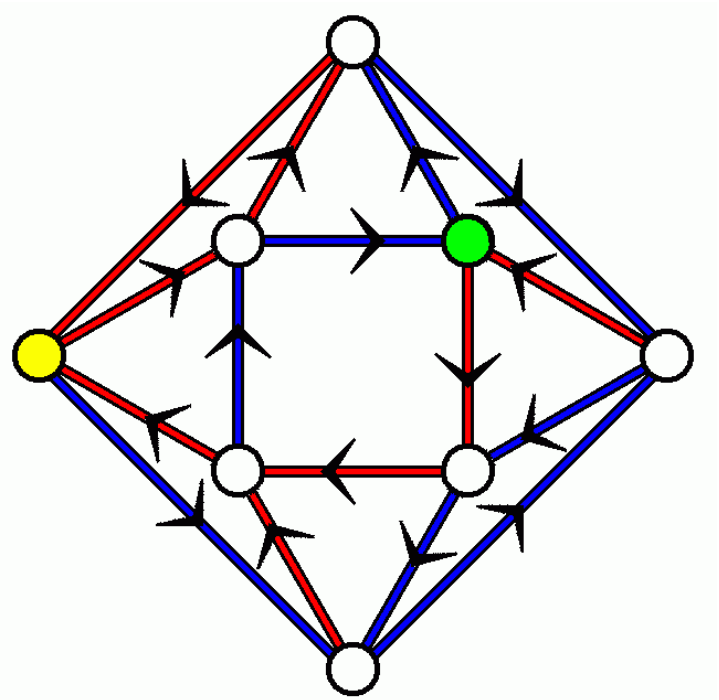
Buffered Communication Analysis in Distributed Multiparty Sessions

Pierre-Malo Deniélou Nobuko Yoshida

Imperial College London



Buffer analysis and binary session types

Asynchrony and buffer analysis

- Asynchronous communications rely on communication buffers, which may grow over time and overflow.
- Preventing communication buffers to overflow is a well-studied area.

Binary session types (two-party) [Takeuchi&al.'94]

- Binary session types specify channel behaviour in an asynchronous π-calculus (with message-order preserving, non-blocking sends).
- Binary session types give some information about buffers:

$$P_0 = s!\langle 3\rangle.s!\langle Orange\rangle.s?(x) \qquad T_0 = !\langle \mathsf{nat}\rangle; !\langle \mathsf{string}\rangle; ?\langle \mathsf{real}\rangle; \mathsf{end}$$

$$P_1 = s!\langle 3\rangle.s?(x).s!\langle Orange\rangle \qquad T_1 = !\langle \mathsf{nat}\rangle; ?\langle \mathsf{real}\rangle; !\langle \mathsf{string}\rangle; \mathsf{end}$$

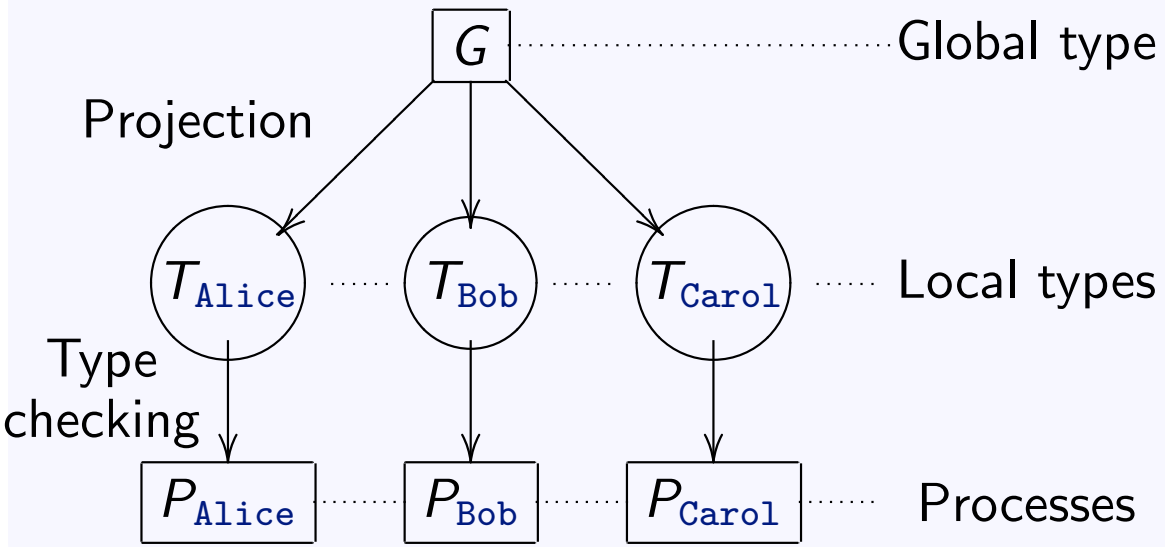
Multiparty sessions for protocol specification and analysis

- Today's distributed applications involve more and more agents that interact through complex communication patterns.
- Multiparty sessions types can describe these interactions and statically ensure type and communication safety and fidelity to a stipulated protocol.

Multiparty sessions for protocol specification and analysis

- Today's distributed applications involve more and more agents that interact through complex communication patterns.
- Multiparty sessions types can describe these interactions and statically ensure type and communication safety and fidelity to a stipulated protocol.

Multiparty session types in a nutshell



$$G = \texttt{Alice} \to \texttt{Bob}\colon s_1\,\langle \mathsf{nat}\rangle;\ \texttt{Bob} \to \texttt{Carol}\colon s_2\,\langle \mathsf{nat}\rangle; \mathsf{end}$$

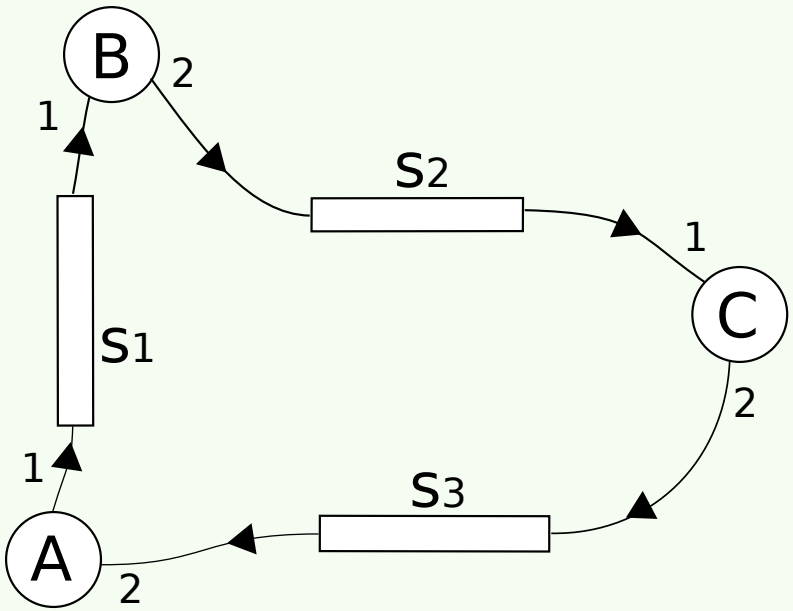
$$T_{\texttt{Bob}} = s_1?\langle \mathsf{nat}\rangle.s_2!\langle \mathsf{nat}\rangle.\mathsf{end}$$

$$P_{\texttt{Bob}} = s_1?(x).s_2!\,\langle x+1\rangle$$

Buffer analysis in the multiparty case

Binary session type techniques to analyse buffer usage do not apply directly.

Example (a)



(A) $\texttt{Alice} = \mu X.s_1!\langle 1\rangle; s_3?(x); X$

(B) $\texttt{Bob} = \mu X.s_1?(x); s_2!\langle \mathit{Orange}\rangle; X$

(C) $\texttt{Carol} = \mu X.s_2?(x); s_3!\langle 2.4\rangle; X$

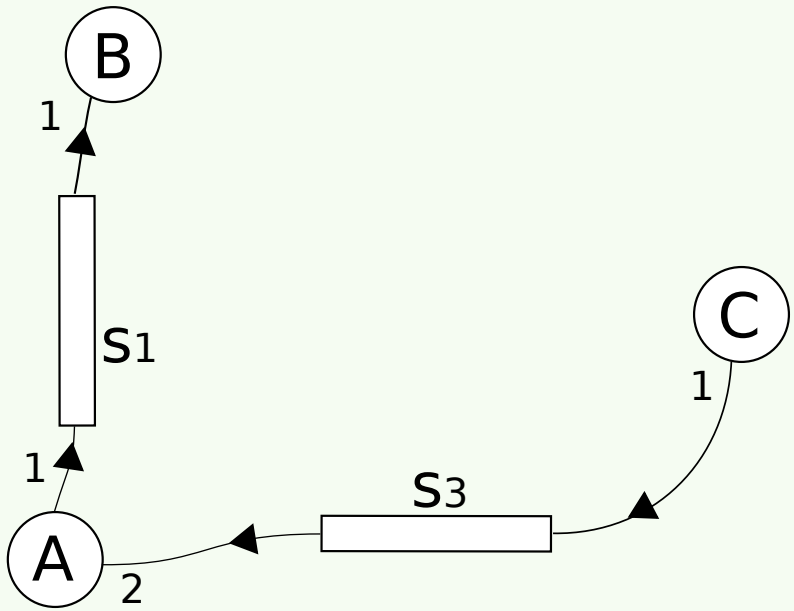
$$G_a = \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}: s_1\langle \mathsf{nat}\rangle; \texttt{Bob} \to \texttt{Carol}: s_2\langle \mathsf{string}\rangle; \texttt{Carol} \to \texttt{Alice}: s_3\langle \mathsf{real}\rangle; \mathbf{x}$$

- All three buffers need not hold more than one value.

Buffer analysis in the multiparty case

Binary session type techniques to analyse buffer usage do not apply directly.

Example (b)



(A) $\texttt{Alice} = \mu X.s_1!\langle 1\rangle; s_3?(x); X$

(B) $\texttt{Bob} = \mu X.s_1?(x); X$

(C) $\texttt{Carol} = \mu X.s_3!\langle 2.4\rangle; X$

$$G_b = \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}: s_1\ \langle \mathsf{nat}\rangle;$$

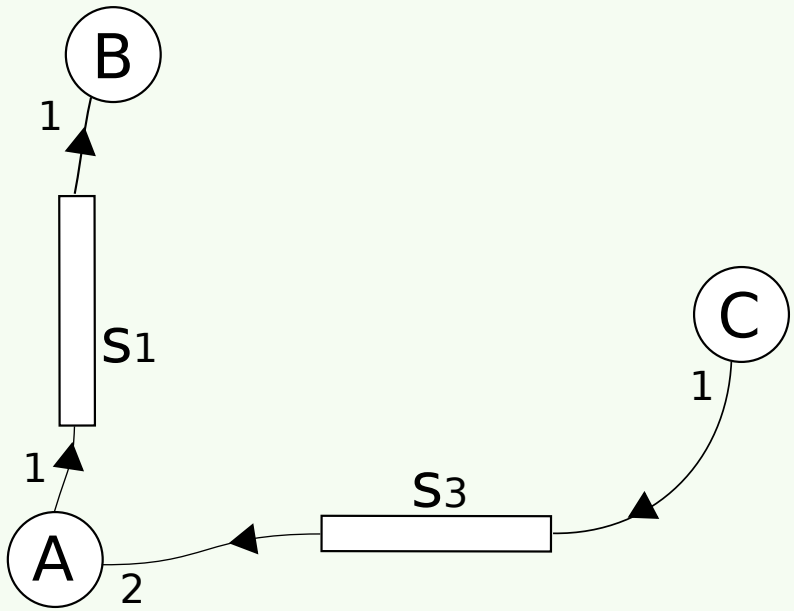
$$\texttt{Carol} \to \texttt{Alice}: s_3\ \langle \mathsf{real}\rangle; \mathbf{x}$$

- Values can accumulate in the two remaining buffers.

Buffer analysis in the multiparty case

Binary session type techniques to analyse buffer usage do not apply directly.

Example (b)



(A) $\texttt{Alice} = \mu X.s_1!\langle 1\rangle; s_3?(x); X$

(B) $\texttt{Bob} = \mu X.s_1?(x); X$

(C) $\texttt{Carol} = \mu X.s_3!\langle 2.4\rangle; X$

$$G_b = \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}: s_1\ \langle \text{nat}\rangle;$$

$$\texttt{Carol} \to \texttt{Alice}: s_3\ \langle \text{real}\rangle; \mathbf{x}$$

- Values can accumulate in the two remaining buffers.
- `Alice`'s local behaviour is the same in example (a) and (b).

Outline

I A general causality analysis of global session types
   a An algorithm to detect infinite buffers
   b An algorithm to compute the buffer size of finite buffers
   c A type system to ensure communication, buffer and type safety and progress properties.

II A framework for session programming and analysis
   a Channel allocation
   b Global refinement
   c Local refinement (messaging optimisation)

Outline

I A general causality analysis of global session types

a An algorithm to detect infinite buffers
b An algorithm to compute the buffer size of finite buffers
c A type system to ensure communication, buffer and type safety and progress properties.

Global types and session graphs

Syntax of global types

$$
\begin{array}{rcll}
G & ::= & \mathtt{p} \to \mathtt{p}' : k\,\langle U \rangle ; G' & \text{values} \\
& | & \mathtt{p} \to \mathtt{p}' : k\,\{l_j : G_j\}_{j \in J} & \text{branching} \\
& | & \mu \mathbf{x}.G \mid \mathbf{x} \mid \text{end} & \text{recursion, end}
\end{array}
$$

Running example

$$
\mu \mathbf{x}.\mathtt{Alice} \to \mathtt{Bob} : s_1\,\langle \text{nat} \rangle ; \mathtt{Bob} \to \mathtt{Carol} : s_2 \left\{ \begin{array}{l} l_1 : \mathtt{Carol} \to \mathtt{Alice} : s_3 \{ l_3 : \mathtt{Alice} \to \mathtt{Carol} : s_2\,\langle \text{string} \rangle ; \mathbf{x} \} \\ l_2 : \mathtt{Carol} \to \mathtt{Alice} : s_3 \{ l_4 : \text{end} \} \end{array} \right\}
$$

Global types and session graphs

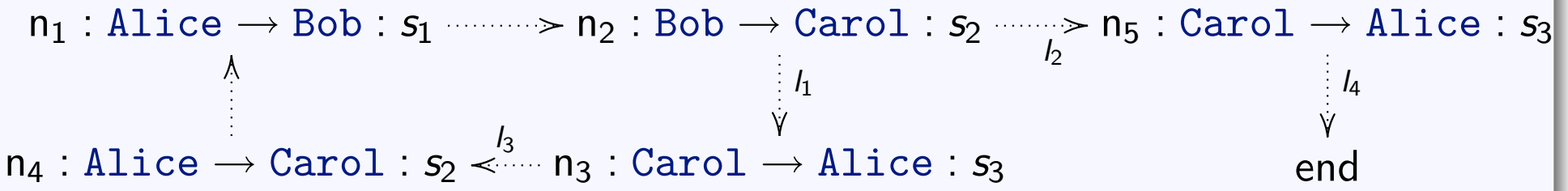
Syntax of global types

$$
\begin{array}{rcll}
G & ::= & \mathtt{p} \to \mathtt{p}' : k\,\langle U \rangle ; G' & \text{values} \\
& | & \mathtt{p} \to \mathtt{p}' : k\,\{l_j : G_j\}_{j \in J} & \text{branching} \\
& | & \mu \mathbf{x}.G \mid \mathbf{x} \mid \mathtt{end} & \text{recursion, end}
\end{array}
$$

Running example

$$
\mu \mathbf{x}.\mathtt{Alice} \to \mathtt{Bob} : s_1\,\langle \mathsf{nat} \rangle ; \mathtt{Bob} \to \mathtt{Carol} : s_2 \left\{ \begin{array}{l} l_1 : \mathtt{Carol} \to \mathtt{Alice} : s_3 \{ l_3 : \mathtt{Alice} \to \mathtt{Carol} : s_2\,\langle \mathsf{string} \rangle ; \mathbf{x} \} \\ l_2 : \mathtt{Carol} \to \mathtt{Alice} : s_3 \{ l_4 : \mathsf{end} \} \end{array} \right.
$$

Session graph



Global types and session graphs

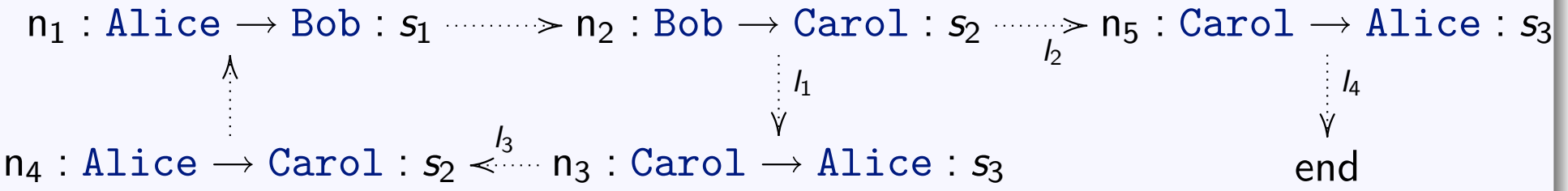
Syntax of global types

$$
\begin{array}{rcll}
G & ::= & \mathtt{p} \to \mathtt{p}' : k\,\langle U \rangle ; G' & \text{values} \\
& \mid & \mathtt{p} \to \mathtt{p}' : k\,\{l_j : G_j\}_{j \in J} & \text{branching} \\
& \mid & \mu \mathbf{x}.G \mid \mathbf{x} \mid \mathsf{end} & \text{recursion, end}
\end{array}
$$

Running example

$$
\mu \mathbf{x}.\mathtt{Alice} \to \mathtt{Bob} : s_1\,\langle \mathsf{nat} \rangle ; \mathtt{Bob} \to \mathtt{Carol} : s_2 \left\{ \begin{array}{l} l_1 : \mathtt{Carol} \to \mathtt{Alice} : s_3 \{ l_3 : \mathtt{Alice} \to \mathtt{Carol} : s_2\,\langle \mathsf{string} \rangle ; \mathbf{x} \} \\ l_2 : \mathtt{Carol} \to \mathtt{Alice} : s_3 \{ l_4 : \mathsf{end} \} \end{array} \right.
$$

Session graph



- Causality analysis = determining which edges stand up to asynchrony

Causality Analysis and Unbounded Buffers

- An IO-dependency links two communications when one guards the other.

$$\texttt{Alice} \longrightarrow \texttt{Bob} : s_1\ \langle \mathsf{nat} \rangle;$$
$$\texttt{Bob} \longrightarrow \texttt{Carol} : s_2\ \langle \mathsf{string} \rangle;$$

- IO-chains guarantee that a buffer is read before proceeding.

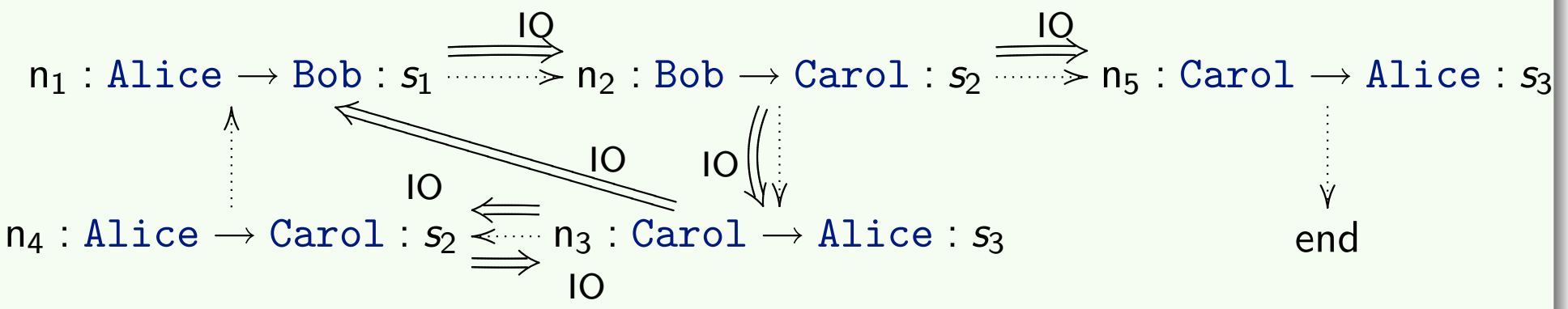
Causality Analysis and Unbounded Buffers

- An IO-dependency links two communications when one guards the other.

$$\texttt{Alice} \to \texttt{Bob}\colon s_1\ \langle \mathsf{nat} \rangle;$$
$$\texttt{Bob} \to \texttt{Carol}\colon s_2\ \langle \mathsf{string} \rangle;$$

- IO-chains guarantee that a buffer is read before proceeding.

IO-dependencies on a session graph



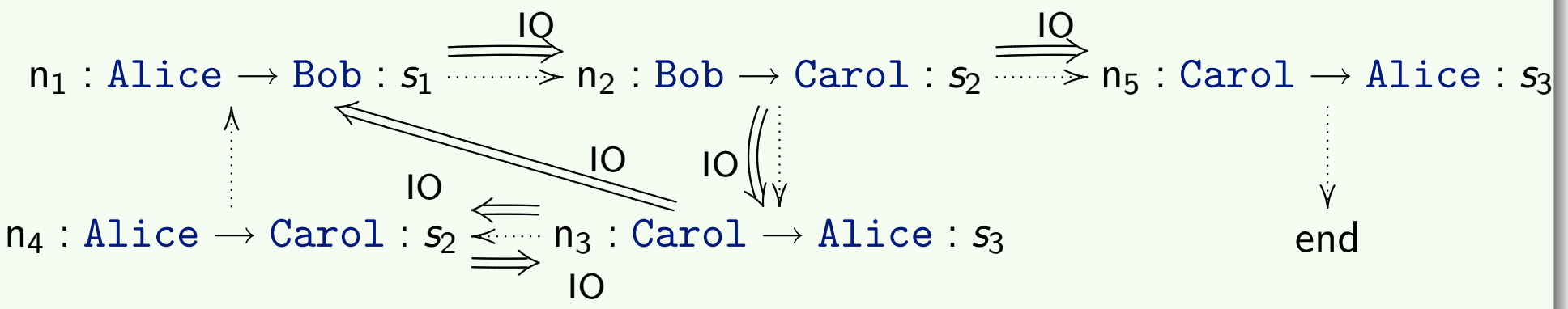
Causality Analysis and Unbounded Buffers

- An IO-dependency links two communications when one guards the other.

$$\texttt{Alice} \to \texttt{Bob}\colon s_1\ \langle \mathsf{nat} \rangle;$$
$$\texttt{Bob} \to \texttt{Carol}\colon s_2\ \langle \mathsf{string} \rangle;$$

- IO-chains guarantee that a buffer is read before proceeding.

IO-dependencies on a session graph



Infinite buffers

If a channel is used in a graph cycle that does not contain an IO-cycle, then this channel is infinite.

Reset Property

The Reset property characterises the paths in the session graph that see a buffer being emptied.

Reset property

A path satisfies the Reset property if its last node is the end of an IO-chain that starts with a node using the same buffer.

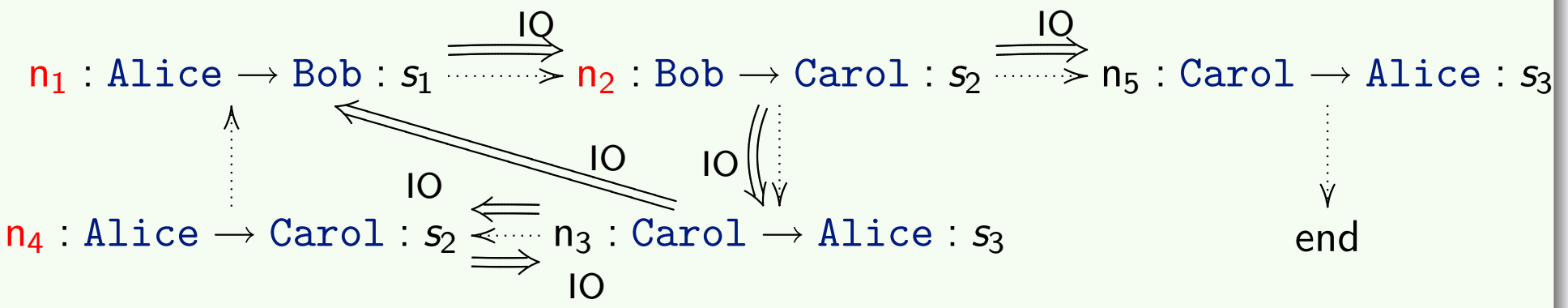
Reset Property

The Reset property characterises the paths in the session graph that see a buffer being emptied.

Reset property

A path satisfies the Reset property if its last node is the end of an IO-chain that starts with a node using the same buffer.

Example



$\mathrm{Reset}(n_4 n_1 n_2)$

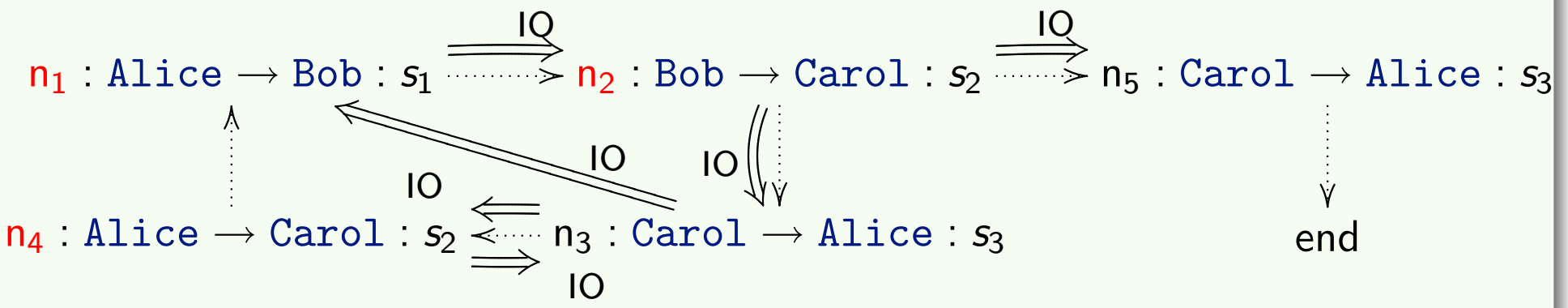
Reset Property

The Reset property characterises the paths in the session graph that see a buffer being emptied.

Reset property

A path satisfies the Reset property if its last node is the end of an IO-chain that starts with a node using the same buffer.

Example



$\mathrm{Reset}(n_4 n_1 n_2)$ does not hold,

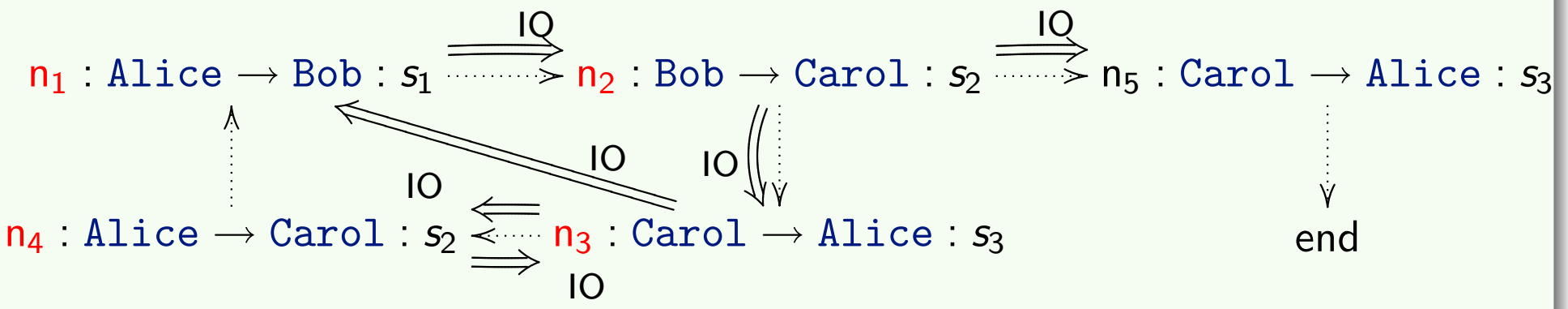
Reset Property

The Reset property characterises the paths in the session graph that see a buffer being emptied.

Reset property

A path satisfies the Reset property if its last node is the end of an IO-chain that starts with a node using the same buffer.

Example



$\mathrm{Reset}(n_4 n_1 n_2)$ does not hold, $\mathrm{Reset}(n_1 n_2 n_3 n_4)$

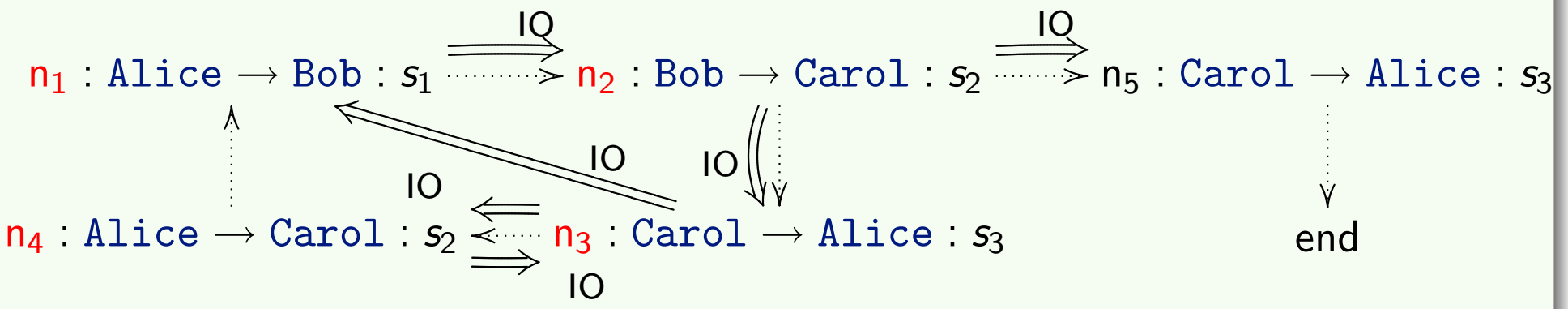
Reset Property

The Reset property characterises the paths in the session graph that see a buffer being emptied.

Reset property

A path satisfies the Reset property if its last node is the end of an IO-chain that starts with a node using the same buffer.

Example



$\mathrm{Reset}(n_4 n_1 n_2)$ does not hold, $\mathrm{Reset}(n_1 n_2 n_3 n_4)$ holds.

An algorithm to compute buffer sizes

- An edge in a session graph between nodes n_1 and n_2 is noted $\mathsf{n}_1 \prec \mathsf{n}_2$
- All values are considered of size 1.

Algorithm (computes the bound for finite channel k)

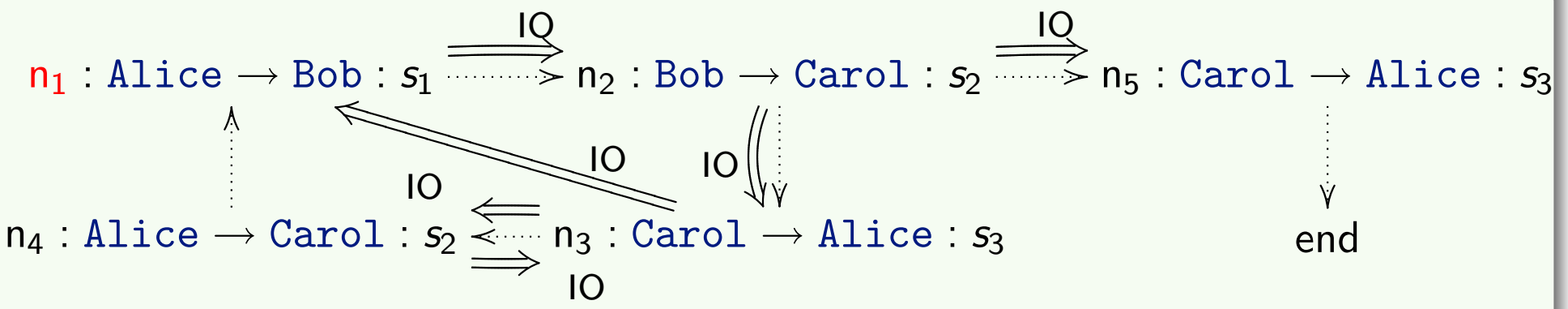
$$\mathcal{B}_k\langle m, \mathcal{P}, \tilde{\mathsf{n}}, \mathsf{n}\rangle = \begin{cases} 0 & \text{if } \mathsf{n} = \text{end or } \tilde{\mathsf{n}} \in \mathcal{P} \\ \max_{\mathsf{n} \prec \mathsf{n}'} \mathcal{B}_k\langle m, \{\tilde{\mathsf{n}}\} \cup \mathcal{P}, \tilde{\mathsf{n}}\mathsf{n}, \mathsf{n}'\rangle & \text{if } \mathtt{ch}(\mathsf{n}) = k', k \neq k' \\ \max_{\mathsf{n} \prec \mathsf{n}'} \mathcal{B}_k\langle 1, \{\tilde{\mathsf{n}}\} \cup \mathcal{P}, \mathsf{n}, \mathsf{n}'\rangle & \text{if } \mathtt{ch}(\mathsf{n}) = k, \mathrm{Reset}(\tilde{\mathsf{n}}\mathsf{n}) \\ \max(m+1, \max_{\mathsf{n} \prec \mathsf{n}'} \mathcal{B}_k\langle m+1, \{\tilde{\mathsf{n}}\} \cup \mathcal{P}, \tilde{\mathsf{n}}\mathsf{n}, \mathsf{n}'\rangle) & \text{if } \mathtt{ch}(\mathsf{n}) = k, \neg\mathrm{Reset}(\tilde{\mathsf{n}}\mathsf{n}) \end{cases}$$

- $\mathrm{Reset}(\tilde{\mathsf{n}})$ can be computed incrementaly
- the length of paths is bounded by linearity and the number of participants

This gives an efficient and practical algorithm.

Example of buffer bound computation

Example: bound of channel s_2

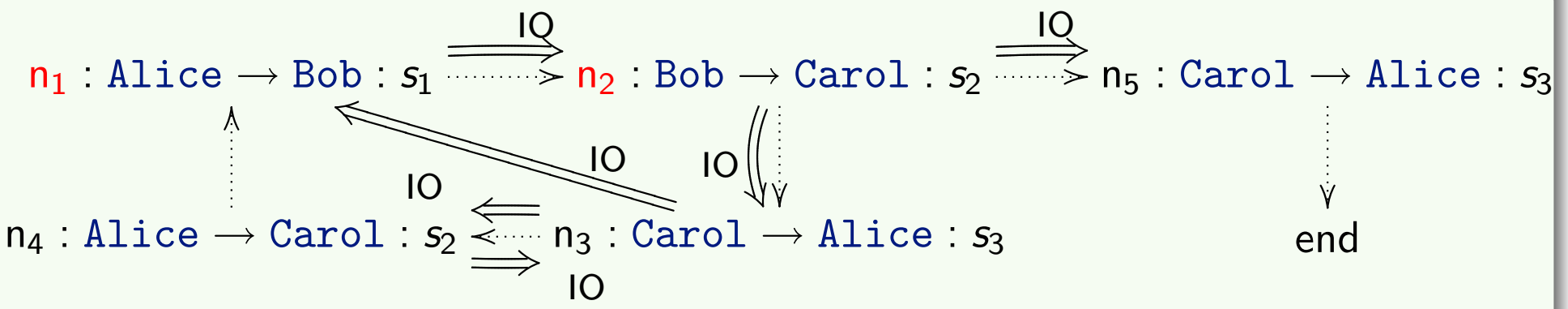


$\mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1 \rangle$

max	explanation

Example of buffer bound computation

Example: bound of channel s_2

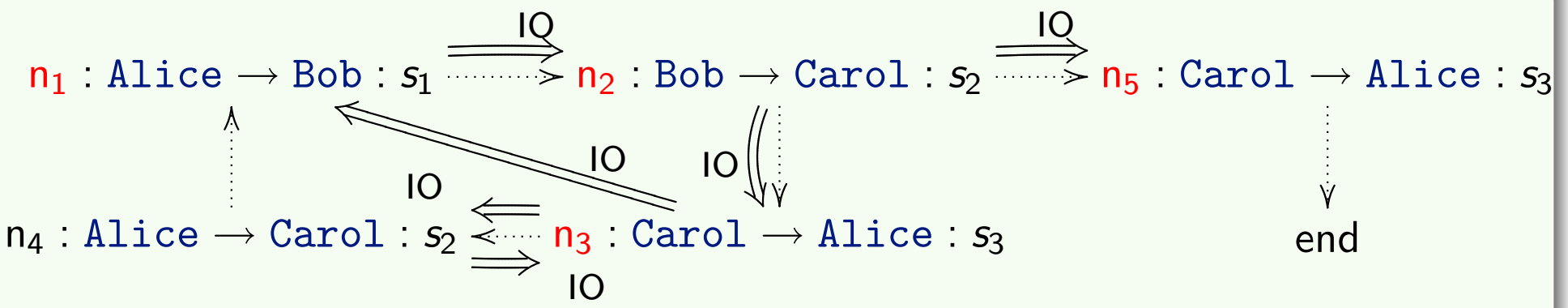


$$\begin{aligned} & \mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1\rangle \\ = \quad & \mathcal{B}_{s_2}\langle 0, \mathsf{n}_1, \mathsf{n}_2\rangle \end{aligned}$$

max	explanation
0	$s_1 \neq s_2$

Example of buffer bound computation

Example: bound of channel s_2

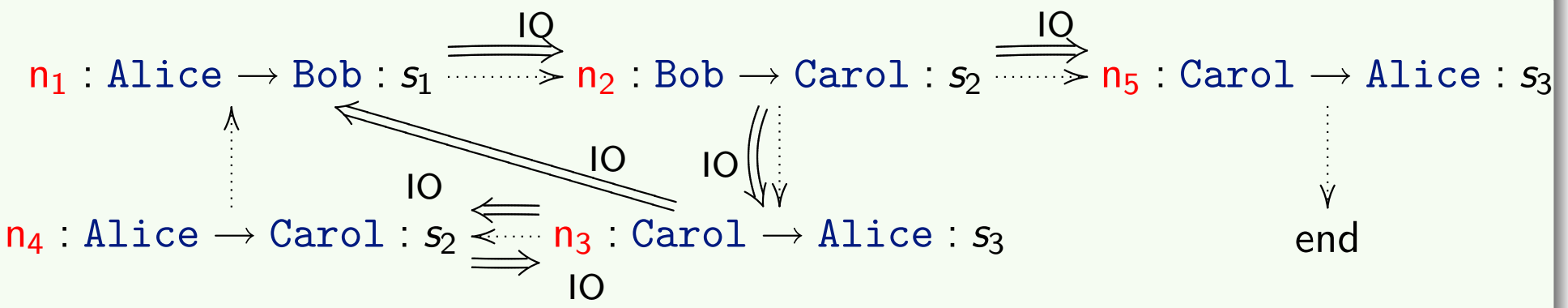


$$
\begin{aligned}
& \mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1\rangle \\
= \quad & \mathcal{B}_{s_2}\langle 0, \mathsf{n}_1, \mathsf{n}_2\rangle \\
= \quad & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle)
\end{aligned}
$$

max	explanation
0	$s_1 \neq s_2$
1	$\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2)$

Example of buffer bound computation

Example: bound of channel s_2

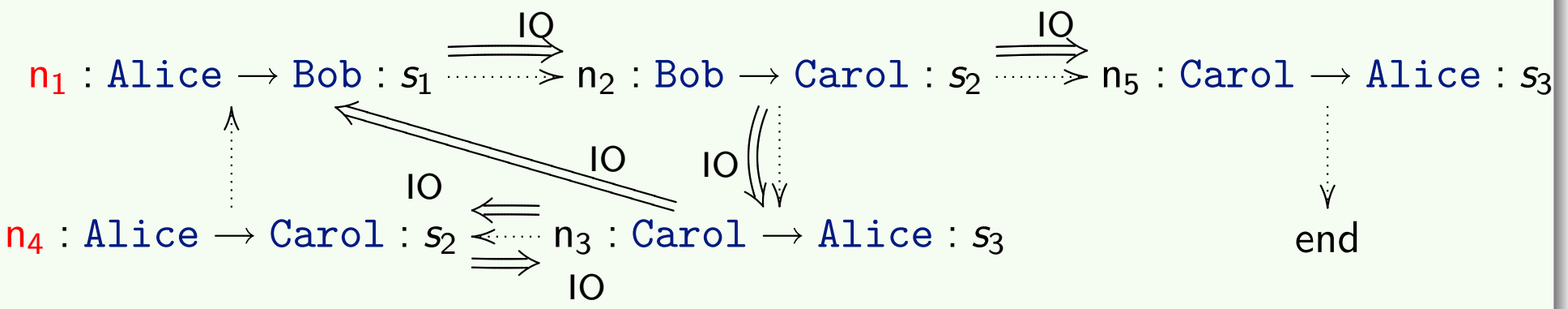


$$
\begin{aligned}
& \mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1\rangle \\
= \quad & \mathcal{B}_{s_2}\langle 0, \mathsf{n}_1, \mathsf{n}_2\rangle \\
= \quad & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle) \\
= \quad & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_3, \mathsf{n}_4\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_5, \mathsf{end}\rangle)
\end{aligned}
$$

max	explanation
0	$s_1 \neq s_2$
1	$\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2)$
1	$s_3 \neq s_2$

Example of buffer bound computation

Example: bound of channel s_2

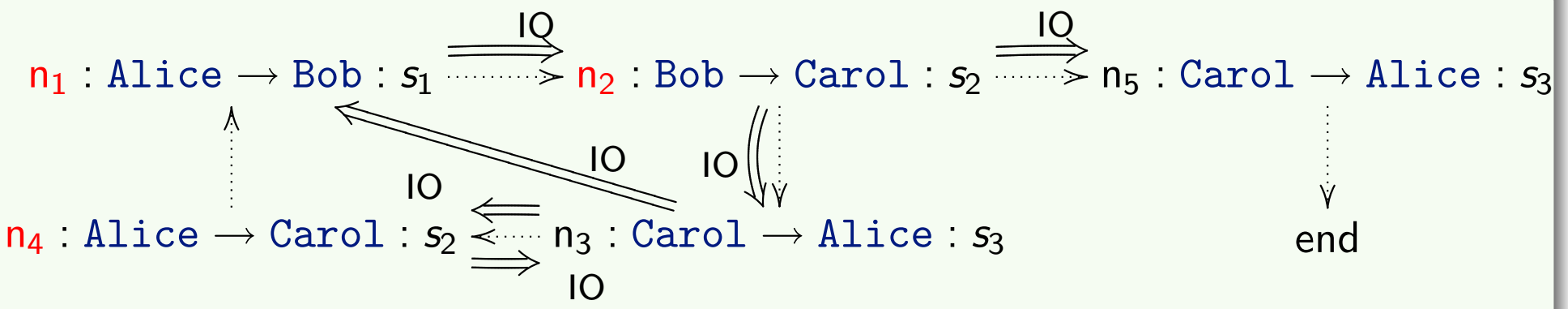


$$
\begin{aligned}
& \mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1\rangle \\
= \quad & \mathcal{B}_{s_2}\langle 0, \mathsf{n}_1, \mathsf{n}_2\rangle \\
= \quad & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle) \\
= \quad & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_3, \mathsf{n}_4\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_5, \mathsf{end}\rangle) \\
= \quad & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_4, \mathsf{n}_1\rangle, 0)
\end{aligned}
$$

max	explanation
0	$s_1 \neq s_2$
1	$\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2)$
1	$s_3 \neq s_2$
1	$\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$

Example of buffer bound computation

Example: bound of channel s_2

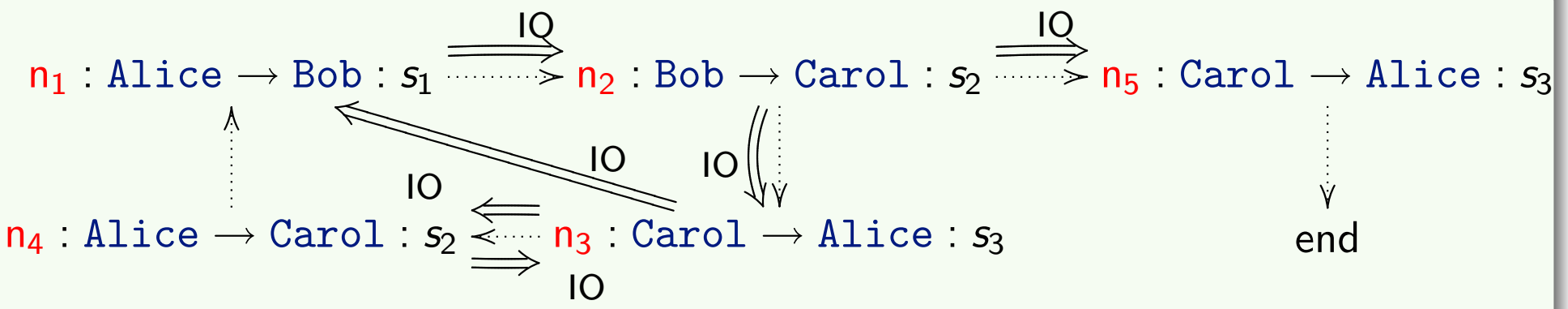


$$
\begin{aligned}
& \mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1\rangle \\
= \;& \mathcal{B}_{s_2}\langle 0, \mathsf{n}_1, \mathsf{n}_2\rangle \\
= \;& \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle) \\
= \;& \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_3, \mathsf{n}_4\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_5, \mathsf{end}\rangle) \\
= \;& \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_4, \mathsf{n}_1\rangle, 0) \\
= \;& \mathcal{B}_{s_2}\langle 1, \mathsf{n}_4\mathsf{n}_1, \mathsf{n}_2\rangle
\end{aligned}
$$

max	explanation
0	$s_1 \neq s_2$
1	$\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2)$
1	$s_3 \neq s_2$
1	$\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$
1	$s_1 \neq s_2$

Example of buffer bound computation

Example: bound of channel s_2

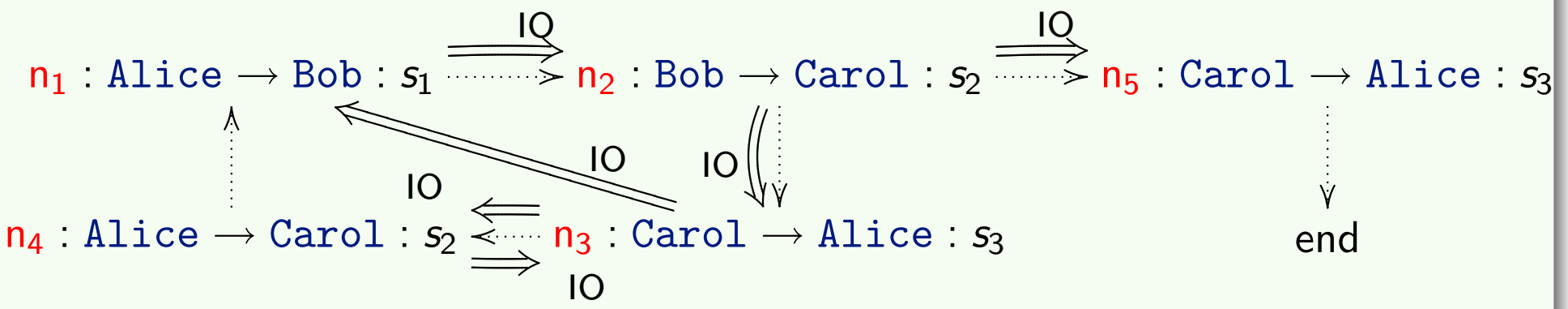


$$
\begin{aligned}
& \mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1\rangle \\
= \;& \mathcal{B}_{s_2}\langle 0, \mathsf{n}_1, \mathsf{n}_2\rangle \\
= \;& \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle) \\
= \;& \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_3, \mathsf{n}_4\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_5, \mathsf{end}\rangle) \\
= \;& \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_4, \mathsf{n}_1\rangle, 0) \\
= \;& \mathcal{B}_{s_2}\langle 1, \mathsf{n}_4\mathsf{n}_1, \mathsf{n}_2\rangle \\
= \;& \max(\mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle)
\end{aligned}
$$

max	explanation
0	$s_1 \neq s_2$
1	$\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2)$
1	$s_3 \neq s_2$
1	$\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$
1	$s_1 \neq s_2$
2	$\neg\mathrm{Reset}(\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2)$

Example of buffer bound computation

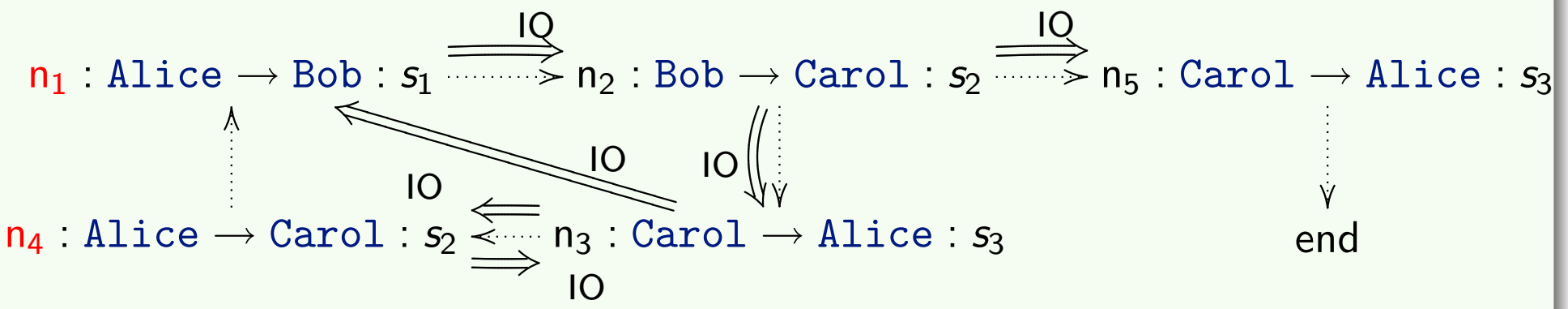
Example: bound of channel s_2



	max	explanation
$\mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1\rangle$		
$= \mathcal{B}_{s_2}\langle 0, \mathsf{n}_1, \mathsf{n}_2\rangle$	0	$s_1 \neq s_2$
$= \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle)$	1	$\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2)$
$= \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_3, \mathsf{n}_4\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_5, \mathsf{end}\rangle)$	1	$s_3 \neq s_2$
$= \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_4, \mathsf{n}_1\rangle, 0)$	1	$\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$
$= \mathcal{B}_{s_2}\langle 1, \mathsf{n}_4\mathsf{n}_1, \mathsf{n}_2\rangle$	1	$s_1 \neq s_2$
$= \max(\mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle)$	2	$\neg\mathrm{Reset}(\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2)$
$= \max(\mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3, \mathsf{n}_4\rangle, \mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_5, \mathsf{end}\rangle)$	2	$s_3 \neq s_2$

Example of buffer bound computation

Example: bound of channel s_2



$$
\begin{array}{ll}
& \mathcal{B}_{s_2}\langle 0, \epsilon, \mathsf{n}_1\rangle \\
= & \mathcal{B}_{s_2}\langle 0, \mathsf{n}_1, \mathsf{n}_2\rangle \\
= & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle) \\
= & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_3, \mathsf{n}_4\rangle, \mathcal{B}_{s_2}\langle 1, \mathsf{n}_1\mathsf{n}_2\mathsf{n}_5, \mathsf{end}\rangle) \\
= & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_4, \mathsf{n}_1\rangle, 0) \\
= & \mathcal{B}_{s_2}\langle 1, \mathsf{n}_4\mathsf{n}_1, \mathsf{n}_2\rangle \\
= & \max(\mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2, \mathsf{n}_3\rangle, \mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2, \mathsf{n}_5\rangle) \\
= & \max(\mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3, \mathsf{n}_4\rangle, \mathcal{B}_{s_2}\langle 2, \mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_5, \mathsf{end}\rangle) \\
= & \max(\mathcal{B}_{s_2}\langle 1, \mathsf{n}_4, \mathsf{n}_1\rangle, 0)
\end{array}
$$

max	explanation
0	$s_1 \neq s_2$
1	$\neg\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2)$
1	$s_3 \neq s_2$
1	$\mathrm{Reset}(\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$
1	$s_1 \neq s_2$
2	$\neg\mathrm{Reset}(\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2)$
2	$s_3 \neq s_2$
2	$\mathrm{Reset}(\mathsf{n}_4\mathsf{n}_1\mathsf{n}_2\mathsf{n}_3\mathsf{n}_4)$

Processes, semantics and type system

Thanks to this algorithm, we can statically enforce buffer safety by typing.

Processes and semantics

$$\overline{a}_{[2..n]}\,(\tilde{s}\,\tilde{n}).P_1 \mid a_{[2]}\,(\tilde{s}).P_2 \mid \cdots \mid a_{[n]}\,(\tilde{s}).P_n \longrightarrow (\nu\,\tilde{s})(P_1 \mid P_2 \mid \ldots \mid P_n \mid s_1^{n_1} : \emptyset \mid \ldots \mid s_m^{n_m} : \emptyset)$$

$$s!\,\langle e \rangle; P \mid s^n : \tilde{h} \longrightarrow P \mid s^n : \tilde{h} \cdot v \qquad (n \gtrneqq |\tilde{h}|,\ e \downarrow v)$$

$$s?(x); P \mid s^n : v \cdot \tilde{h} \longrightarrow P[v/x] \mid s^n : \tilde{h}$$

Typing system

$$\frac{\Gamma \vdash a : G \qquad \Gamma \vdash P \triangleright \Delta, \tilde{s}^{\tilde{m}} : (G \upharpoonright 1) \qquad |\tilde{s}| = chan(G) \qquad \mathcal{B}_k\langle G \rangle = m_k}{\Gamma \vdash \overline{a}_{[2..n]}\,(\tilde{s}\,\tilde{m}).P \triangleright \Delta}$$

Type, buffer and communication safety, progress

Theorem (Subject reduction)

$\Gamma \vdash_\Sigma P \triangleright \Delta$ and $P \longrightarrow Q$ with $(\Gamma, \Sigma, P, \Delta)$ fully coherent imply $\Gamma \vdash_\Sigma Q \triangleright \Delta'$ for some Δ', s_k such that $\Delta = \Delta'$ or $\Delta(\xrightarrow{s_k})^* \Delta'$ and $\mathcal{B}\langle G(k)\rangle \geq \mathcal{B}\langle G'(k)\rangle$ with $\Delta(\tilde{s}) = [\![G]\!]$, $\Delta'(\tilde{s}) = [\![G']\!]$ and $(\Gamma, \Sigma, Q, \Delta')$ fully coherent.

We define the *buffer overflow error* as follows:

$$n \leq |\tilde{h}| \quad \Rightarrow \quad s!\langle \tilde{e} \rangle; P \mid s^n : \tilde{h} \longrightarrow \mathsf{Err}$$

Corollary (Buffer safety)

If $\Gamma \vdash_\Sigma P \triangleright \Delta$, then for all P' s.t. $P \longrightarrow^* P'$, $P' \not\rightarrow \mathsf{Err}$.

We can also derive using standard techniques:

- Communication safety
- Progress

Outline

I A general causality analysis of global session types
   a An algorithm to detect infinite buffers
   b An algorithm to compute the buffer size of finite buffers
   c A type system to ensure communication, buffer and type safety and progress properties.

II A framework for session programming and analysis
   a Channel allocation
   b Global refinement
   c Local refinement (messaging optimisation)

Outline

II A framework for session programming and analysis

a Channel allocation
b Global refinement
c Local refinement (messaging optimisation)

Channel Allocation (by example)

When designing a session, our buffer analysis can help allocate channels

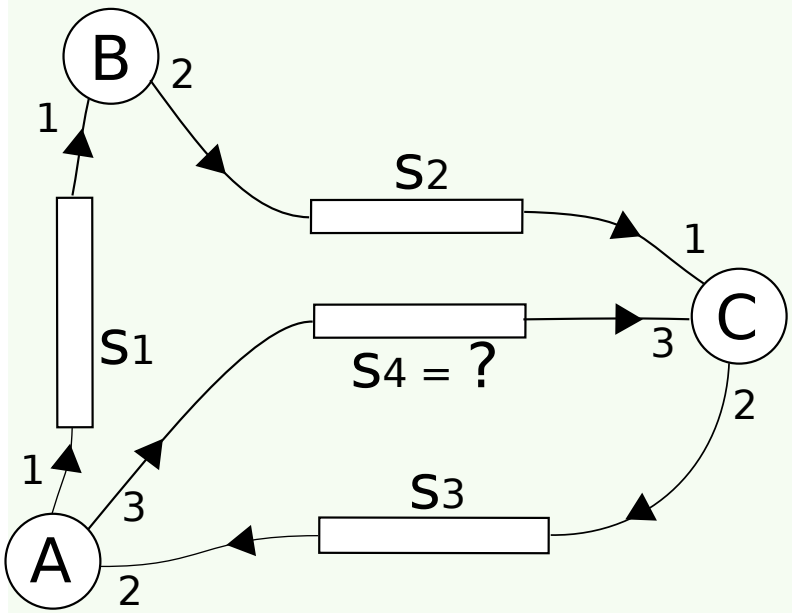
- with constrains such as a maximun number of channels or
- a maximum size of channels

Channel Allocation (by example)

When designing a session, our buffer analysis can help allocate channels

- with constrains such as a maximun number of channels or
- a maximum size of channels

Example (c)



$$(A_2)\quad \mathtt{Alice}_2 = \mu X.s_1!\langle 1\rangle; s_3?(x); s_4!\langle x+1\rangle; X$$

$$(B)\quad \mathtt{Bob} = \mu X.s_1?(x); s_2!\langle \mathit{Orange}\rangle; X$$

$$(C_2)\quad \mathtt{Carol}_2 = \mu X.s_2?(x); s_3!\langle 2.4\rangle; s_4?(y); X$$

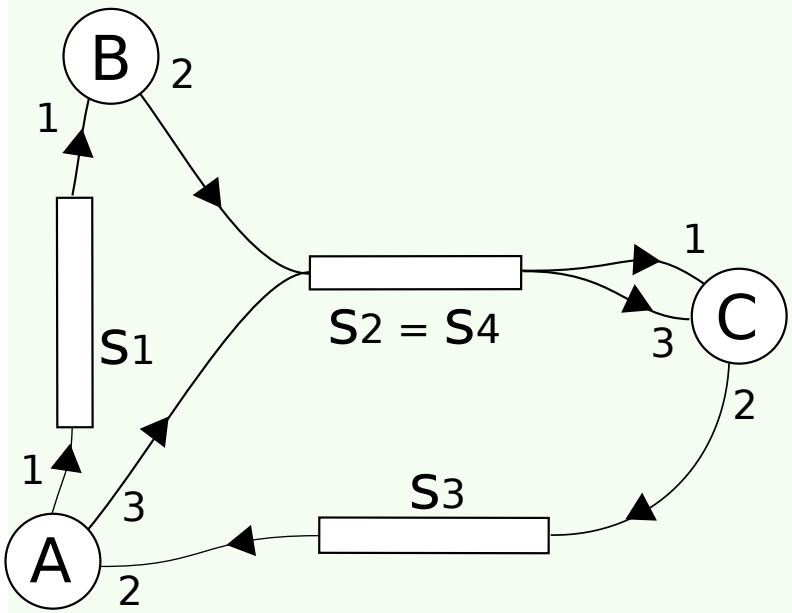
$$\begin{aligned} G_c = \mu \mathbf{x}.\ & \mathtt{Alice} \to \mathtt{Bob}: s_1\langle \mathsf{nat}\rangle; \\ & \mathtt{Bob} \to \mathtt{Carol}: s_2\langle \mathsf{string}\rangle; \\ & \mathtt{Carol} \to \mathtt{Alice}: s_3\langle \mathsf{real}\rangle; \\ & \mathtt{Alice} \to \mathtt{Carol}: s_4\langle \mathsf{nat}\rangle; \mathbf{x} \end{aligned}$$

Channel Allocation (by example)

When designing a session, our buffer analysis can help allocate channels

- with constrains such as a maximun number of channels or
- a maximum size of channels

Example (d)



$$(A_3)\quad \texttt{Alice}_3 = \mu X.s_1!\langle 1\rangle; s_3?(x); s_2!\langle x+1\rangle; X$$

$$(B)\quad \texttt{Bob} = \mu X.s_1?(x); s_2!\langle \mathit{Orange}\rangle; X$$

$$(C_3)\quad \texttt{Carol}_3 = \mu X.s_2?(x); s_3!\langle 2.4\rangle; s_2?(y); X$$

$$\begin{aligned} G_d = \mu \mathbf{x}.&\texttt{Alice} \to \texttt{Bob}: s_1\langle \mathsf{nat}\rangle;\\ &\texttt{Bob} \to \texttt{Carol}: s_2\langle \mathsf{string}\rangle;\\ &\texttt{Carol} \to \texttt{Alice}: s_3\langle \mathsf{real}\rangle;\\ &\texttt{Alice} \to \texttt{Carol}: \mathbf{s_2}\langle \mathsf{nat}\rangle; \mathbf{x} \end{aligned}$$

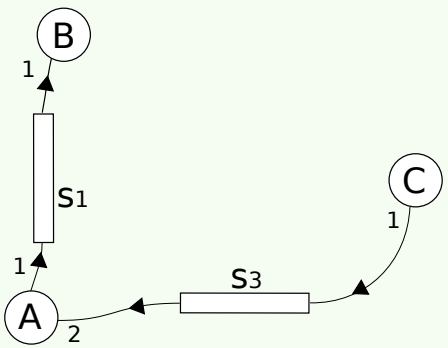
Global refinement (by example)

How do we fix overflowing buffers?

- when there is an infinite buffer or
- when a buffer size does not occupy the available space

We can insert *confirmation* (synchronisation, acknowledgement) messages

We set buffers to be of size at most 2



$$G_b = \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}: s_1\,\langle \mathsf{nat} \rangle; \texttt{Carol} \to \texttt{Alice}: s_3\,\langle \mathsf{real} \rangle; \mathbf{x}$$

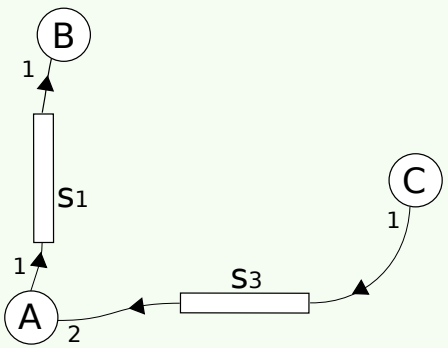
Global refinement (by example)

How do we fix overflowing buffers?

- when there is an infinite buffer or
- when a buffer size does not occupy the available space

We can insert *confirmation* (synchronisation, acknowledgement) messages

We set buffers to be of size at most 2



$$G_b = \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}: s_1\,\langle \text{nat} \rangle; \texttt{Carol} \to \texttt{Alice}: s_3\,\langle \text{real} \rangle; \mathbf{x}$$

$$G_b' = \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}: s_1\,\langle \text{nat} \rangle; \texttt{Carol} \to \texttt{Alice}: s_3\,\langle \text{real} \rangle; \texttt{Alice} \to \texttt{Bob}: s_1\,\langle \text{nat} \rangle; \texttt{Carol} \to \texttt{Alice}: s_3\,\langle \text{real} \rangle; \mathbf{x}$$

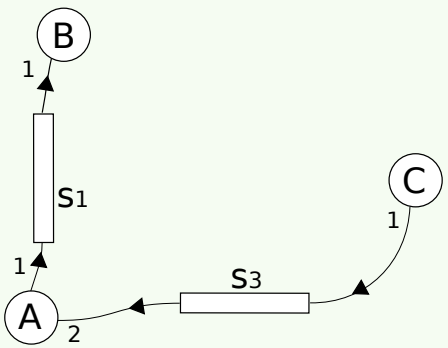
Global refinement (by example)

How do we fix overflowing buffers?

- when there is an infinite buffer or
- when a buffer size does not occupy the available space

We can insert *confirmation* (synchronisation, acknowledgement) messages

We set buffers to be of size at most 2



$$G_b = \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}: s_1\,\langle \text{nat} \rangle; \texttt{Carol} \to \texttt{Alice}: s_3\,\langle \text{real} \rangle; \mathbf{x}$$

$$G_b' = \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}: s_1\,\langle \text{nat} \rangle; \texttt{Carol} \to \texttt{Alice}: s_3\,\langle \text{real} \rangle; \texttt{Alice} \to \texttt{Bob}: s_1\,\langle \text{nat} \rangle; \underbrace{\texttt{Carol} \to \texttt{Alice}: s_3\,\langle \text{real} \rangle; \texttt{Bob} \to \texttt{Carol}: \mathbf{s_2}\,\langle \text{string} \rangle;}$$

Messaging optimisation

Messaging optimisation by example

$$P_1 = \mathbf{t}?(\mathbf{y}); \mathbf{s}!\langle \mathbf{5} \rangle; \mathbf{s}!\langle \mathrm{Orange} \rangle \qquad P_2 = \mathbf{s}?(\mathbf{z_1}); \mathbf{s}?(\mathbf{z_2})$$

$$P_3 = b?(y); \mathbf{t}!\langle \mathbf{7} \rangle$$

Messaging optimisation

Messaging optimisation by example

$$P_1 = \mathbf{t}?(\mathbf{y}); \mathbf{s}! \langle \mathbf{5} \rangle; \mathbf{s}! \langle \mathrm{Orange} \rangle \qquad P_2 = \mathbf{s}?(\mathbf{z_1}); \mathbf{s}?(\mathbf{z_2})$$

$$P_3 = b?(y); \mathbf{t}! \langle \mathbf{7} \rangle$$

Messaging optimisation

Messaging optimisation by example

$$P_1 = \mathbf{t}?(\mathbf{y}); \mathbf{s}!\langle \mathbf{5} \rangle; \mathbf{s}!\langle \mathrm{Orange} \rangle \qquad P_2 = \mathbf{s}?(\mathbf{z_1}); \mathbf{s}?(\mathbf{z_2})$$

$$P_3 = b?(y); \mathbf{t}!\langle \mathbf{7} \rangle$$

Messaging optimisation

Messaging optimisation by example

$$P_1 = \mathbf{t}?(\mathbf{y}); \mathbf{s}!\langle \mathbf{5} \rangle; \mathbf{s}!\langle \mathrm{Orange} \rangle \qquad P_2 = \mathbf{s}?(\mathbf{z_1}); \mathbf{s}?(\mathbf{z_2})$$

$$P_3 = b?(y); \mathbf{t}!\langle \mathbf{7} \rangle$$

$b?(y)$ blocks the communication.

Messaging optimisation

Messaging optimisation by example

$$P_1' = \mathbf{s}!\,\langle \mathbf{5} \rangle; \mathbf{s}!\,\langle \mathrm{Orange} \rangle; \mathbf{t}?(\mathbf{y}) \qquad P_2 = \mathbf{s}?(\mathbf{z_1}); \mathbf{s}?(\mathbf{z_2})$$

$$P_3' = \mathbf{t}!\,\langle \mathbf{7} \rangle; b?(y)$$

$b?(y)$ blocks the communication.

Messaging optimisation

Messaging optimisation by example

$$P_1' = \mathbf{s}!\langle \mathbf{5} \rangle; \mathbf{s}!\langle \mathrm{Orange} \rangle; \mathbf{t}?(\mathbf{y}) \qquad P_2 = \mathbf{s}?(\mathbf{z_1}); \mathbf{s}?(\mathbf{z_2})$$

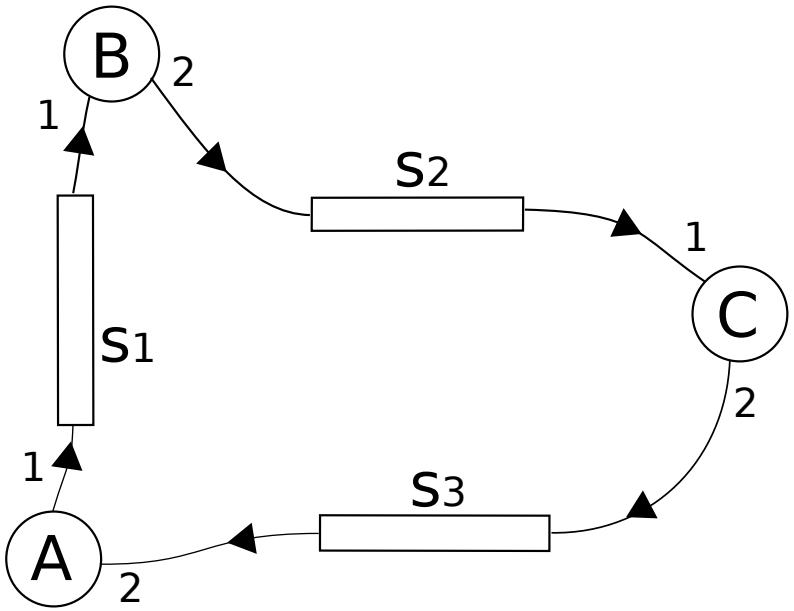
$$P_3' = \mathbf{t}!\langle \mathbf{7} \rangle; b?(y)$$

$b?(y)$ blocks the communication.

Communication subtyping [ESOP'09]

$$!\langle U \rangle; ?\langle T \rangle \leq ?\langle T \rangle; !\langle U \rangle$$

Local refinement does not preserve buffer sizes



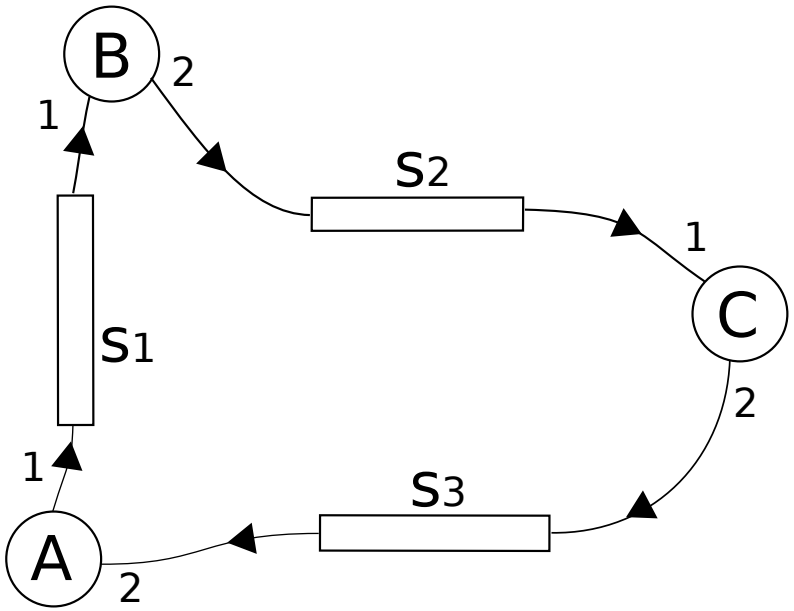
$$G \quad = \quad \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}\colon s_1\,\langle \mathsf{nat}\rangle;\ \texttt{Bob} \to \texttt{Carol}\colon s_2\,\langle \mathsf{string}\rangle;\ \texttt{Carol} \to \texttt{Alice}\colon s_3\,\langle \mathsf{real}\rangle;\mathbf{x}$$

(A) $\texttt{Alice} = \mu X.s_1!\langle 1\rangle; s_3?(x); X$

(B) $\texttt{Bob} = \mu X.\mathbf{s_1}?(\mathbf{x}); \mathbf{s_2}!\langle \mathit{Orange}\rangle; X$

(C) $\texttt{Carol} = \mu X.s_2?(x); s_3!\langle 2.4\rangle; X$

Local refinement does not preserve buffer sizes



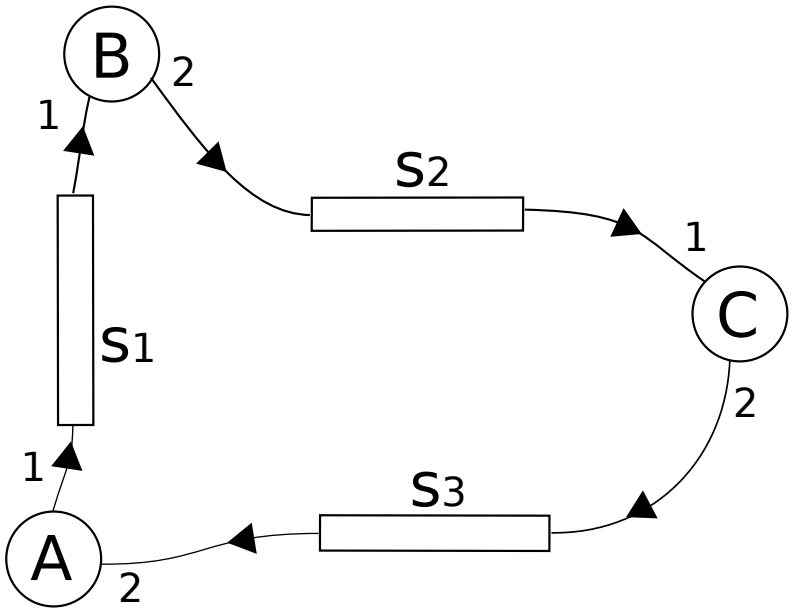
$$G \quad = \quad \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}\colon s_1\,\langle \mathsf{nat}\rangle;\ \texttt{Bob} \to \texttt{Carol}\colon s_2\,\langle \mathsf{string}\rangle;\ \texttt{Carol} \to \texttt{Alice}\colon s_3\,\langle \mathsf{real}\rangle;\mathbf{x}$$

(A) $\texttt{Alice} = \mu X.s_1!\langle 1\rangle; s_3?(x); X$ $\qquad$ $\texttt{Alice} = \mu X.s_1!\langle 1\rangle; s_3?(x); X$

(B) $\texttt{Bob} = \mu X.\mathbf{s_1}?(\mathbf{x}); \mathbf{s_2}!\langle \mathit{Orange}\rangle; X$ $\qquad$ $\texttt{Bob}' = \mu X.\mathbf{s_2}!\langle \mathit{Orange}\rangle; \mathbf{s_1}?(\mathbf{x}); X$

(C) $\texttt{Carol} = \mu X.s_2?(x); s_3!\langle 2.4\rangle; X$ $\qquad$ $\texttt{Carol} = \mu X.s_2?(x); s_3!\langle 2.4\rangle; X$

Local refinement does not preserve buffer sizes



$$G \quad = \quad \mu \mathbf{x}.\texttt{Alice} \to \texttt{Bob}\colon s_1\,\langle \text{nat}\rangle;\ \texttt{Bob} \to \texttt{Carol}\colon s_2\,\langle \text{string}\rangle;\ \texttt{Carol} \to \texttt{Alice}\colon s_3\,\langle \text{real}\rangle;\ \mathbf{x}$$

(A) $\texttt{Alice} = \mu X.s_1!\langle 1\rangle; s_3?(x); X$ $\qquad$ $\texttt{Alice} = \mu X.s_1!\langle 1\rangle; s_3?(x); X$

(B) $\texttt{Bob} = \mu X.\mathbf{s_1}?(\mathbf{x}); \mathbf{s_2}!\langle \mathit{Orange}\rangle; X$ $\qquad$ $\texttt{Bob}' = \mu X.\mathbf{s_2}!\langle \mathit{Orange}\rangle; \mathbf{s_1}?(\mathbf{x}); X$

(C) $\texttt{Carol} = \mu X.s_2?(x); s_3!\langle 2.4\rangle; X$ $\qquad$ $\texttt{Carol} = \mu X.s_2?(x); s_3!\langle 2.4\rangle; X$

- Local refinement can potentially change any buffer requirement.
- How can we determine which optimisations are safe?

Acceptable local refinement

We define a new subtyping relation that respect established buffer bounds.

Dependent nodes of a channel k

- Reset nodes: nodes that are part of an IO-chain that resets a path
- Minimal resetting paths:
  Paths that satisfy the reset property while none of their suffix does
- Dependent nodes $\mathtt{dep}(k)$:
  Union of the reset nodes of the minimal resetting paths that end with k

Dependent nodes are guards of a communication and control buffer sizes.

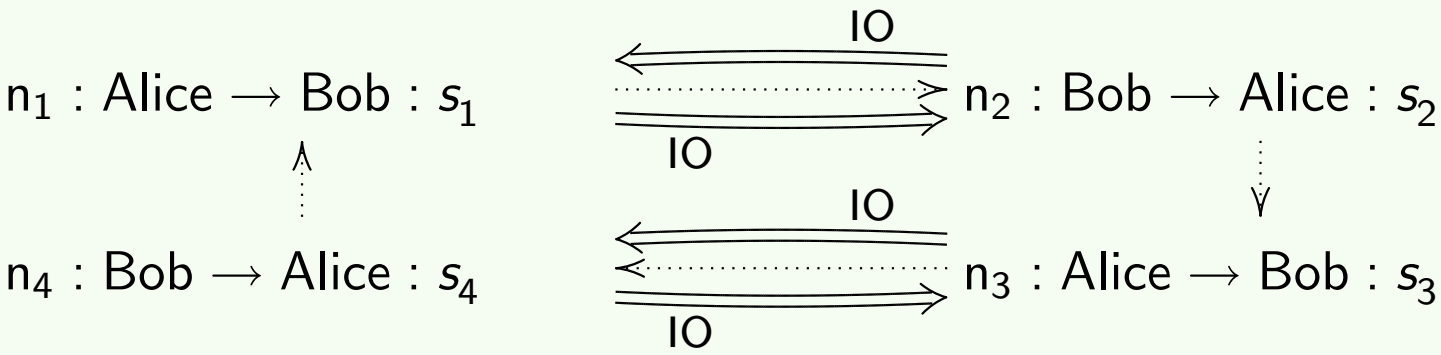
For a global type G, we choose a partitioning $\{N_0, \ldots, N_n\}$ of the set of nodes of G. This partition should satisfy the two properties:
$\forall \mathtt{n} \in N_i, \mathtt{n}' \in N_j, \mathtt{ch}(\mathtt{n}) = \mathtt{ch}(\mathtt{n}') \Rightarrow N_i = N_j$ and $\forall \mathtt{n} \in N_i, \mathtt{dep}(\mathtt{ch}(\mathtt{n})) \subset N_i$.

Asynchronous subtyping

$$\text{(OI)} \qquad k^N!\langle U\rangle; k_0^{N_0}?\langle U'\rangle; T \;\ll\; k_0^{N_0}?\langle U'\rangle; k^N!\langle U\rangle; T \qquad (N \cap N_0 = \emptyset)$$

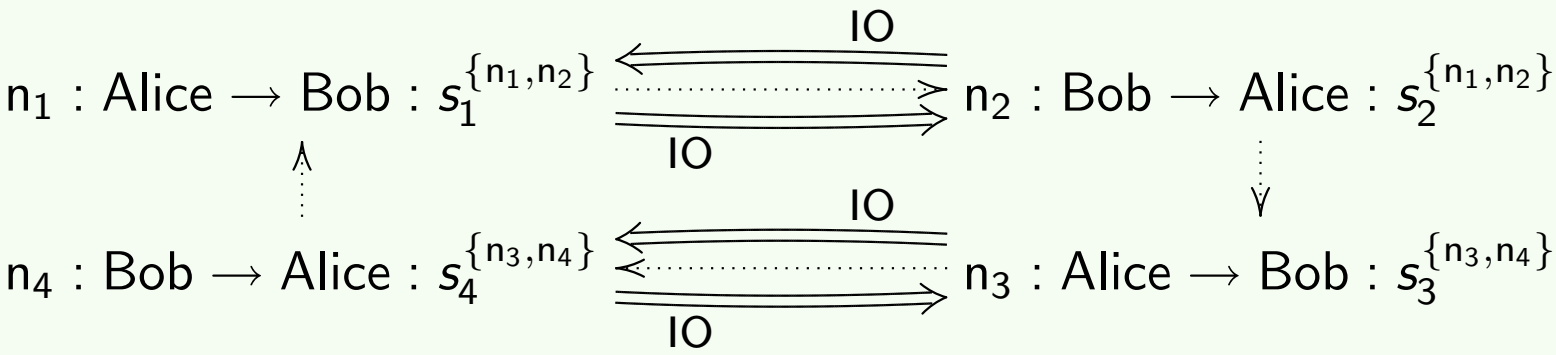
Examples of local refinement

Annotation of session channels and messaging optimisation



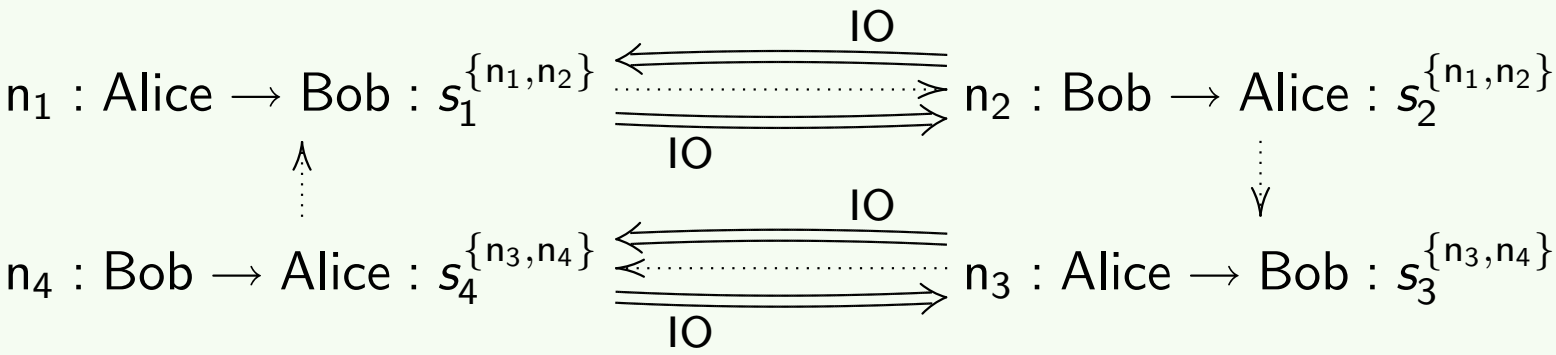
Examples of local refinement

Annotation of session channels and messaging optimisation



Examples of local refinement

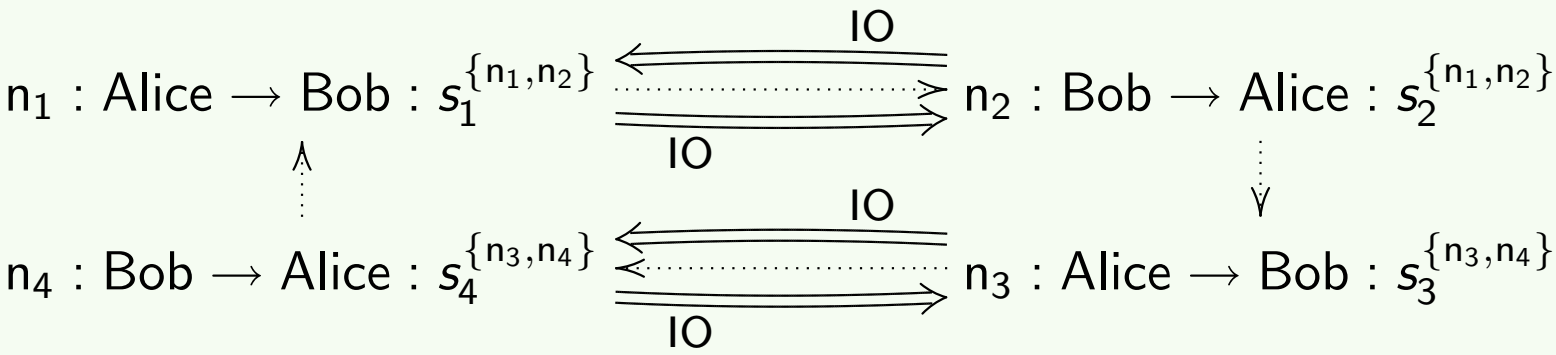
Annotation of session channels and messaging optimisation



$$\texttt{Alice} \;=\; \mu \mathbf{x}.s_1^{\{n_1,n_2\}}!\langle \text{nat} \rangle; s_2^{\{n_1,n_2\}}?\langle \text{string} \rangle; s_3^{\{n_3,n_4\}}!\langle \text{nat} \rangle; s_4^{\{n_3,n_4\}}?\langle \text{real} \rangle; \mathbf{x}$$

Examples of local refinement

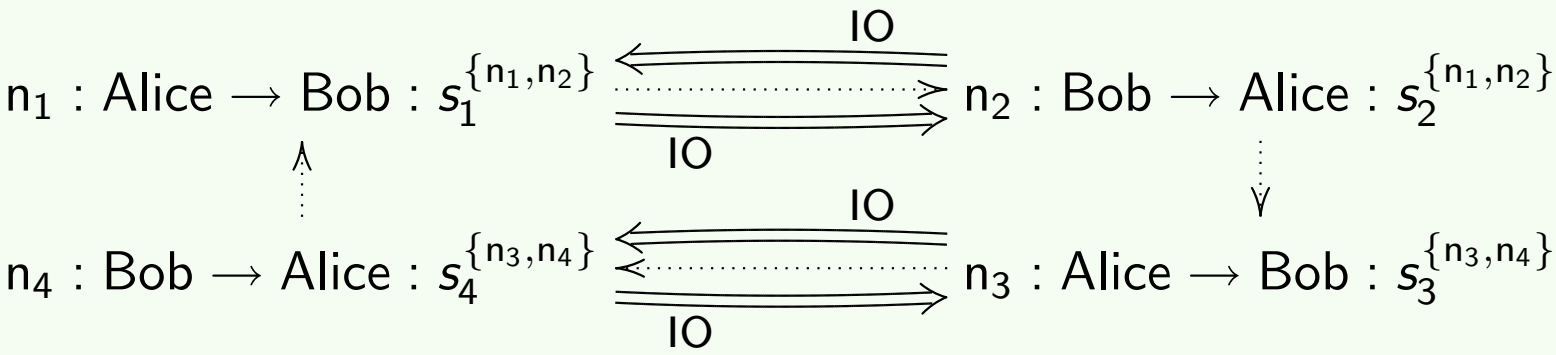
Annotation of session channels and messaging optimisation



$$\texttt{Alice} = \mu \mathbf{x}.s_1^{\{n_1,n_2\}}!\langle \text{nat} \rangle; s_2^{\{n_1,n_2\}}?\langle \text{string} \rangle; s_3^{\{n_3,n_4\}}!\langle \text{nat} \rangle; s_4^{\{n_3,n_4\}}?\langle \text{real} \rangle; \mathbf{x}$$

Examples of local refinement

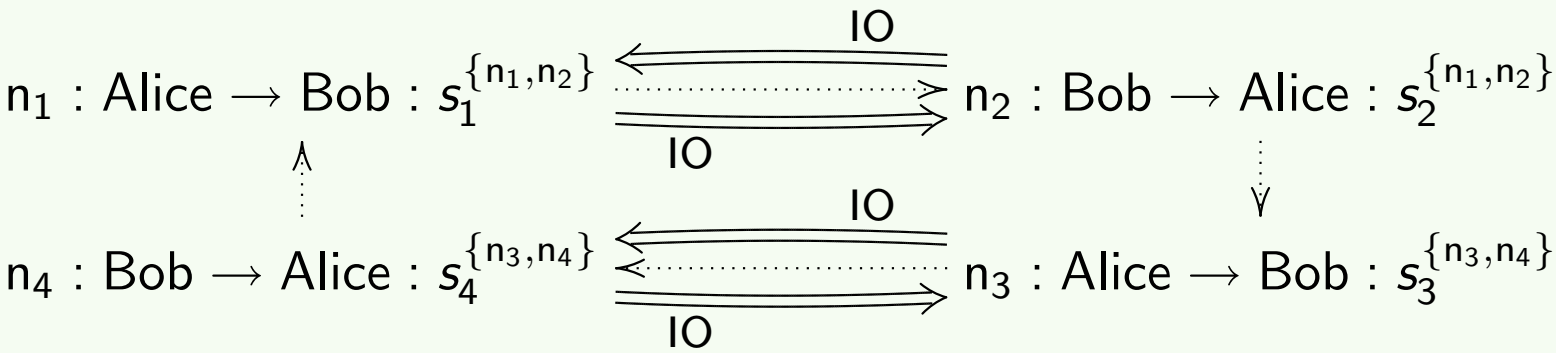
Annotation of session channels and messaging optimisation



$$\texttt{Alice} = \mu \mathbf{x}.s_1^{\{n_1,n_2\}}!\langle \text{nat} \rangle; s_3^{\{n_3,n_4\}}!\langle \text{nat} \rangle; s_2^{\{n_1,n_2\}}?\langle \text{string} \rangle; s_4^{\{n_3,n_4\}}?\langle \text{real} \rangle; \mathbf{x}$$

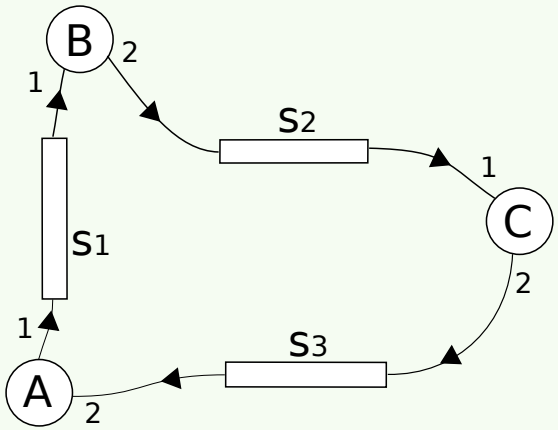
Examples of local refinement

Annotation of session channels and messaging optimisation



$$\texttt{Alice} = \mu \mathbf{x}.s_1^{\{n_1,n_2\}}!\langle \text{nat} \rangle; s_3^{\{n_3,n_4\}}!\langle \text{nat} \rangle; s_2^{\{n_1,n_2\}}?\langle \text{string} \rangle; s_4^{\{n_3,n_4\}}?\langle \text{real} \rangle; \mathbf{x}$$

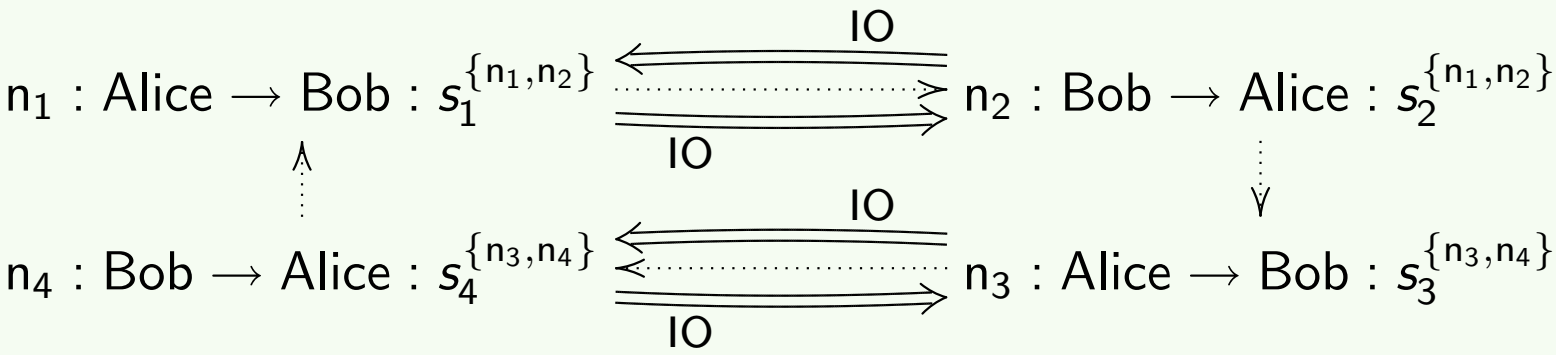
Unsafe optimisation



$$\begin{aligned} G = \mu \mathbf{x}.&\texttt{Alice} \to \texttt{Bob}: s_1 \langle \text{nat} \rangle; \\ &\texttt{Bob} \to \texttt{Carol}: s_2 \langle \text{string} \rangle; \\ &\texttt{Carol} \to \texttt{Alice}: s_3 \langle \text{real} \rangle; \mathbf{x} \end{aligned}$$

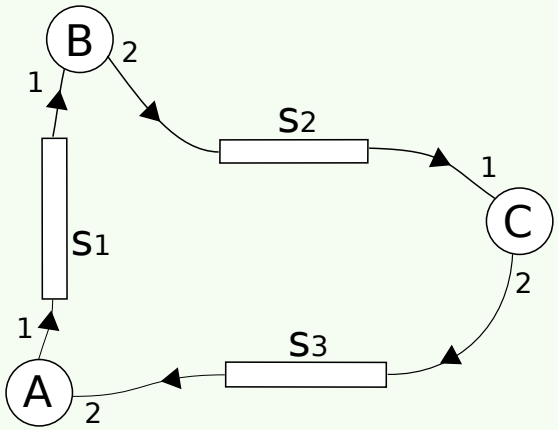
Examples of local refinement

Annotation of session channels and messaging optimisation



$$\texttt{Alice} = \mu \mathbf{x}.s_1^{\{n_1,n_2\}}!\langle \text{nat} \rangle; s_3^{\{n_3,n_4\}}!\langle \text{nat} \rangle; s_2^{\{n_1,n_2\}}?\langle \text{string} \rangle; s_4^{\{n_3,n_4\}}?\langle \text{real} \rangle; \mathbf{x}$$

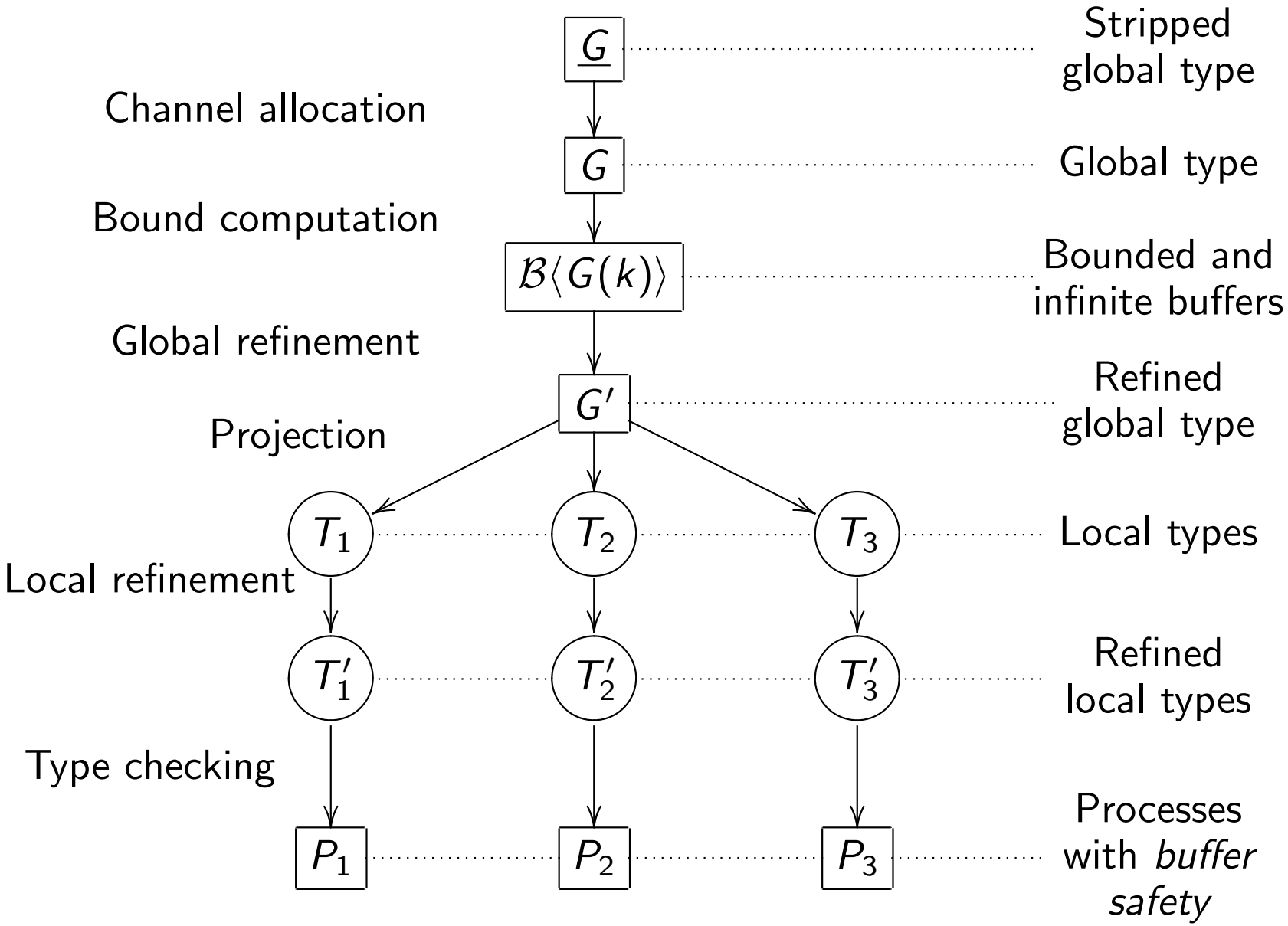
Unsafe optimisation



$$\begin{aligned} G = \mu \mathbf{x}.&\texttt{Alice} \to \texttt{Bob}: s_1\langle \text{nat} \rangle; \\ &\texttt{Bob} \to \texttt{Carol}: s_2\langle \text{string} \rangle; \\ &\texttt{Carol} \to \texttt{Alice}: s_3\langle \text{real} \rangle; \mathbf{x} \end{aligned}$$

Only the trivial partition is possible!

Overview


Channel allocation
Bound computation
Global refinement
Projection
Local refinement
Type checking
$\underline{G}$
G
$\mathcal{B}\langle G(k)\rangle$
G'
T_1
T_2
T_3
T_1'
T_2'
T_3'
P_1
P_2
P_3
Stripped global type
Global type
Bounded and infinite buffers
Refined global type
Local types
Refined local types
Processes with *buffer safety*

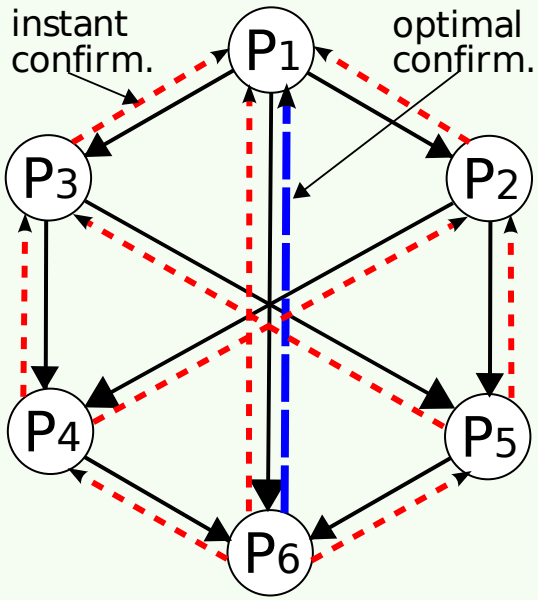
Application

- Multi-buffering algorithm [ESOP'09] which used unsafe optimisation.

Application

- Multi-buffering algorithm [ESOP'09] which used unsafe optimisation.

Multiprocessor SoC [Cheung&al. HLDTV'07]



Six processes connected by nine FIFOs

$$G = \mu \mathbf{x}.(\ \texttt{foreach}(i \in \{2,3\})\{G[i]\}; \mathrm{p}[1] \to \mathrm{p}[6] : \mathrm{on}); \mathbf{x}$$

$$G[i] = \mathrm{p}[1] \to \mathrm{p}[i] : \left\{ \begin{array}{l} \mathrm{on} : \mathrm{p}[i] \to \mathrm{p}[4] : \mathrm{on};\ \ \mathrm{p}[4] \to \mathrm{p}[6] : \mathrm{on}; \\ \mathrm{off} : \mathrm{p}[i] \to \mathrm{p}[5] : \mathrm{off};\ \ \mathrm{p}[5] \to \mathrm{p}[6] : \mathrm{off} \end{array} \right\}$$

Proposition

The above MPSoC six processes are communication-safe, buffer safe and satisfy progress (after global and local refinements).

Related Work

- Buffer analysis
  - Petri Nets, Kahn Networks [Geilen&Basten ESOP'03]
  - Binary session types [Gay&Vasconcelos JFP'09]
  - Static analysis [Nielson&Nielson POPL'94, Kobayashi&al. SAS'95, Teraushi&Megasz ESOP'08]
- Multiparty Session types
  - Multiparty Session Types [Carbone, Honda, Yoshida, POPL'08, CONCUR'08, ESOP'09]
  - Session Java [Hu, Yoshida, Honda, ECOOP'08, ECOOP'10]
- Other session type system
  - Conversation Types [Caires, Vieira, ESOP'09]
  - Contracts [Castagna, Padovani, CONCUR'09]
- Long version and prototype implementation
  - `http://www.doc.ic.ac.uk/~malo/multianalysis/`

Related Work

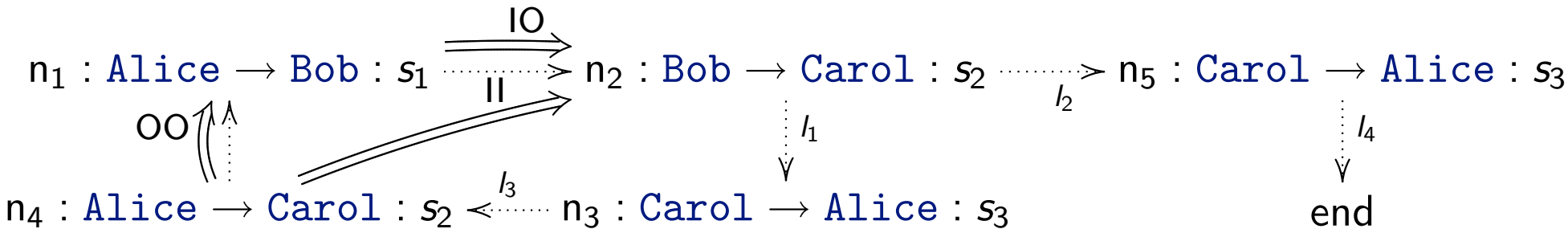
- Buffer analysis
  - Petri Nets, Kahn Networks [Geilen&Basten ESOP'03]
  - Binary session types [Gay&Vasconcelos JFP'09]
  - Static analysis [Nielson&Nielson POPL'94, Kobayashi&al. SAS'95, Teraushi&Megasz ESOP'08]
- Multiparty Session types
  - Multiparty Session Types [Carbone, Honda, Yoshida, POPL'08, CONCUR'08, ESOP'09]
  - Session Java [Hu, Yoshida, Honda, ECOOP'08, ECOOP'10]
- Other session type system
  - Conversation Types [Caires, Vieira, ESOP'09]
  - Contracts [Castagna, Padovani, CONCUR'09]
- Long version and prototype implementation
  - `http://www.doc.ic.ac.uk/~malo/multianalysis/`

Thanks!

Extra slides

- Linearity

Causality Analysis and Linearity



- OO-dependency: local sends are ordered.
- II-dependency: local receives are ordered.
- IO-dependency: reception is a guard for sends.
- Linearity property: two uses of the same channels cannot be in conflict
  The senders should be ordered by OO-chains and the receivers by II-chains.