

Ldoos

A Distributed Object-Oriented language with Session Types

joint work with Nobuko Yoshida,
Alexander Ahern,
and Sophia Drossopoulou

TGC'05

Symposium on Trustworthy Global Computing

Saturday, April 9 2005

Edinburgh, UK



Session Types

What will we talk about?

Session Types

What will we talk about?

a subject of conversation is not enough.

Session Types

What will we talk about?

a subject of conversation is not enough.

We want to specify

the sequence and the direction of data exchanged

Session Types

What will we talk about?

a subject of conversation is not enough.

We want to specify

the sequence and the direction of data exchanged

Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo.
Language primitives and type disciplines for structured
communication-based programming.

ESOP'98, LNCS1381, pages 22–138. Springer-Verlag, 1998.

Session Types

What will we talk about?

a subject of conversation is not enough.

We want to specify

the sequence and the direction of data exchanged

Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo.
Language primitives and type disciplines for structured
communication-based programming.

ESOP'98, LNCS1381, pages 22–138. Springer-Verlag, 1998.

Scenario

users at different locations interacting by means of
object-oriented code

examples

```
!int.!int.?bool.end
```

expresses that two `int`-values will be sent then a `bool`-value is expected as an input

examples

```
!int.!int.?bool.end
```

expresses that two `int`-values will be sent then a `bool`-value is expected as an input

```
!(<!int,?float>).end
```

sends a `bool`-value and then sends an `int` if the value is true or receives a `float` if the value is false

branching type

examples

```
!int.!int.?bool.end
```

expresses that two `int`-values will be sent then a `bool`-value is expected as an input

```
!(<!int,?float>).end
```

sends a `bool`-value and then sends an `int` if the value is true or receives a `float` if the value is false

branching type

```
!(<!int.?float>)*.end
```

sends a `bool`-value and if that value is true, continues with

```
!int.?float
```

if the value sent is false, this session finishes

recursive type

Syntax of session types

(direction)	\dagger	$::=$	$! \mid ?$
(part of session)	π	$::=$	$\varepsilon \mid \dagger t \mid \dagger \langle \pi, \pi \rangle \mid \dagger \langle \pi \rangle^* \mid \pi.\pi$
(session)	s	$::=$	$\pi.\mathbf{end}$

Syntax of session types

(direction) $\dagger ::= ! \mid ?$
(part of session) $\pi ::= \epsilon \mid \dagger t \mid \dagger \langle \pi, \pi \rangle \mid \dagger \langle \pi \rangle^* \mid \pi.\pi$
(session) $s ::= \pi.\mathbf{end}$

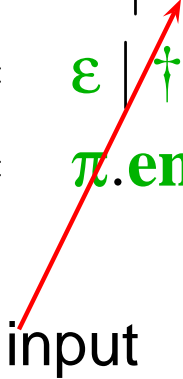
output



Syntax of session types

(direction)	\dagger	::=	! ?
(part of session)	π	::=	ε $\dagger t$ $\dagger \langle \pi, \pi \rangle$ $\dagger \langle \pi \rangle^*$ $\pi.\pi$
(session)	s	::=	$\pi.\mathbf{end}$

input



Syntax of session types

(direction) $\dagger ::= ! \mid ?$
(part of session) $\pi ::= \varepsilon \mid \dagger t \mid \dagger \langle \pi, \pi \rangle \mid \dagger \langle \pi \rangle^* \mid \pi.\pi$
(session) $s ::= \pi.\mathbf{end}$

no exchange



Syntax of session types

(direction) $\dagger ::= ! \mid ?$

(part of session) $\pi ::= \varepsilon \mid \dagger t \mid \dagger \langle \pi, \pi \rangle \mid \dagger \langle \pi \rangle^* \mid \pi.\pi$

(session) $s ::= \pi.\mathbf{end}$

exchange one value of type t

Syntax of session types

(direction) $\dagger ::= ! \mid ?$
(part of session) $\pi ::= \varepsilon \mid \dagger t \mid \dagger \langle \pi, \pi \rangle \mid \dagger \langle \pi \rangle^* \mid \pi.\pi$
(session) $s ::= \pi.\mathbf{end}$

branching



Syntax of session types

(direction) $\dagger ::= ! \mid ?$
(part of session) $\pi ::= \varepsilon \mid \dagger t \mid \dagger \langle \pi, \pi \rangle \mid \dagger \langle \pi \rangle^* \mid \pi.\pi$
(session) $s ::= \pi.\mathbf{end}$

recursion

Syntax of session types

(direction) $\dagger ::= ! \mid ?$
(part of session) $\pi ::= \varepsilon \mid \dagger t \mid \dagger \langle \pi, \pi \rangle \mid \dagger \langle \pi \rangle^* \mid \pi.\pi$
(session) $s ::= \pi.\mathbf{end}$

composition



Syntax of session types

(direction)	\dagger	$::=$	$! \mid ?$
(part of session)	π	$::=$	$\varepsilon \mid \dagger t \mid \dagger \langle \pi, \pi \rangle \mid \dagger \langle \pi \rangle^* \mid \pi.\pi$
(session)	s	$::=$	$\pi.\mathbf{end}$

Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.
What does its declaration look like?

Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.

What does its declaration look like?

```
void communicate ( )  c : !int.!int.?bool.end {...}
```

Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.

What does its declaration look like?

```
void communicate ( )  c : !int.!int.?bool.end {...}
```

$$tm \ (tx) \ \Sigma \ \{e\}$$

Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.

What does its declaration look like?

```
void communicate ( ) c : !int.!int.?bool.end {...}
```

$tm (tx) \Sigma \{ e \}$

return type



Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.

What does its declaration look like?

```
void communicate ( )  c : !int.!int.?bool.end {...}
```

$$tm (tx) \Sigma \{ e \}$$

method name

Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.

What does its declaration look like?

```
void communicate ( )  c : !int.!int.?bool.end {...}
```

`tm` (`tx`) Σ { `e` }

parameter type


Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.

What does its declaration look like?

```
void communicate ( ) c : !int.!int.?bool.end {...}
```

$tm (tx) \Sigma \{e\}$



parameter

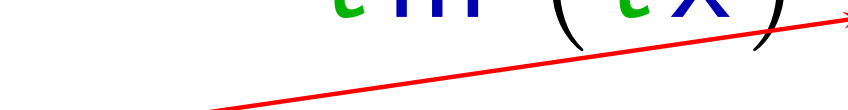
Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.

What does its declaration look like?

```
void communicate ( )  c : !int.!int.?bool.end {...}
```

$tm (tx) \Sigma \{e\}$



session environment mapping channels to session types

Session in Object-Oriented Paradigm

The method `communicate` uses the channel `c` to send two integers and receive a boolean.

What does its declaration look like?

```
void communicate ( )  c : !int.!int.?bool.end {...}
```

$$tm \ (tx) \ \Sigma \ \{e\}$$

method body



Session start

... **request** u_1 **s** $\{e_1\}$... || ... **accept** u_2 **s** $\{e_2\}$...

(this is a multi-threaded presentation of a distributed calculus)

Session start

... **request** u_1 **s** $\{e_1\}$... || ... **accept** u_2 **s** $\{e_2\}$...

if

u_1 and u_2 are the same channel name or

u_1 is a channel name and u_2 is a channel variable with value u_1 or

u_2 is a channel name and u_1 is a channel variable with value u_2 or

u_1 and u_2 are variables whose value is the same channel name

(this is a multi-threaded presentation of a distributed calculus)

Session start

... **request** u_1 s $\{e_1\}$... || ... **accept** u_2 s $\{e_2\}$...

if

u_1 and u_2 are the same channel name or

u_1 is a channel name and u_2 is a channel variable with value u_1 or

u_2 is a channel name and u_1 is a channel variable with value u_2 or

u_1 and u_2 are variables whose value is the same channel name



$(\nu c : s)(\dots e_1[c/u_1] \dots || \dots e_2[c/u_2] \dots)$

c fresh

(this is a multi-threaded presentation of a distributed calculus)

Communication

$$(\nu c:\dagger t .s)(\dots c.\mathbf{send}(v)\dots \parallel \dots c.\mathbf{receive}\dots)$$

Communication

$(\nu c:\dagger t . s)(\dots c.\text{send}(v) \dots \parallel \dots c.\text{receive} \dots)$



Communication

$(\mathbf{vc}:\dagger\mathbf{t}.\mathbf{s})(\dots\mathbf{c}.\mathbf{send}(v)\dots \parallel \dots\mathbf{c}.\mathbf{receive}\dots)$



$(\mathbf{vc}:\mathbf{s})(\dots\mathbf{null}\dots \parallel \dots\mathbf{v}\dots)$

Communication

$(\mathbf{v}c:\dagger t .s)(\dots c.\mathbf{send}(v)\dots \parallel \dots c.\mathbf{receive}\dots)$



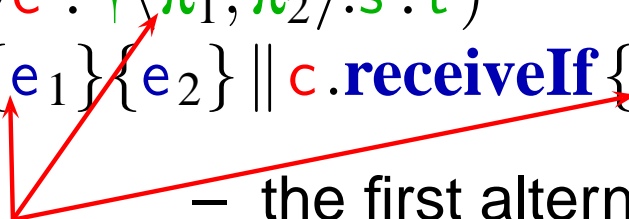
$(\mathbf{v}c:s)(\dots \mathbf{null}\dots \parallel \dots v\dots)$

$\dagger t$ is consumed

Choice

$$\begin{aligned} & (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots & c.\mathbf{sendIf}(\mathbf{true})\{e_1\}\{e_2\} \parallel c.\mathbf{receiveIf}\{e_3\}\{e_4\} \dots) \end{aligned}$$

Choice

$$(v c : \dagger \langle \pi_1, \pi_2 \rangle . s : t)$$
$$(\dots c.\mathbf{sendIf}(\mathbf{true})\{e_1\}\{e_2\} \parallel c.\mathbf{receiveIf}\{e_3\}\{e_4\} \dots)$$


true choices – the first alternative

Choice

$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{true})\{e_1\}\{e_2\} \parallel c.\mathbf{receiveIf}\{e_3\}\{e_4\} \dots) \end{array}$$


Choice

$$(v c : \dagger \langle \pi_1, \pi_2 \rangle . s : t)$$
$$(\dots c.\mathbf{sendIf}(\mathbf{true})\{e_1\}\{e_2\} \parallel c.\mathbf{receiveIf}\{e_3\}\{e_4\} \dots)$$

$$(v c : \pi_1 . s : t)(\dots e_1 \parallel e_3 \dots)$$

Choice

$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{true})\{e_1\}\{e_2\} \parallel c.\mathbf{receivelf}\{e_3\}\{e_4\} \dots) \end{array}$$

$$(\nu c : \pi_1 . s : t) (\dots e_1 \parallel e_3 \dots)$$
$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{false})\{e_1\}\{e_2\} \parallel c.\mathbf{receivelf}\{e_3\}\{e_4\} \dots) \end{array}$$

Choice

$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{true})\{e_1\}\{e_2\} \parallel c.\mathbf{receivelf}\{e_3\}\{e_4\} \dots) \end{array}$$

$$(\nu c : \pi_1 . s : t) (\dots e_1 \parallel e_3 \dots)$$
$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{false})\{e_1\}\{e_2\} \parallel c.\mathbf{receivelf}\{e_3\}\{e_4\} \dots) \end{array}$$

false choices the  – the second alternative

Choice

$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{true})\{e_1\}\{e_2\} \parallel c.\mathbf{receivelf}\{e_3\}\{e_4\} \dots) \end{array}$$

$$(\nu c : \pi_1 . s : t) (\dots e_1 \parallel e_3 \dots)$$
$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{false})\{e_1\}\{e_2\} \parallel c.\mathbf{receivelf}\{e_3\}\{e_4\} \dots) \end{array}$$


Choice

$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{true})\{e_1\}\{e_2\} \parallel c.\mathbf{receivelf}\{e_3\}\{e_4\} \dots) \end{array}$$

$$(\nu c : \pi_1 . s : t) (\dots e_1 \parallel e_3 \dots)$$
$$\begin{array}{c} (\nu c : \dagger \langle \pi_1, \pi_2 \rangle . s : t) \\ (\dots c.\mathbf{sendIf}(\mathbf{false})\{e_1\}\{e_2\} \parallel c.\mathbf{receivelf}\{e_3\}\{e_4\} \dots) \end{array}$$

$$(\nu c : \pi_2 . s : t) (\dots e_2 \parallel e_4 \dots)$$

Example: the Agent

```
class Agent extends Object {  
  
    float price, minPrice; // seller' asking and minimum price  
    float offer;          // the offer made by the buyer  
  
    void mediate() c1: ?float.?float.!<?float>*.end {  
        // connect with a seller  
        request c1 ?float.?float.!<?float>*.end {  
            price := c1.receive; minPrice := c1.receive;  
            c1.sendWhile ( needToContinue() )  
                // if the value of needToContinue() is true  
                { minPrice := c1.receive; } }  
  
        boolean needToContinue() c2: !float.?float.!boolean.end {  
            // connect with a buyer  
            request c2 : !float.?float.!boolean.end {  
                c2.send(price); offer := c2.receive; c2.send(offer < minPrice);  
                return (offer < minPrice); } }  
    }  
}
```

Example: the Agent

```
void mediate() c1: ?float.?float.!<?float>*.end
{
    // connect with a seller
    request c1 ?float.?float.!<?float>*.end {
        price := c1.receive;
        minPrice := c1.receive;
        c1.sendWhile ( needToContinue() )
            // if the value of
            needToContinue() is true
            { minPrice := c1.receive; } }}
}
```

Example: the Agent

```
boolean needToContinue()  
c2: !float.?float.!boolean.end {  
  // connect with a buyer  
  request c2 : !float.?float.!boolean.end {  
    c2.send(price);  
    offer := c2.receive;  
    c2.send(offer < minPrice);  
    return (offer < minPrice); } }  
}
```

Example: the Buyer

```
class Buyer extends Object {  
    float price; // seller's asking price  
    float offer; // offer made by the buyer  
  
    void buy() c2: !float.?float.!boolean.end {  
        accept c2 !float.?float.!boolean.end {  
            // connect with an agent  
            price := c2.receive; offer:=....;  
            c2.send(offer)  
            if c2.receive then .... else ... } }  
    }  
}
```

Example: the Seller

```
class Seller {  
  
    float price,minPrice; // asking price and minimum price  
  
    void sell( ) c1: ?float.?float.!<?float>*.end {  
        price:= ... ; minPrice:= ... ;  
        // connect to an agent  
        accept c1 ?float.?float.!<?float>*.end {  
            c1.send(price); c1.send(minPrice);  
            c1.receiveWhile  
                // if the value received is true, then  
                { minPrice:= ... ; c1.send(minPrice); } } }  
    }  
}
```

Example: the Seller

```
void sell( ) c1: ?float.?float.!<?float>*.end {  
    price:= ... ; minPrice:= ... ;  
    // connect to an agent  
    accept c1 ?float.?float.!<?float>*.end {  
        c1.send(price);  
        c1.send(minPrice);  
        c1.receiveWhile  
            // if the value received is true,  
            then  
            { minPrice:= ... ; c1.send(minPrice);  
            } } }  
}
```

Typing judgment

$$\Gamma; \Sigma \vdash e : t; \Sigma'$$

Typing judgment

$$\Gamma; \Sigma \vdash e : t; \Sigma'$$

environment



Typing judgment

$$\Gamma; \Sigma \vdash e : t ; \Sigma'$$

expression



Typing judgment

$$\Gamma; \Sigma \vdash e :: t; \Sigma'$$

type



Typing judgment

$$\Gamma; \Sigma \vdash e : t; \Sigma'$$

session types of channels before the evaluation of e

Typing judgment

$$\Gamma; \Sigma \vdash e : t; \Sigma'$$

session types of channels after the evaluation of e

Typing judgment

$$\Gamma; \Sigma \vdash e : t; \Sigma'$$

$$\Gamma; \Sigma, c : ?t . s \vdash c.\mathbf{receive} : t; \Sigma, c : s$$

Typing judgment

$$\Gamma; \Sigma \vdash e : t; \Sigma'$$

$$\Gamma; \Sigma, c : ?t . s \vdash c.\mathbf{receive} : t; \Sigma, c : s$$

?t is consumed



Subject Reduction

If $\Gamma; \Sigma \vdash e : t; \Sigma'$,
and $\Gamma \vdash \sigma : \text{ok}$,
and $\Gamma \vdash \text{CT} : \text{ok}$,
and $e, \sigma, \text{CT} \longrightarrow (\nu \vec{u} : \vec{t}') (e', \sigma', \text{CT})$

Subject Reduction

If $\Gamma; \Sigma \vdash e : t; \Sigma'$,
and $\Gamma \vdash \sigma : \text{ok}$,
and $\Gamma \vdash \text{CT} : \text{ok}$,
and $e, \sigma, \text{CT} \longrightarrow (\nu \vec{u} : \vec{t}') (e', \sigma', \text{CT})$
then
 $\Gamma, \vec{u} : \vec{t}'; \Sigma \vdash e' : t'; \Sigma'$,
with $t' <: t$
and $\Gamma, \vec{u} : \vec{t}' \vdash \sigma' : \text{ok}$.

Soundness

- no **connection error** will occur, *i.e.* request and accept on the same channel must have the same session type;

Soundness

- no **connection error** will occur, *i.e.* request and accept on the same channel must have the same session type;

... **request** $c\ s\ \{e_1\}$ || **accept** $c\ s'\ \{e_2\}$... \longrightarrow ConnErr

Soundness

- no **connection error** will occur, *i.e.* request and accept on the same channel must have the same session type;
- no **communication error** will occur, *i.e.* in the same net there will not be two sends or two receives on the same channel;

Soundness

- no **connection error** will occur, *i.e.* request and accept on the same channel must have the same session type;
- no **communication error** will occur, *i.e.* in the same net there will not be two sends or two receives on the same channel;

... **c.send**(v) || **c.send**(v') ... \longrightarrow CommErr

... **c.receive** || **c.receive** ... \longrightarrow CommErr

Soundness

- no **connection error** will occur, *i.e.* request and accept on the same channel must have the same session type;
- no **communication error** will occur, *i.e.* in the same net there will not be two sends or two receives on the same channel;
- after a session has begun **the required communications are always executed in the expected order**;

Soundness

- no **connection error** will occur, *i.e.* request and accept on the same channel must have the same session type;
- no **communication error** will occur, *i.e.* in the same net there will not be two sends or two receives on the same channel;
- after a session has begun **the required communications are always executed in the expected order**;
- after a session has begun **all the required communications are executed** unless one of the following situations occurs:
 - a null pointer exception is thrown;
 - the computation diverges; or
 - there is a request or accept instruction waiting for the dual instruction.

Future Work

- **part of sessions as method parameters** - to delegate part of the session ;

Future Work

- part of sessions as method parameters; **IN PROGRESS.**

Future Work

- part of sessions as method parameters; **IN PROGRESS**.
- communication of running channels - to delegate part of the session ;

Future Work

- part of sessions as method parameters; IN PROGRESS.
- communication of running channels; IN PROGRESS.

Future Work

- part of sessions as method parameters; IN PROGRESS.
- communication of running channels; IN PROGRESS.
- private channels in user syntax, to guarantee privacy of communications.

Future Work

- part of sessions as method parameters; IN PROGRESS.
- communication of running channels; IN PROGRESS.
- private channels in user syntax, to guarantee privacy of communications.

Alternative Design

- local types for channels, and run time type-check.

Future Work

- part of sessions as method parameters; IN PROGRESS.
- communication of running channels; IN PROGRESS.
- private channels in user syntax, to guarantee privacy of communications.

Alternative Design

- local types for channels, and run time type-check.
- add explicit selection, rather than use method names.

Future Work

- part of sessions as method parameters; **IN PROGRESS.**
- communication of running channels; **IN PROGRESS.**
- private channels in user syntax, to guarantee privacy of communications.

Alternative Design

- local types for channels, and run time type-check.
- add explicit selection, rather than use method names.
- use objects *instead of channels*, to have a smaller language.

Conclusion



Conclusion



thank you for your attention

No explicit selection

- calculi for sessions have explicit selection based on labels

No explicit selection

- calculi for sessions have explicit selection based on labels

$$(c \triangleleft l_i; P) \parallel (c \triangleright \{l_1 : P_1 \square \dots \square l_n : P_n\})$$



$$P \parallel P_i$$

No explicit selection

- calculi for sessions have explicit selection based on labels

$$(\mathbf{c} \triangleleft l_i; P) \parallel (\mathbf{c} \triangleright \{l_1 : P_1 \square \dots \square l_n : P_n\})$$



$$P \parallel P_i$$

- we use methods names instead!

Global versus Local Types

the session types of channels are **global**

the session types of channels are **local**

Global versus Local Types

the session types of channels are **global**

- in favour: request and accept on the same channel must have the same session type (no *connection error* can occur)

the session types of channels are **local**

Global versus Local Types

the session types of channels are **global**

- in favour: request and accept on the same channel must have the same session type (no *connection error* can occur)
- against: possibly realistic in distributed settings

the session types of channels are **local**

Global versus Local Types

the session types of channels are **global**

- in favour: request and accept on the same channel must have the same session type (no *connection error* can occur)
- against: possibly realistic in distributed settings

the session types of channels are **local**

- in favour: it fits nicely the distributed scenario

Global versus Local Types

the session types of channels are **global**

- in favour: request and accept on the same channel must have the same session type (no *connection error* can occur)
- against: possibly realistic in distributed settings

the session types of channels are **local**

- in favour: it fits nicely the distributed scenario
- against: it requires a run-time type check

Loop

$$\begin{array}{c} (\nu c : \dagger \langle \pi \rangle^* . s : t) \\ (\dots c.\text{sendWhile}(e)\{e_1\} \parallel c.\text{receiveWhile}\{e_2\} \dots) \end{array}$$

Loop

$$(v c : \dagger \langle \pi \rangle^* . s : t)$$
$$(\dots c.\text{sendWhile}(e)\{e_1\} \parallel c.\text{receiveWhile}\{e_2\} \dots)$$


Loop

$$(\nu c : \dagger \langle \pi \rangle^* . s : t)$$
$$(\dots c.\mathbf{sendWhile}(e)\{e_1\} \parallel c.\mathbf{receiveWhile}\{e_2\} \dots)$$

$$(\nu c : \dagger \langle \pi . \dagger \langle \pi \rangle^* , \epsilon \rangle . s : t)$$
$$(\dots c.\mathbf{sendIf}(e)\{e_1; c.\mathbf{sendWhile}(e)\{e_1\}\}\{\mathbf{null}\} \parallel$$
$$c.\mathbf{receivelf}\{e_2; c.\mathbf{receiveWhile}\{e_2\}\}\{\mathbf{null}\} \dots)$$

Typing Communication

$$\Gamma; \Sigma, c : ?t . s \vdash c.\mathbf{receive} : t; \Sigma, c : s$$
$$\Gamma; \Sigma \vdash e : t; \Sigma', c : !t . s$$

$$\Gamma; \Sigma \vdash c.\mathbf{send}(e) : \mathit{Object}; \Sigma', c : s$$

Typing Session Opening

$$\frac{\Gamma, u : s; \Sigma, c : s \vdash e[c/u] : t; \Sigma', c : \mathbf{end} \quad c \notin \text{fn}(e) \quad c \notin \text{dom}(\Gamma)}{\Gamma, u : s; \Sigma \vdash \mathbf{request} \ u \ s \{e\} : t; \Sigma'}$$

$$\frac{\Gamma, u : s; \Sigma, c : \bar{s} \vdash e[c/u] : t; \Sigma', c : \mathbf{end} \quad c \notin \text{fn}(e) \quad c \notin \text{dom}(\Gamma)}{\Gamma, u : s; \Sigma \vdash \mathbf{accept} \ u \ s \{e\} : t; \Sigma'}$$

Method Declaration ok

$$\frac{\Sigma, \text{this} : C, x : t_1; \emptyset \vdash e : t; \emptyset}{\Gamma, \text{this} : C \vdash t_2 m(t_1 x) \Sigma\{e\} : \text{ok in } C}$$
$$\Sigma \subseteq \Gamma$$
$$\text{mtype}(m, C) = t_1 \rightarrow t_2$$
$$t <: t_2$$

$$\frac{\Sigma, \text{this} : C, x : \pi.\text{end}; c : \pi.\text{end} \vdash e[c/x] : t; c : \text{end}}{\Gamma, \text{this} : C \vdash t_2 m(\pi x) \Sigma\{e\} : \text{ok in } C}$$
$$\Sigma \subseteq \Gamma$$
$$\text{mtype}(m, C) = \pi \rightarrow t_2$$
$$t <: t_2$$

Typing Method Call

$$\frac{\Gamma; \Sigma \vdash e : C; \Sigma' \quad \Gamma; \Sigma' \vdash e' : t'; \Sigma''}{\Gamma; \Sigma \vdash e.m(e') : t; \Sigma''}$$

$$\text{msignature}(m, C) \subseteq \Gamma$$

$$\text{mtype}(m, C) = t'' \rightarrow t$$

$$t' <: t''$$

$$\frac{\Gamma; \Sigma \vdash e : C; \Sigma', c : \pi.s}{\Gamma; \Sigma \vdash e.m(c) : t; \Sigma', c : s}$$

$$\text{msignature}(m, C) \subseteq \Gamma$$

$$\text{mtype}(m, C) = \pi \rightarrow t$$

Branching Types

$$\frac{\Gamma; \Sigma, c : \pi_1.s \vdash e_1 : t; \Sigma', c : s \quad \Gamma; \Sigma, c : \pi_2.s \vdash e_2 : t; \Sigma', c : s}{\Gamma; \Sigma, c : ?\langle \pi_1, \pi_2 \rangle.s \vdash c.\mathbf{receiveIf} \{e_1\} \{e_2\} : t; \Sigma', c : s}$$

$$\frac{\Gamma; \Sigma \vdash e : \mathbf{bool}; \Sigma \quad \begin{array}{l} \Gamma; \Sigma, c : \pi_1.s \vdash e_1 : t; \Sigma', c : s \\ \Gamma; \Sigma, c : \pi_2.s \vdash e_2 : t; \Sigma', c : s \end{array}}{\Gamma; \Sigma, c : !\langle \pi_1, \pi_2 \rangle.s \vdash c.\mathbf{sendIf}(e) \{e_1\} \{e_2\} : t; \Sigma', c : s}$$

Recursive Types

$$\frac{\Gamma; \Sigma, c : \pi.s \vdash e : t; \Sigma, c : s}{\Gamma; \Sigma, c : ?\langle \pi \rangle^*.s \vdash c.\mathbf{receiveWhile} \{e\} : t; \Sigma, c : s}$$

$$\frac{\Gamma; \Sigma \vdash e : \mathbf{bool}; \Sigma \quad \Gamma; \Sigma, c : \pi.s \vdash e' : t; \Sigma, c : s}{\Gamma; \Sigma, c : !\langle \pi \rangle^*.s \vdash c.\mathbf{sendWhile} (e) \{e'\} : t; \Sigma, c : s}$$

Communication of running Channels

$$\frac{\Gamma, x:s; \Sigma, c:s, d:s' \vdash e[c/x] : t, \Sigma', c:\mathbf{end}, d:s''}{\Gamma, x:s; \Sigma, d:?!s.s' \vdash x := c.\mathbf{receive}; e : t; \Sigma', d:s''}$$

$$\Gamma; \Sigma, c:s, d:?!s.s' \vdash d.\mathbf{send}(c) : \mathit{Object}; \Sigma, c:\mathbf{end}, d:s'$$