\,.\,\mathsf{lin}\,?\,\mathsf{bool}}$.
- $\overline{x_1}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false}$
  No: $x_1 : q!\,\mathsf{bool}\,, x_2 : q!\,\mathsf{bool} \neq \overline{q!\,\mathsf{bool}}$.
- $\overline{c_1}\,\mathsf{true}\,.c_1(w) \mid c_2(z).c_2(t)$
  No: $c_1 : \mathsf{lin}\,!\,\mathsf{bool}\,.\,\mathsf{lin}\,?T, c_2 : \mathsf{lin}\,?\,\mathsf{bool}\,.\,\mathsf{lin}\,?U \neq \overline{\mathsf{lin}\,!\,\mathsf{bool}\,.\,\mathsf{lin}\,?T}$.

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Why not $\overline{q?T.U} = q!\overline{T}.\overline{U}$? Let's assume this (wrong) setting for this example

- $P = \overline{x_1}y_2 \mid x_2(z).\overline{z}\,\mathsf{true} \mid \overline{y_1}\,\mathsf{false}$
- Typed at context $x_1 : !(!\,\mathsf{bool}), x_2 : ?(?\,\mathsf{bool}), y_1 : !\,\mathsf{bool}\,, y_2 : !\,\mathsf{bool}$.
- Type of y_2 in send on x_1 is dual to type of z in receive on x_2: $\overline{!\,\mathsf{bool}} = ?\,\mathsf{bool}$.

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Why not $\overline{q?T.U} = q!\overline{T}.\overline{U}$? Let's assume this (wrong) setting for this example

- $P = \overline{x_1}y_2 \mid x_2(z).\overline{z}\,\mathsf{true} \mid \overline{y_1}\,\mathsf{false}$
- Typed at context $x_1 : !(!\,\mathsf{bool}), x_2 : ?(?\,\mathsf{bool}), y_1 : !\,\mathsf{bool}\,, y_2 : !\,\mathsf{bool}$.
- Type of y_2 in send on x_1 is dual to type of z in receive on x_2: $\overline{!\,\mathsf{bool}} = ?\,\mathsf{bool}$.
- $P \rightarrow \overline{y_2}\,\mathsf{true} \mid \overline{y_1}\,\mathsf{false}$: an illegal process!

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Duality is *not total*:

- Not defined on bool (or any similar "base" types).
- Only defined on session types: send, receive, and end.

Duality

Motto

The server's type is dual to the client's type.

$$\overline{q?T.U} = q!T.\overline{U} \qquad \overline{q!T.U} = q?T.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Duality is *not total*:

- Not defined on bool (or any similar "base" types).
- Only defined on session types: send, receive, and end.
- Suppose $\overline{\mathsf{bool}} = \mathsf{bool}$.
- We could type illegal process $(\nu xy)\,\mathsf{if}\ x\ \mathsf{then}\ 0\ \mathsf{else}\ 0$.

Type system

Invariants:

- Linear channel ends occur in *exactly one thread*.
- Co-variables have dual types.

Type system

Invariants:

- Linear channel ends occur in *exactly one thread*.
  Enforced by *context split*.
- Co-variables have dual types.

Type system

Invariants:

- Linear channel ends occur in *exactly one thread*.
  Enforced by *context split*.
- Co-variables have dual types.
  Enforced by typing rule for restriction.

Type system

Invariants:

- Linear channel ends occur in *exactly one thread*.
  Enforced by *context split*.
- Co-variables have dual types.
  Enforced by typing rule for restriction.

Qualifier predicates:

- $\mathrm{un}(T)$ iff $T =$ bool or $T =$ end or $T =$ un p.
- $\mathrm{lin}(T)$ iff true.
- $q(\Gamma)$ iff $(x : T) \in \Gamma$ implies $q(T)$.

Type system: context split and update

Context split ($\Gamma = \Gamma \circ \Gamma$):

$$\frac{}{\emptyset = \emptyset \circ \emptyset} \qquad \frac{\Gamma = \Gamma_1 \circ \Gamma_2 \qquad \mathrm{un}(T)}{\Gamma, x : T = (\Gamma_1, x : T) \circ (\Gamma_2, x : T)}$$

$$\frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x : \mathsf{lin}\, p = (\Gamma_1, x : \mathsf{lin}\, p) \circ \Gamma_2} \qquad \frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x : \mathsf{lin}\, p = \Gamma_1 \circ (\Gamma_2, x : \mathsf{lin}\, p)}$$

Context update ($\Gamma + x : T = \Gamma$):

$$\frac{x : U \notin \Gamma}{\Gamma + x : T = \Gamma, x : T} \qquad \frac{\mathrm{un}(T)}{(\Gamma, x : T) + x : T = (\Gamma, x : T)}$$

Type system: typing rules for values

Typing rules for values ($\Gamma \vdash v : T$):

$$\text{[T-True]}\quad \frac{\text{un}(\Gamma)}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}$$

$$\text{[T-False]}\quad \frac{\text{un}(\Gamma)}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}$$

$$\text{[T-Var]}\quad \frac{\text{un}(\Gamma)}{\Gamma, x : T \vdash x : T}$$

Type system: typing rules for processes

Typing rules for processes ($\Gamma \vdash P$):

[T-INACT]
$$\frac{\mathrm{un}(\Gamma)}{\Gamma \vdash 0}$$

[T-PAR]
$$\frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 \circ \Gamma_2 \vdash P \mid Q}$$

[T-RES]
$$\frac{\Gamma, x : S, y : \overline{S} \vdash P}{\Gamma \vdash (\nu xy)P}$$

[T-IF]
$$\frac{\Gamma_1 \vdash v : \mathsf{bool} \qquad \Gamma_2 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 \circ \Gamma_2 \vdash \mathsf{if}\ v\ \mathsf{then}\ P\ \mathsf{else}\ Q}$$

[T-RECV]
$$\frac{\Gamma_1 \vdash x : q?T.S \qquad (\Gamma_2 + x : S), y : T \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash x(y).P}$$

[T-SEND]
$$\frac{\Gamma_1 \vdash x : q!T.S \qquad \Gamma_2 \vdash v : T \qquad \Gamma_3 + x : S \vdash P}{\Gamma_1 \circ \Gamma_2 \circ \Gamma_3 \vdash \overline{x}v.P}$$

Type system: some thoughts

- Cannot use unrestricted channels (*yet*).
- Deadlock is possible, even when well typed.

Type system: some thoughts

- Cannot use unrestricted channels (*yet*).
  To type $\overline{x}\,\mathsf{true} \mid \overline{x}\,\mathsf{true}$, we seek context with $x : \mathsf{un}\,!\,\mathsf{bool}\,.T$. But rule [T-SEND] requires $(x : \mathsf{un}\,!\,\mathsf{bool}\,.T) + (x : T)$: impossible!
- Deadlock is possible, even when well typed.

Type system: some thoughts

- Cannot use unrestricted channels (*yet*).
- Deadlock is possible, even when well typed.
  $\overline{x_1}\,\mathsf{true}\,.\overline{y_1}\,\mathsf{false} \mid y_2(x).x_2(w)$

Type system: some thoughts

- Cannot use unrestricted channels (*yet*).
- Deadlock is possible, even when well typed.
  $\overline{x_1}\,\mathsf{true}\,.\overline{y_1}\,\mathsf{false} \mid y_2(x).x_2(w)$
  $\overline{x_1}y_1 \mid x_2(z).\overline{z}\,\mathsf{true}\,.y_2(w)$

Recursive types

New syntactic forms:

Types:

$T ::= \ldots$

a	type variable
$\mu a.T$	recursive type

- μ is a binder, giving rise to bound and free type variables, and alpha equivalence.
- Capture-avoiding substitution of a by U in T: $T[U/a]$.
- Two types are equal if their *infinite unfoldings* are syntactically equal: $\mu a.T \equiv T[\mu a.T/a]$ until the type does not start with μ.
- Therefore, $q(\mu a.T) = q(T)$.

New duality rules:

$$\overline{\mu a.T} = \mu a.\overline{T} \qquad \overline{a} = a$$

Recursive types: example

- Now we can derive

$$x_2 : \mathsf{un}\,?(!\,\mathsf{bool}).T \vdash x_2(z).\overline{z}\,\mathsf{true} \mid x_2(w).\overline{w}\,\mathsf{false}\,.$$

- For which T?

Recursive types: example

- Now we can derive

$$x_2 : \mathsf{un}\,?(!\,\mathsf{bool}).T \vdash x_2(z).\overline{z}\,\mathsf{true} \mid x_2(w).\overline{w}\,\mathsf{false}\,.$$

- For which T?
- For example, $T \equiv \mu a.\,\mathsf{un}\,?(!\,\mathsf{bool}).a$.

Recursive types: example

- Now we can derive

$$x_2 : \mathsf{un}\ ?(!\,\mathsf{bool}).T \vdash x_2(z).\overline{z}\,\mathsf{true} \mid x_2(w).\overline{w}\,\mathsf{false}\,.$$

- For which T?
- For example, $T \equiv \mu a.\,\mathsf{un}\ ?(!\,\mathsf{bool}).a$.
- Abbreviation for this form of type: $*U \triangleq \mu a.U.a$.

Tuples

- No primitive tuple passing.
- For linear x:
  - $\overline{x}\langle u, v\rangle.P \triangleq \overline{x}u.\overline{x}v.P$
  - Given $u : T, v : U$, typable $x : !T.!U$.
- For unrestricted x_1, x_2:
  - $\overline{x_1}\langle u, v\rangle.P \triangleq (\nu y_1 y_2)\overline{x_1}y_2.\overline{y_1}u.\overline{y_1}.v.P$
    $x_2(w, t).P \triangleq x_2(z).z(w).z(t).P$
  - y_1 linear, typed $y_1 : !T.!U$, and y_2 dually.
  - Then $x_1 : *!(?T.?U)$, and x_2 dually.
  - $*!\langle T, U\rangle \triangleq *!(?T.?U)$ and $*?\langle T, U\rangle \triangleq \overline{*!\langle T, U\rangle}$.

Tuples: example

- We own $p_2 : !\,\mathsf{bool}\,.!\,\mathsf{bool}\,.?\,\mathsf{bool}$.
- Want to delegate sending to another process, then read the result.

Tuples: example

- We own $p_2 : \,!\,\mathsf{bool}\,.!\,\mathsf{bool}\,.?\,\mathsf{bool}$.
- Want to delegate sending to another process, then read the result.
- Writer: $p_1(z, w).\overline{z}\,\mathsf{true}\,.\overline{z}\,\mathsf{true}\,.\overline{w}z$.

Tuples: example

- We own $p_2 : \,!\,\mathsf{bool}\,.!\,\mathsf{bool}\,.?\,\mathsf{bool}$.
- Want to delegate sending to another process, then read the result.
- Writer: $p_1(z, w).\overline{z}\,\mathsf{true}\,.\overline{z}\,\mathsf{true}\,.\overline{w}z$.
- Reader: $(\nu x_1 x_2)\overline{p_2}\langle c, x_1\rangle.x_2(z).z(y)$.

Tuples: example

- We own $p_2 : !\,\mathsf{bool}\,.!\,\mathsf{bool}\,.?\,\mathsf{bool}$.
- Want to delegate sending to another process, then read the result.
- Writer: $p_1(z, w).\overline{z}\,\mathsf{true}\,.\overline{z}\,\mathsf{true}\,.\overline{w}z$.
- Reader: $(\nu x_1 x_2)\overline{p_2}\langle c, x_1\rangle.x_2(z).z(y)$.
- $p_1 : *?\langle !\,\mathsf{bool}\,.!\,\mathsf{bool}\,.?\,\mathsf{bool}\,, !(?\,\mathsf{bool})\rangle$.

Recursive types: more examples

- $T = \,!\,\mathsf{bool}\,.\,*\,?\,\mathsf{bool}$ and $x_1 : T, x_2 : \overline{T}$. Are the following processes well typed?
  - $\overline{x_1}\,\mathsf{true}\,.(x_1(y) \mid x_1(z)) \mid x_2(x).(\overline{x_2}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false} \mid \overline{x_2}\,\mathsf{true})$
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y).x_1(y) \mid x_2(z)$
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y) \mid x_2(y).\overline{x_2}\,\mathsf{true} \mid x_2(w).\overline{x_2}\,\mathsf{true}$

Recursive types: more examples

- $T = \,!\,\mathsf{bool}\,.\,*\,?\,\mathsf{bool}$ and $x_1 : T, x_2 : \overline{T}$. Are the following processes well typed?
  - $\overline{x_1}\,\mathsf{true}\,.(x_1(y) \mid x_1(z)) \mid x_2(x).(\overline{x_2}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false} \mid \overline{x_2}\,\mathsf{true})$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y).x_1(y) \mid x_2(z)$
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y) \mid x_2(y).\overline{x_2}\,\mathsf{true} \mid x_2(w).\overline{x_2}\,\mathsf{true}$

Recursive types: more examples

- $T = \,!\,\mathsf{bool}\,.\,*\,?\,\mathsf{bool}$ and $x_1 : T, x_2 : \overline{T}$. Are the following processes well typed?
  - $\overline{x_1}\,\mathsf{true}\,.(x_1(y) \mid x_1(z)) \mid x_2(x).(\overline{x_2}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false} \mid \overline{x_2}\,\mathsf{true})$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y).x_1(y) \mid x_2(z)$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y) \mid x_2(y).\overline{x_2}\,\mathsf{true} \mid x_2(w).\overline{x_2}\,\mathsf{true}$

Recursive types: more examples

- $T = \,!\,\mathsf{bool}\,.\,*\,?\,\mathsf{bool}$ and $x_1 : T, x_2 : \overline{T}$. Are the following processes well typed?
  - $\overline{x_1}\,\mathsf{true}\,.(x_1(y) \mid x_1(z)) \mid x_2(x).(\overline{x_2}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false} \mid \overline{x_2}\,\mathsf{true})$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y).x_1(y) \mid x_2(z)$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y) \mid x_2(y).\overline{x_2}\,\mathsf{true} \mid x_2(w).\overline{x_2}\,\mathsf{true}$
    Not well typed: x_2 is not used linearly for the first receive!

Recursive types: more examples

- $T = \,!\,\mathsf{bool}\,.\,*\,?\,\mathsf{bool}$ and $x_1 : T, x_2 : \overline{T}$. Are the following processes well typed?
  - $\overline{x_1}\,\mathsf{true}\,.(x_1(y) \mid x_1(z)) \mid x_2(x).(\overline{x_2}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false} \mid \overline{x_2}\,\mathsf{true})$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y).x_1(y) \mid x_2(z)$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y) \mid x_2(y).\overline{x_2}\,\mathsf{true} \mid x_2(w).\overline{x_2}\,\mathsf{true}$
    Not well typed: x_2 is not used linearly for the first receive!
- Channel for both reading and writing?

Recursive types: more examples

- $T = \,!\,\mathsf{bool}\,.\,*\,?\,\mathsf{bool}$ and $x_1 : T, x_2 : \overline{T}$. Are the following processes well typed?
  - $\overline{x_1}\,\mathsf{true}\,.(x_1(y) \mid x_1(z)) \mid x_2(x).(\overline{x_2}\,\mathsf{true} \mid \overline{x_2}\,\mathsf{false} \mid \overline{x_2}\,\mathsf{true})$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y).x_1(y) \mid x_2(z)$
    Well typed.
  - $\overline{x_1}\,\mathsf{true}\,.x_1(y) \mid x_2(y).\overline{x_2}\,\mathsf{true} \mid x_2(w).\overline{x_2}\,\mathsf{true}$
    Not well typed: x_2 is not used linearly for the first receive!
- Channel for both reading and writing?
  Use tuples to pass both ends of a channel!

$$a_1 : *!\langle !\,\mathsf{bool}\,, ?\,\mathsf{bool}\rangle, a_2 : *?\langle !\,\mathsf{bool}\,, ?\,\mathsf{bool}\rangle$$
$$\vdash a_2(y_1, y_2).(\overline{y_1}\,\mathsf{false} \mid y_2(z)) \mid (\nu x_1 x_2)\overline{a_1}\langle x_1, x_2\rangle$$

Replication

Despite recursive types, processes so far are *strongly normalizing*.

New syntactic forms:

Processes:

$$P ::= \ldots$$
$$q x(y).P \quad \text{receive}$$

We have lin $x(y).P$ and un $x(y).P$.

New reduction rules:

[R-LinCom]

$$\frac{}{(\nu xy)(\overline{x}v.P \mid \text{lin}\, y(z).Q \mid R) \rightarrow (\nu xy)(P \mid Q[v/z] \mid R)}$$

[R-UnCom]

$$\frac{}{(\nu xy)(\overline{x}v.P \mid \text{un}\, y(z).Q \mid R) \rightarrow (\nu xy)(P \mid Q[v/z] \mid \text{un}\, y(z).Q \mid R)}$$

Replication: typing

New typing rules:

$$\frac{q_1(\Gamma_1 \circ \Gamma_2) \qquad \Gamma_1 \vdash x : q_2?T.S \qquad (\Gamma_2 + x : S), y : T \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash q_1 x(y).P}\ [\text{T-Recv}]$$

- Same as before when $q_1 = \text{lin}$: $\text{lin}(\Gamma)$ for all Γ.
- Not necessarily $q_1 = q_2$, but $q_2 = \text{un}$ implies $q_1 = \text{un}$.

Replication: typing

New typing rules:

$$\text{[T-RECV]}\ \frac{q_1(\Gamma_1 \circ \Gamma_2) \qquad \Gamma_1 \vdash x : q_2?T.S \qquad (\Gamma_2 + x : S), y : T \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash q_1 x(y).P}$$

- Same as before when $q_1 =$ lin: $\mathrm{lin}(\Gamma)$ for all Γ.
- Not necessarily $q_1 = q_2$, but $q_2 =$ un implies $q_1 =$ un.
- un $x_2(z).\overline{c}$ true $|$ $\overline{x_1}$ true $|$ $\overline{x_1}$ false
  Assume c : lin ! bool. Is it well-typed?

Replication: typing

New typing rules:

$$\frac{q_1(\Gamma_1 \circ \Gamma_2) \qquad \Gamma_1 \vdash x : q_2?T.S \qquad (\Gamma_2 + x : S), y : T \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash q_1 x(y).P}\ \text{[T-Recv]}$$

- Same as before when $q_1 = \text{lin}$: $\text{lin}(\Gamma)$ for all Γ.
- Not necessarily $q_1 = q_2$, but $q_2 = \text{un}$ implies $q_1 = \text{un}$.
- $\text{un}\, x_2(z).\overline{c}\,\text{true} \mid \overline{x_1}\,\text{true} \mid \overline{x_1}\,\text{false}$
  Assume $c : \text{lin}\,!\,\text{bool}$. Is it well-typed? No, linear channel c is replicated!

$$\begin{aligned}
&(\nu x_1 x_2)(\text{un}\, x_2(z).\overline{c}\,\text{true} \mid \overline{x_1}\,\text{true} \mid \overline{x_1}\,\text{false}) \\
\to\ &(\nu x_1 x_2)(\text{un}\, x_2(z).\overline{c}\,\text{true} \mid \overline{c}\,\text{true} \mid \overline{x_1}\,\text{false}) \\
\to\ &(\nu x_1 x_2)(\text{un}\, x_2(z).\overline{c}\,\text{true} \mid \overline{c}\,\text{true} \mid \overline{c}\,\text{true})
\end{aligned}$$

Replication: typing

New typing rules:

$$\text{[T-RECV]}\quad \frac{q_1(\Gamma_1 \circ \Gamma_2) \qquad \Gamma_1 \vdash x : q_2?T.S \qquad (\Gamma_2 + x : S), y : T \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash q_1 x(y).P}$$

- Same as before when $q_1 = \mathsf{lin}$: $\text{lin}(\Gamma)$ for all Γ.
- Not necessarily $q_1 = q_2$, but $q_2 = \mathsf{un}$ implies $q_1 = \mathsf{un}$.
- $\mathsf{un}\, x_2(z).\overline{c}\,\mathsf{true} \mid \overline{x_1}\,\mathsf{true} \mid \overline{x_1}\,\mathsf{false}$
  Assume $c : \mathsf{lin}\,!\,\mathsf{bool}$. Is it well-typed? No, linear channel c is replicated!

$$\begin{aligned} & (\nu x_1 x_2)(\mathsf{un}\, x_2(z).\overline{c}\,\mathsf{true} \mid \overline{x_1}\,\mathsf{true} \mid \overline{x_1}\,\mathsf{false}) \\ \to\ & (\nu x_1 x_2)(\mathsf{un}\, x_2(z).\overline{c}\,\mathsf{true} \mid \overline{c}\,\mathsf{true} \mid \overline{x_1}\,\mathsf{false}) \\ \to\ & (\nu x_1 x_2)(\mathsf{un}\, x_2(z).\overline{c}\,\mathsf{true} \mid \overline{c}\,\mathsf{true} \mid \overline{c}\,\mathsf{true}) \end{aligned}$$

Well typed: $\mathsf{un}\, x_2(z).\overline{z}\,\mathsf{true}$. Cannot be used by $\overline{x_1}c \mid \overline{x_1}c$.

General replication

- Milner's original, more general replication: $!P = P \mid P \mid \ldots$
- How can this be simulated?

General replication

- Milner's original, more general replication: $!P = P \mid P \mid \ldots$
- How can this be simulated?

$$!P \triangleq (\nu x_1 x_2)(\overline{x_1} x_2 \mid \mathsf{un}\, x_2(y).(P \mid \overline{x_1} y))$$

where $x_1, x_2, y \notin \mathrm{fv}(P)$.

General replication

- Milner's original, more general replication: $!P = P \mid P \mid \dots$
- How can this be simulated?

$$!P \triangleq (\nu x_1 x_2)(\overline{x_1} x_2 \mid \textsf{un}\, x_2(y).(P \mid \overline{x_1} y))$$

  where $x_1, x_2, y \notin \mathrm{fv}(P)$.
- Admissible typing rule:

$$\frac{\mathrm{un}(\Gamma) \qquad \Gamma \vdash P}{\Gamma \vdash !P}$$

  where $x_1 : \mu a.\, \textsf{un}\, !a.a$ and $x_2 : \mu b.\, \textsf{un}\, ?b.b$.

Choice

New syntactic forms:

Processes:

$$
\begin{array}{lll}
P ::= & \ldots & \\
& x \triangleleft l.P & \text{selection} \\
& x \triangleright \{i : P_i\}_{i \in I} & \text{branching}
\end{array}
$$

New reduction rules:

$$
[\textsc{R-Case}] \quad \frac{j \in I}{(\nu xy)(x \triangleleft j.P \mid y \triangleright \{i : Q_i\}_{i \in I} \mid R) \to (\nu xy)(P \mid Q_j \mid R)}
$$

Choice: typing

New syntactic forms:

Pretypes:

$$\begin{array}{rll} p ::= & \dots & \\ & \oplus\{i : S_i\}_{i \in I} & \text{select} \\ & \&\{i : S_i\}_{i \in I} & \text{branch} \end{array}$$

New duality rules:

$$\overline{q\oplus\{i : S_i\}_{i \in I}} = q\&\{i : \overline{S_i}\}_{i \in I} \qquad \overline{q\&\{i : S_i\}_{i \in I}} = q\oplus\{i : \overline{S_i}\}_{i \in I}$$

New typing rules:

$$\text{[T-BRANCH]}\ \frac{\Gamma_1 \vdash x : q\&\{i : S_i\}_{i \in I} \qquad \forall i \in I : \Gamma_2 + x : S_i \vdash P_i}{\Gamma_1 \circ \Gamma_2 \vdash x \triangleright \{i : P_i\}_{i \in I}}$$

$$\text{[T-SEL]}\ \frac{\Gamma_1 \vdash x : q\oplus\{i : S_i\}_{i \in I} \qquad \Gamma_2 + x : S_j \vdash P \qquad j \in I}{\Gamma_1 \circ \Gamma_2 \vdash x \triangleleft j.P}$$

Choice: examples

Well typed?

- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0\}$
- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0, m : 0\}$
- $x_1 \triangleleft l \mid x_1 \triangleleft m \mid x_1 \triangleleft m \mid x_2 \triangleright \{l : 0, m : 0\}$
- $\overline{x_1}\,\mathsf{true} \mid x_2 \triangleright \{l : 0\}$
- $x_1 \triangleleft l \mid x_2(z)$
- $x_1 \triangleleft l \mid x_2 \triangleright \{m : 0\}$

Choice: examples

Well typed?

- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0\}$
  Yes: $x_1 : q\oplus\{l : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0, m : 0\}$
- $x_1 \triangleleft l \mid x_1 \triangleleft m \mid x_1 \triangleleft m \mid x_2 \triangleright \{l : 0, m : 0\}$
- $\overline{x_1}\,\mathsf{true} \mid x_2 \triangleright \{l : 0\}$
- $x_1 \triangleleft l \mid x_2(z)$
- $x_1 \triangleleft l \mid x_2 \triangleright \{m : 0\}$

Choice: examples

Well typed?

- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0\}$
  Yes: $x_1 : q\oplus\{l : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes: $x_1 : q\oplus\{l : \mathsf{end}\,, m : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\,, m : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_1 \triangleleft m \mid x_1 \triangleleft m \mid x_2 \triangleright \{l : 0, m : 0\}$
- $\overline{x_1}\,\mathsf{true} \mid x_2 \triangleright \{l : 0\}$
- $x_1 \triangleleft l \mid x_2(z)$
- $x_1 \triangleleft l \mid x_2 \triangleright \{m : 0\}$

Choice: examples

Well typed?

- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0\}$
  Yes: $x_1 : q\oplus\{l : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes: $x_1 : q\oplus\{l : \mathsf{end}\,, m : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\,, m : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_1 \triangleleft m \mid x_1 \triangleleft m \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes:
  $x_1 : \mu a.\,\mathsf{un} \oplus\{l : a, m : a\} \triangleq *\oplus\{l, m\}, x_2 : \mu b.\,\mathsf{un}\,\&\{l : b, m : b\} \triangleq *\&\{l, m\}$.
- $\overline{x_1}\,\mathsf{true} \mid x_2 \triangleright \{l : 0\}$
- $x_1 \triangleleft l \mid x_2(z)$
- $x_1 \triangleleft l \mid x_2 \triangleright \{m : 0\}$

Choice: examples

Well typed?

- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0\}$
  Yes: $x_1 : q \oplus \{l : \mathsf{end}\}, x_2 : q \& \{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes: $x_1 : q \oplus \{l : \mathsf{end}\, , m : \mathsf{end}\}, x_2 : q \& \{l : \mathsf{end}\, , m : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_1 \triangleleft m \mid x_1 \triangleleft m \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes:
  $x_1 : \mu a.\, \mathsf{un} \oplus \{l : a, m : a\} \triangleq * \oplus \{l, m\}, x_2 : \mu b.\, \mathsf{un}\, \& \{l : b, m : b\} \triangleq * \& \{l, m\}$.
- $\overline{x_1}\, \mathsf{true} \mid x_2 \triangleright \{l : 0\}$
  No: $x_1 : q!\, \mathsf{bool}\, , x_2 : q \& \{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2(z)$
- $x_1 \triangleleft l \mid x_2 \triangleright \{m : 0\}$

Choice: examples

Well typed?

- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0\}$
  Yes: $x_1 : q{\oplus}\{l : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes: $x_1 : q{\oplus}\{l : \mathsf{end}\,, m : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\,, m : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_1 \triangleleft m \mid x_1 \triangleleft m \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes:
  $x_1 : \mu a.\, \mathsf{un} \oplus\{l : a, m : a\} \triangleq *{\oplus}\{l, m\}, x_2 : \mu b.\, \mathsf{un}\,\&\{l : b, m : b\} \triangleq *\&\{l, m\}$.
- $\overline{x_1}\, \mathsf{true} \mid x_2 \triangleright \{l : 0\}$
  No: $x_1 : q!\,\mathsf{bool}\,, x_2 : q\&\{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2(z)$
  No: $x_1 : q{\oplus}\{l : \mathsf{end}\,, \ldots\}, x_2 : q?T$.
- $x_1 \triangleleft l \mid x_2 \triangleright \{m : 0\}$

Choice: examples

Well typed?

- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0\}$
  Yes: $x_1 : q\oplus\{l : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes: $x_1 : q\oplus\{l : \mathsf{end}\,, m : \mathsf{end}\}, x_2 : q\&\{l : \mathsf{end}\,, m : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_1 \triangleleft m \mid x_1 \triangleleft m \mid x_2 \triangleright \{l : 0, m : 0\}$
  Yes:
  $x_1 : \mu a.\, \mathsf{un} \oplus\{l : a, m : a\} \triangleq *\oplus\{l, m\}, x_2 : \mu b.\, \mathsf{un}\,\&\{l : b, m : b\} \triangleq *\&\{l, m\}$.
- $\overline{x_1}\, \mathsf{true} \mid x_2 \triangleright \{l : 0\}$
  No: $x_1 : q!\, \mathsf{bool}\,, x_2 : q\&\{l : \mathsf{end}\}$.
- $x_1 \triangleleft l \mid x_2(z)$
  No: $x_1 : q\oplus\{l : \mathsf{end}\,, \ldots\}, x_2 : q?T$.
- $x_1 \triangleleft l \mid x_2 \triangleright \{m : 0\}$
  No: $x_1 : q\oplus\{l : \mathsf{end}\,, \ldots\}, x_2 : q\&\{m : \mathsf{end}\}$.

Example: map

How to use a *map* operating on x_2?

- Put operation for key k and value v:

$$x_1 \triangleleft \mathit{put}.\overline{x_1}k.\overline{x_1}v$$

- Get operation for key k:

$$x_1 \triangleleft \mathit{get}.\overline{x_1}k.x_1 \triangleright \{\mathit{some} : x_1(y).P, \mathit{none} : Q\}$$

- What is the type of the client's side of the (linear) map?

Example: map

How to use a *map* operating on x_2?

- Put operation for key k and value v:

$$x_1 \triangleleft \textit{put}.\overline{x_1}k.\overline{x_1}v$$

- Get operation for key k:

$$x_1 \triangleleft \textit{get}.\overline{x_1}k.x_1 \triangleright \{\textit{some} : x_1(y).P, \textit{none} : Q\}$$

- What is the type of the client's side of the (linear) map?
  $x_1 : \oplus\{\textit{put} : !\,\mathsf{key}\,.!\,\mathsf{value}\,.\,\mathsf{end}\,, \textit{get} : !\,\mathsf{key}\,.\&\{\textit{some} : ?\,\mathsf{value}\,.\,\mathsf{end}\,, \textit{none} : \mathsf{end}\}\}$

Example: iterator

- Iterator of booleans: offers at x_2 operations *hasNext* and *next*, until *hasNext* returns “no”.
- A client that reads and discards every value. Which is correct?
  - $!(\mathsf{un}\ \mathit{loop}_2(y).y \triangleleft \mathit{hasNext}.y \triangleright \{\mathit{yes} : y \triangleleft \mathit{next}.y(z).\overline{\mathit{loop}_1}\, y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}\, x_2$
  - $!(\mathsf{un}\ \mathit{loop}_2(y).x_2 \triangleleft \mathit{hasNext}.x_2 \triangleright \{\mathit{yes} : x_2 \triangleleft \mathit{next}.x_2(z).\overline{\mathit{loop}_1}\, y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}\ \mathsf{true}$

Example: iterator

- Iterator of booleans: offers at x_2 operations *hasNext* and *next*, until *hasNext* returns “no”.
- A client that reads and discards every value. Which is correct?
  - $!(\mathsf{un}\ \mathit{loop}_2(y).y \triangleleft \mathit{hasNext}.y \triangleright \{\mathit{yes} : y \triangleleft \mathit{next}.y(z).\overline{\mathit{loop}_1}y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}x_2$
    Yes.
  - $!(\mathsf{un}\ \mathit{loop}_2(y).x_2 \triangleleft \mathit{hasNext}.x_2 \triangleright \{\mathit{yes} : x_2 \triangleleft \mathit{next}.x_2(z).\overline{\mathit{loop}_1}y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}\,\mathsf{true}$
    No: x_2 is linear.

Example: iterator

- Iterator of booleans: offers at x_2 operations *hasNext* and *next*, until *hasNext* returns "no".
- A client that reads and discards every value. Which is correct?
  - $!(\text{un } \mathit{loop}_2(y).y \triangleleft \mathit{hasNext}.y \triangleright \{\mathit{yes} : y \triangleleft \mathit{next}.y(z).\overline{\mathit{loop}_1}y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}x_2$
    Yes.
  - $!(\text{un } \mathit{loop}_2(y).x_2 \triangleleft \mathit{hasNext}.x_2 \triangleright \{\mathit{yes} : x_2 \triangleleft \mathit{next}.x_2(z).\overline{\mathit{loop}_1}y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}\,\text{true}$
    No: x_2 is linear.
- Type of x_2 is linear, yet infinite:

$$\oplus\{\mathit{hasNext} : \&\{\mathit{no} : \text{end}, \mathit{yes} : \oplus\{\mathit{next} : !\,\text{bool}.\oplus\{\mathit{hasNext} : \&\{\ldots\}\}\}\}\}$$

Example: iterator

- Iterator of booleans: offers at x_2 operations *hasNext* and *next*, until *hasNext* returns "no".
- A client that reads and discards every value. Which is correct?
  - $!(\mathsf{un}\ \mathit{loop}_2(y).y \triangleleft \mathit{hasNext}.y \triangleright \{\mathit{yes} : y \triangleleft \mathit{next}.y(z).\overline{\mathit{loop}_1}y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}x_2$
    Yes.
  - $!(\mathsf{un}\ \mathit{loop}_2(y).x_2 \triangleleft \mathit{hasNext}.x_2 \triangleright \{\mathit{yes} : x_2 \triangleleft \mathit{next}.x_2(z).\overline{\mathit{loop}_1}y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}\ \mathsf{true}$
    No: x_2 is linear.
- Type of x_2 is linear, yet infinite:

$$\oplus\{\mathit{hasNext} : \&\{\mathit{no} : \mathsf{end}\,, \mathit{yes} : \oplus\{\mathit{next} : !\,\mathsf{bool}\,.\oplus\{\mathit{hasNext} : \&\{\dots\}\}\}\}\}$$

- Finite form?

Example: iterator

- Iterator of booleans: offers at x_2 operations *hasNext* and *next*, until *hasNext* returns "no".
- A client that reads and discards every value. Which is correct?
  - $!(\text{un } \mathit{loop}_2(y).y \triangleleft \mathit{hasNext}.y \triangleright \{\mathit{yes} : y \triangleleft \mathit{next}.y(z).\overline{\mathit{loop}_1}y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}x_2$
    Yes.
  - $!(\text{un } \mathit{loop}_2(y).x_2 \triangleleft \mathit{hasNext}.x_2 \triangleright \{\mathit{yes} : x_2 \triangleleft \mathit{next}.x_2(z).\overline{\mathit{loop}_1}y, \mathit{no} : 0\}) \mid \overline{\mathit{loop}_1}\,\text{true}$
    No: x_2 is linear.
- Type of x_2 is linear, yet infinite:

$$\oplus\{\mathit{hasNext} : \&\{\mathit{no} : \text{end}\,, \mathit{yes} : \oplus\{\mathit{next} : !\,\text{bool}\,.\oplus\{\mathit{hasNext} : \&\{\ldots\}\}\}\}\}$$

- Finite form?

$$\mu a.\oplus\{\mathit{hasNext} : \&\{\mathit{no} : \text{end}\,, \mathit{yes} : \oplus\{\mathit{next} : !\,\text{bool}\,.a\}\}\}$$

Primitive types are redundant

- With choice, primitive types are redundant.
- Example: booleans.

Primitive types are redundant

- With choice, primitive types are redundant.
- Example: booleans.
- $$\begin{aligned} \text{true} &\triangleq\ !(t_1 \triangleleft \textit{true}) \\ \text{false} &\triangleq\ !(f_1 \triangleleft \textit{false}) \\ \text{if } x \text{ then } P \text{ else } Q &\triangleq x \triangleright \{\textit{true} : P, \textit{false} : Q\} \end{aligned}$$
- $$\text{true} \mid \text{false} \mid \text{if } t_2 \text{ then } P \text{ else } Q \longrightarrow\longrightarrow \text{true} \mid \text{false} \mid P$$

Well typedness guarantees well formedness

- Patterns of ill formedness. Why?
  - if x then P else Q
  - $\overline{a}\,\mathsf{true} \mid a(z)$
  - $(\nu xy)(\overline{x}\,\mathsf{true} \mid y \triangleright \{i : P_i\}_{i \in I})$

Well typedness guarantees well formedness

- Patterns of ill formedness. Why?
  - if x then P else Q
    x is not a boolean value.
  - $\overline{a}\,\mathsf{true} \mid a(z)$
  - $(\nu xy)(\overline{x}\,\mathsf{true} \mid y \triangleright \{i : P_i\}_{i \in I})$

Well typedness guarantees well formedness

- Patterns of ill formedness. Why?
  - if x then P else Q
    x is not a boolean value.
  - $\overline{a}\,\mathsf{true} \mid a(z)$
    a is unrestricted, but used differently.
  - $(\nu xy)(\overline{x}\,\mathsf{true} \mid y \triangleright \{i : P_i\}_{i \in I})$

Well typedness guarantees well formedness

- Patterns of ill formedness. Why?
  - if x then P else Q
    x is not a boolean value.
  - $\overline{a}\,\mathsf{true} \mid a(z)$
    a is unrestricted, but used differently.
  - $(\nu xy)(\overline{x}\,\mathsf{true} \mid y \triangleright \{i : P_i\}_{i \in I})$
    x and y are not typed dually.

Well typedness guarantees well formedness

- Patterns of ill formedness. Why?
  - if x then P else Q
    x is not a boolean value.
  - $\overline{a}\,\mathsf{true} \mid a(z)$
    a is unrestricted, but used differently.
  - $(\nu xy)(\overline{x}\,\mathsf{true} \mid y \triangleright \{i : P_i\}_{i \in I})$
    x and y are not typed dually.
- Typing excludes ill formedness, even after reduction:

Theorem (Main result)

If $\emptyset \vdash P$ and $P \to^* Q$[2], then Q is well formed.

[2] $\to^*$ means zero or more steps.

Well typedness guarantees well formedness

- Patterns of ill formedness. Why?
  - if x then P else Q
    x is not a boolean value.
  - $\overline{a}\,\text{true} \mid a(z)$
    a is unrestricted, but used differently.
  - $(\nu xy)(\overline{x}\,\text{true} \mid y \triangleright \{i : P_i\}_{i \in I})$
    x and y are not typed dually.
- Typing excludes ill formedness, even after reduction:

Theorem (Main result)

If $\emptyset \vdash P$ and $P \rightarrow^* Q$, then Q is well formed.

- Follows from:

Theorem (Preservation)

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

Theorem (Safety)

If $\emptyset \vdash P$, then P is well formed.