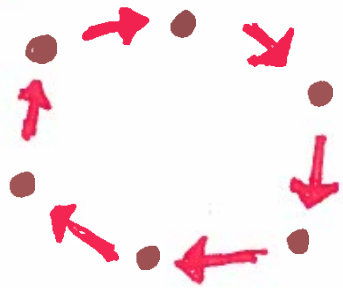
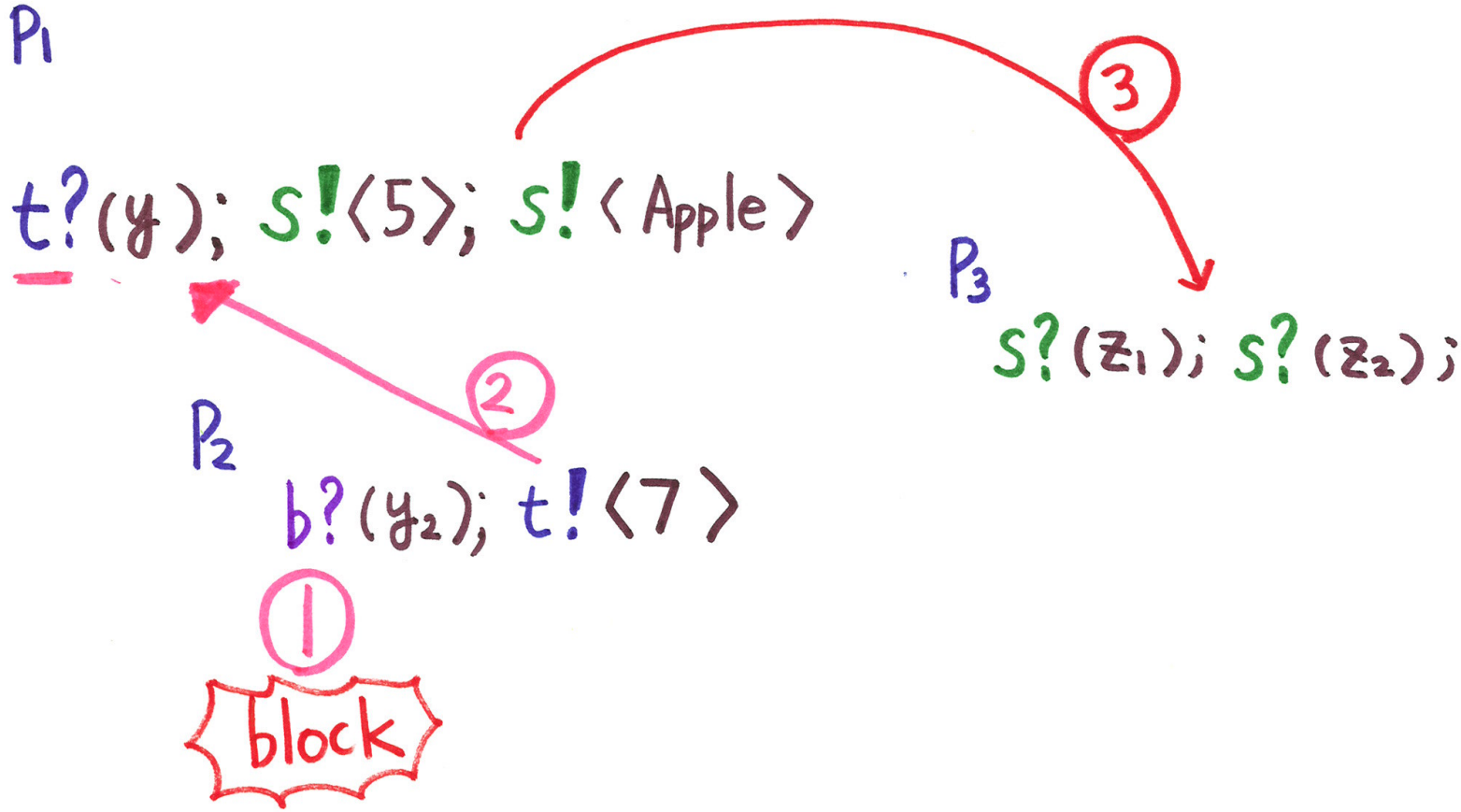
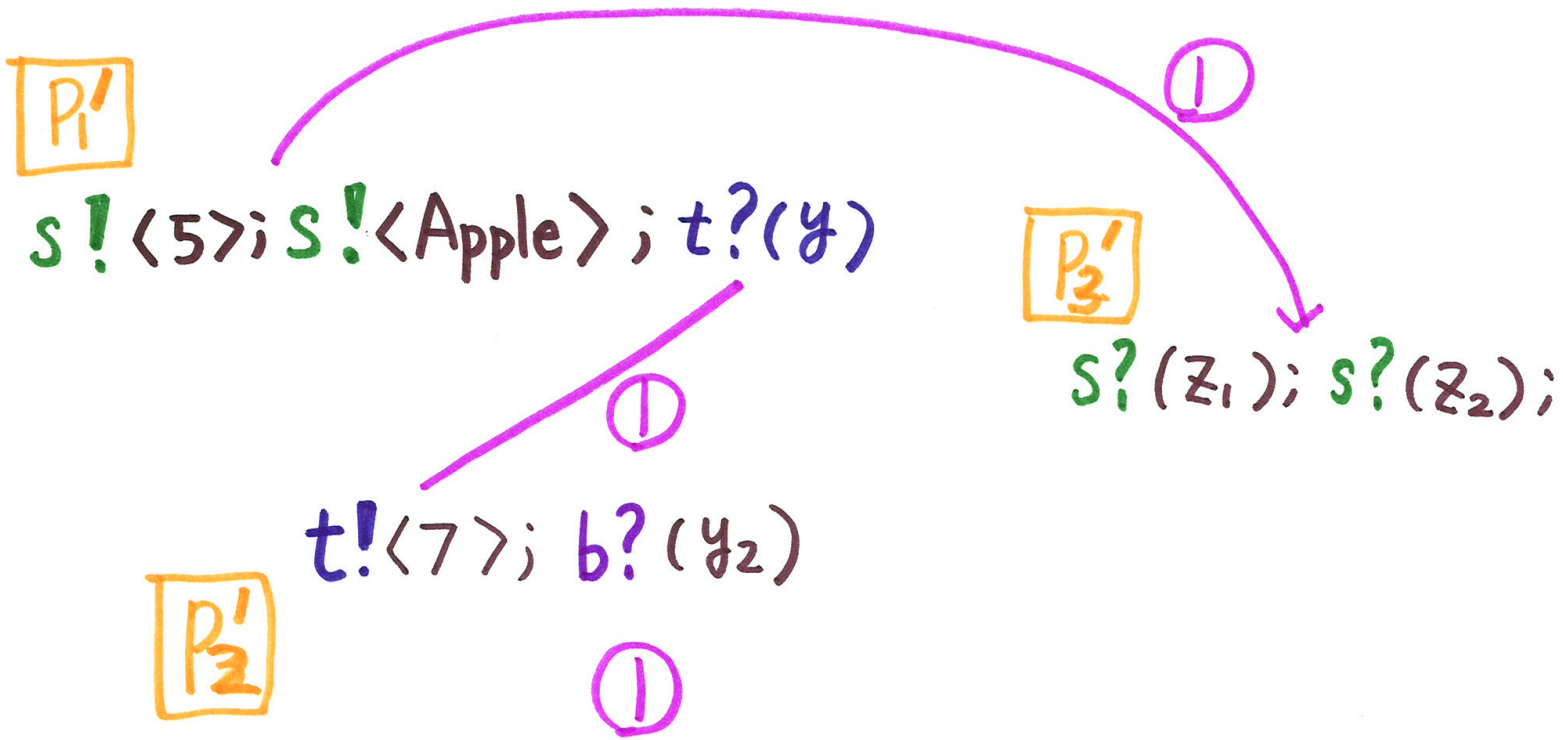
Undecidability of Asynchronous Session Subtyping

Julien Lange

Nobuko Yoshida




P1
t?(y); s!⟨5⟩; s!⟨Apple⟩
3
P3
s?(z1); s?(z2);
2
P2
b?(y2); t!⟨7⟩
1
block



Communication Subtyping

$$!\langle T\rangle; ?\langle T'\rangle \leqslant ?\langle T'\rangle; !\langle T\rangle$$

Partial Permutations

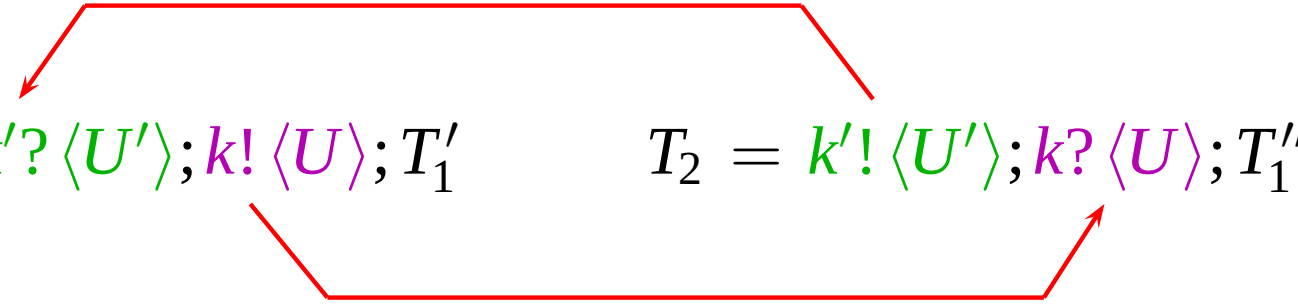
- top-level actions can be permuted using rules of $\ll$
- for example:

$$T_1 = k'?\langle U' \rangle; k!\langle U \rangle; T_1' \qquad T_2 = k'!\langle U' \rangle; k?\langle U \rangle; T_1''$$

$$T_1' = k!\langle U \rangle; k'?\langle U' \rangle; T_1' \qquad T_2 = k'!\langle U' \rangle; k?\langle U \rangle; T_1''$$

Partial Permutations

- top-level actions can be permuted using rules of $\ll$
- for example:

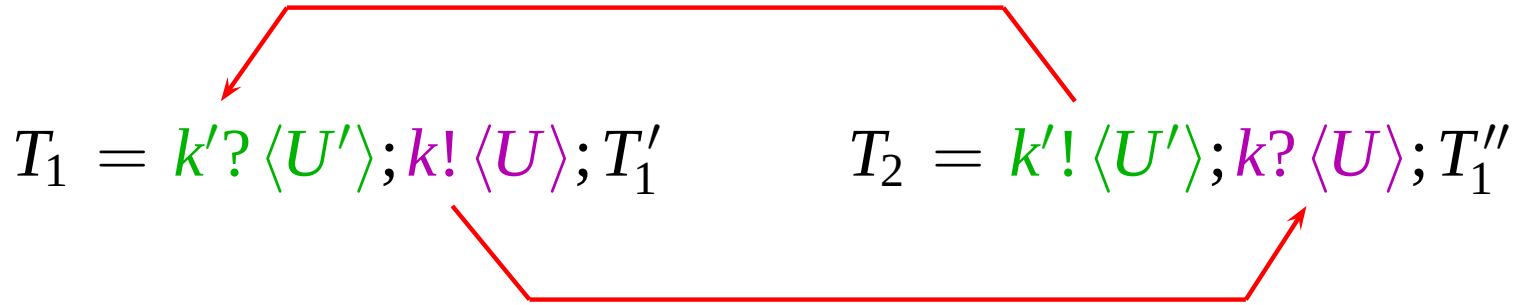


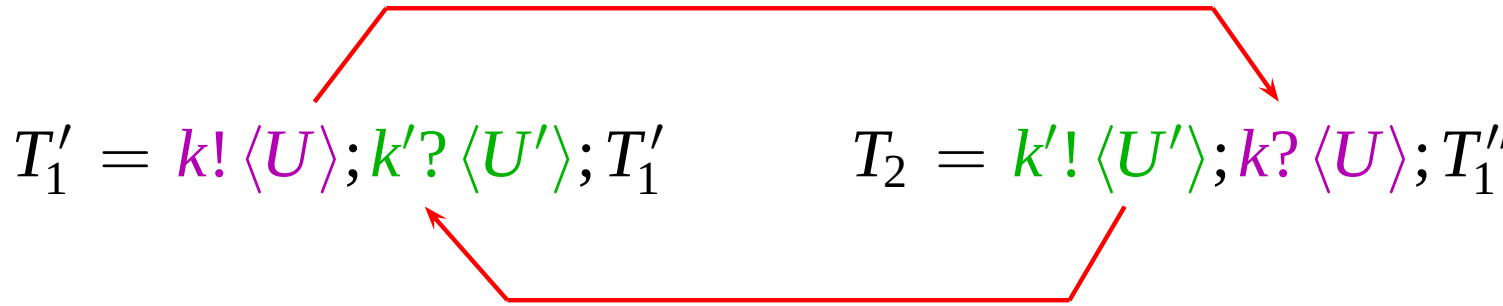
$$T_1 = k'?\langle U' \rangle; k!\langle U \rangle; T_1' \qquad T_2 = k'!\langle U' \rangle; k?\langle U \rangle; T_1''$$

$$T_1' = k!\langle U \rangle; k'?\langle U' \rangle; T_1' \qquad T_2 = k'!\langle U' \rangle; k?\langle U \rangle; T_1''$$

Partial Permutations

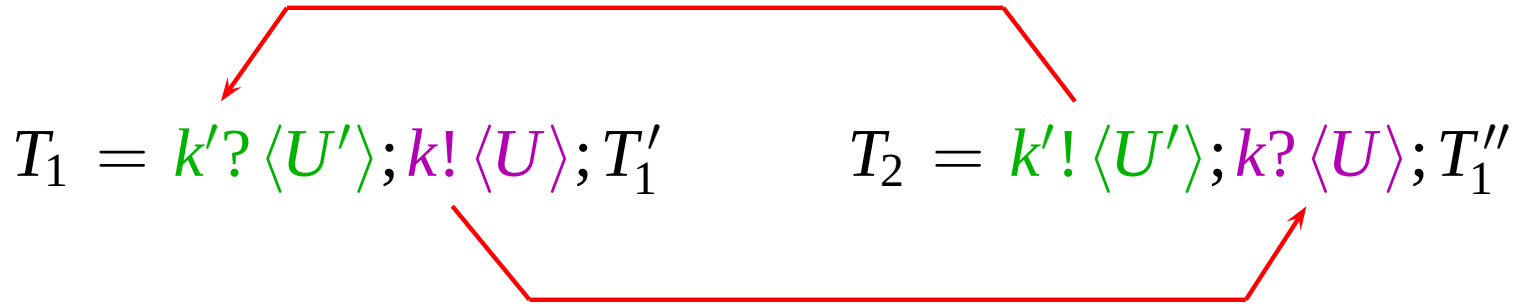
- top-level actions can be permuted using rules of $\ll$
- for example:

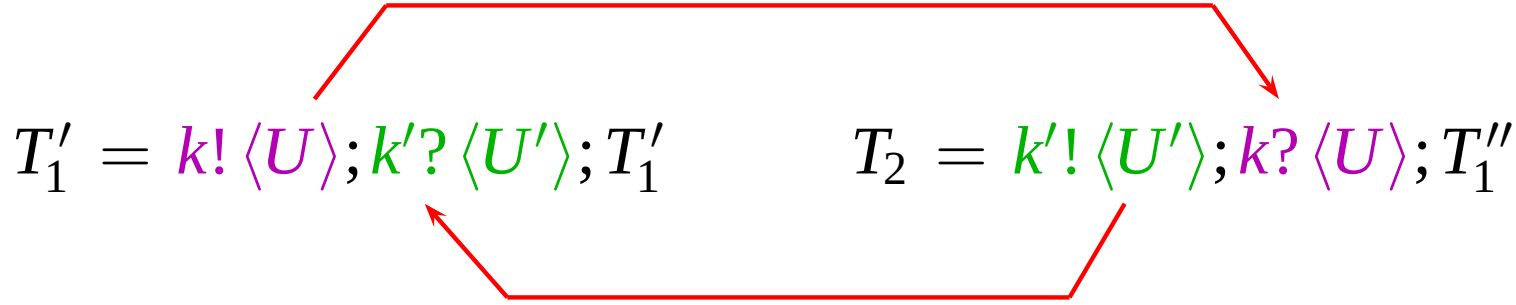




Partial Permutations

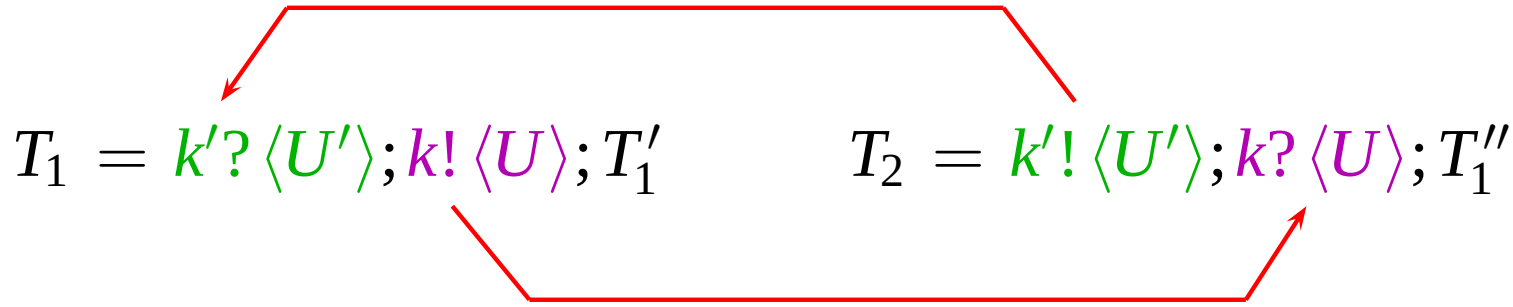
- top-level actions can be permuted using rules of $\ll$
- for example: $T_1' \ll T_1$

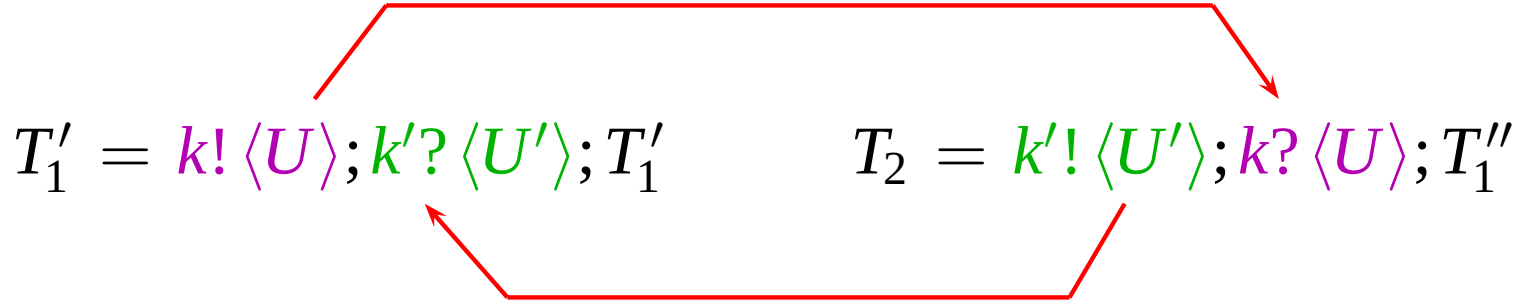




Partial Permutations

- top-level actions can be permuted using rules of $\ll$
- for example: $T_1' \ll T_1$

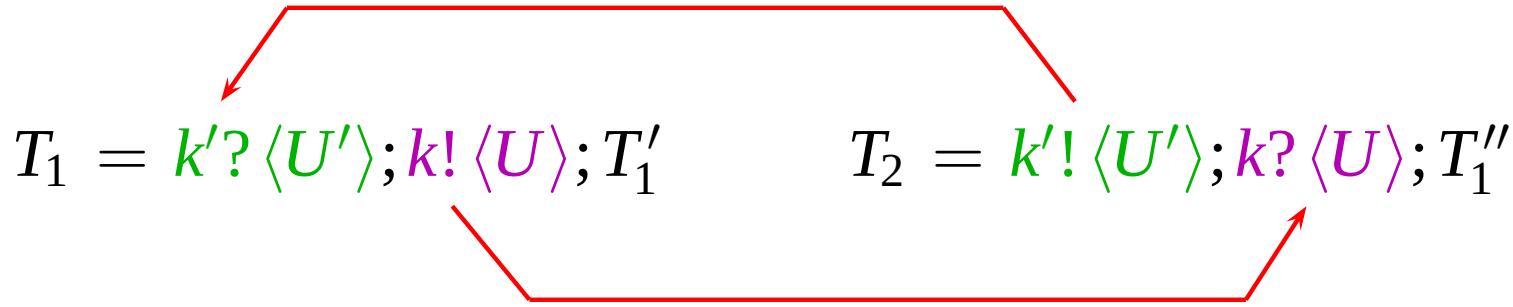


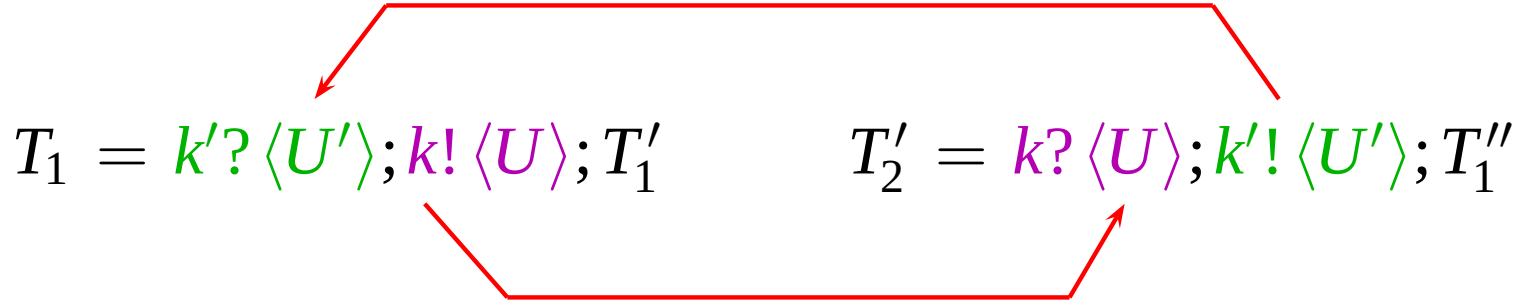


(OI) $\quad k!\langle U \rangle; k'?\langle U' \rangle; T \quad \ll \quad k'?\langle U' \rangle; k!\langle U \rangle; T$

Partial Permutations

- top-level actions can be permuted using rules of $\ll$
- for example: $T_1' \ll T_1$ $T_2' \not\ll T_2$





No progress

History (1)

2007 Scribble (70 pages) (Kohei Honda & Gary Brown)

2008 FMCO Multicore Programming (NY, Vasconcelos, Paulino, Honda)

2009 ESOP'09 (Mostrous, NY and Honda) Multiparty

2010 TLCA'10 (Mostrous, NY) HOπ

2010 ⟺ 2015 Mostrous Thesis HOπ (Info. Comp)

2012 TOOLS'12 Session C (Ng, NY, Honda)

History (1)

2007 Scribble (70 pages) (Kohei Honda & Gary Brown)

2008 FMCO Multicore Programming (NY, Vasconcelos, Paulino, Honda)

2009 ESOP'09 (Mostrous, NY and Honda)

2010 TLCA'10 (Mostrous, NY) HO π

2010 ⇔ 2015 Mostrous Thesis

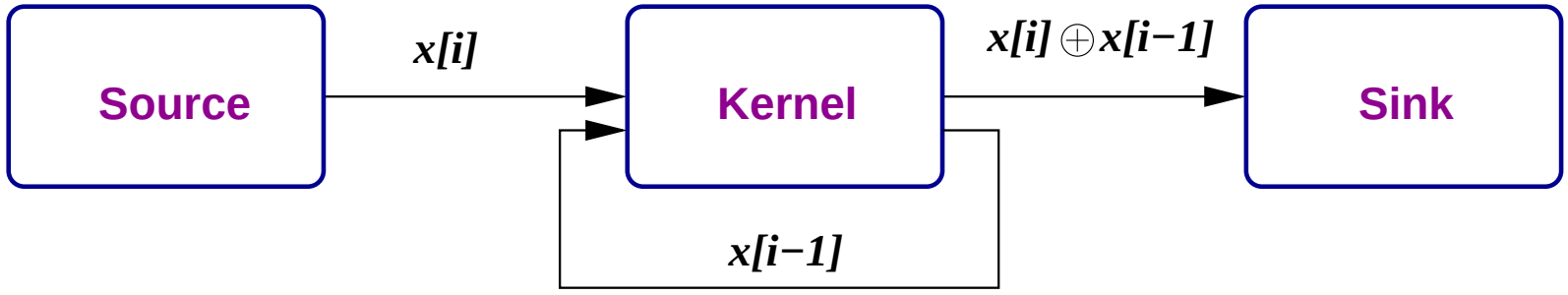
2012 TOOLS'12 Session C (Ng, NY, Honda)

① ESOP Subtyping

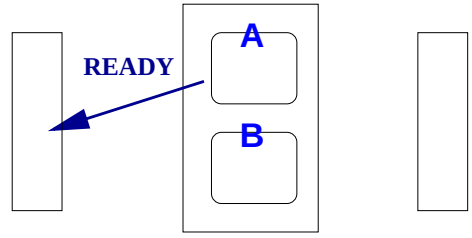
② Mostrous Comp

Example: Double Buffering Algorithm

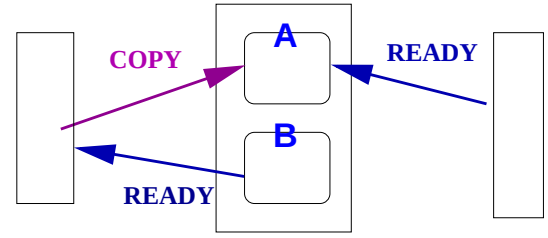
- Optimisation by overlapping computation and communication
- Source — Kernel — Sink
  - Source sends data to Kernel
  - Kernel computes on data
  - Kernel sends to Sink
  - Use of 2 buffers at Kernel allows Sink to write in one while Sink reads from the other.



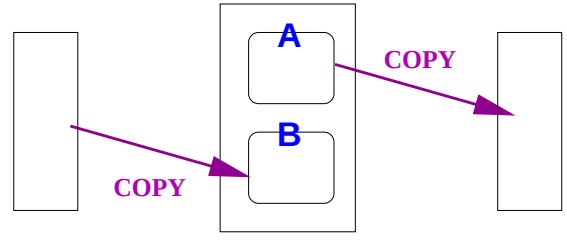
Example: Double Buffering Algorithm



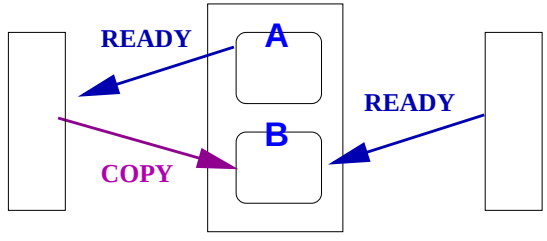
(a)



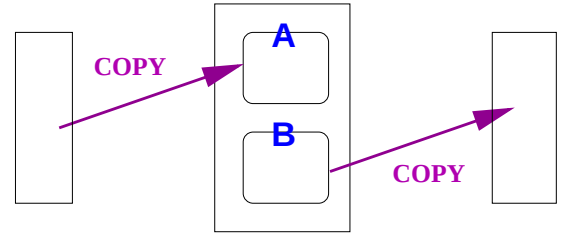
(b)



(c)



(d)



(e)

Types for Double Buffering

- **Original Local Types**

$$
\begin{aligned}
T_{\mathsf{source}} = \quad & \mu \mathbf{t}.r_1?\langle\rangle; s_1!\langle U\rangle; r_2?\langle\rangle; s_2!\langle U\rangle; \mathbf{t} \\
T_{\mathsf{kernel}} = \quad & \mu \mathbf{t}.r_1!\langle\rangle; s_1?\langle U\rangle; t_1?\langle\rangle; u_1!\langle U\rangle; \\
& \quad r_2!\langle\rangle; s_2?\langle U\rangle; t_2?\langle\rangle; u_2!\langle U\rangle; \mathbf{t} \\
T_{\mathsf{sink}} = \quad & \mu \mathbf{t}.t_1!\langle\rangle; u_1?\langle U\rangle; t_2!\langle\rangle; u_2?\langle U\rangle; \mathbf{t}
\end{aligned}
$$

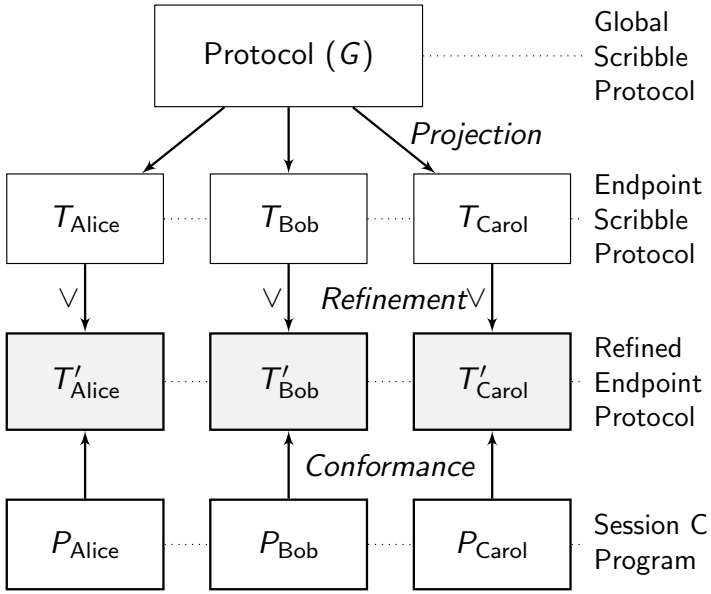
- **Optimized Kernel Type**

$$
\begin{aligned}
T_{\mathsf{opt}} = \quad & r_1!\langle\rangle; r_2!\langle\rangle; \mu \mathbf{t}.s_1?\langle U\rangle; t_1?\langle\rangle; u_1!\langle U\rangle; r_1!\langle\rangle; \\
& \quad s_2?\langle U\rangle; t_2?\langle\rangle; u_2!\langle U\rangle; r_2!\langle\rangle; \mathbf{t}
\end{aligned}
$$

- **Theorem** $T_{\mathsf{opt}} \leqslant_c T_{\mathsf{kernel}}$

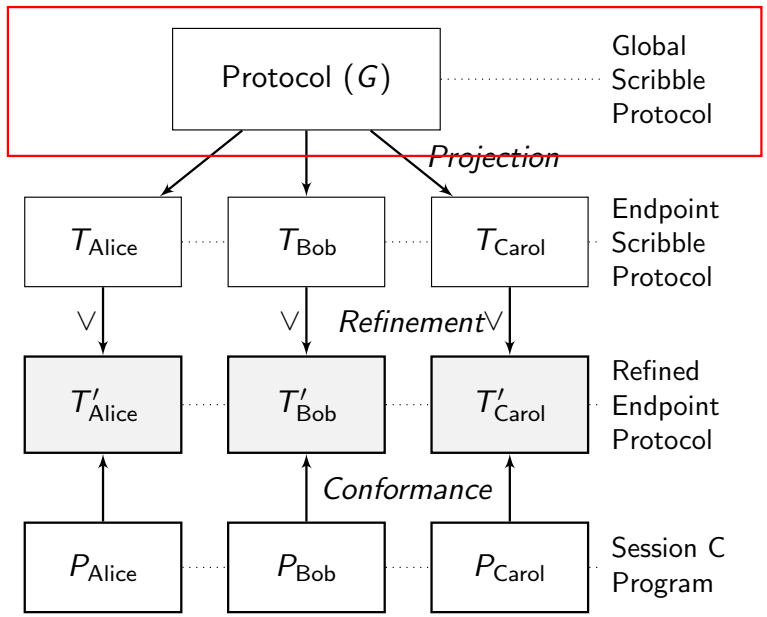
Session C programming: Overview

- Top down approach
- Based on multiparty session types (MPST) [Honda et al., POPL'08]
  - Communication should have a dual
  - Communication safety and deadlock freedom by typing



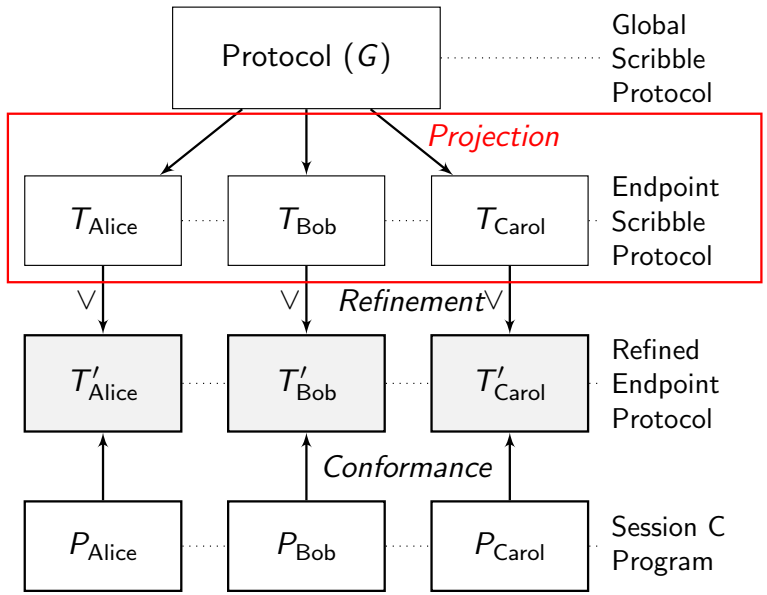
Session C programming: Key reasoning

1. Design protocol in global view
2. Automatic *projection* to endpoint protocol, algorithm preserves safety
3. Write program according to endpoint protocol
4. Check program conforms to protocol
5. $\Rightarrow$ Safe program by design



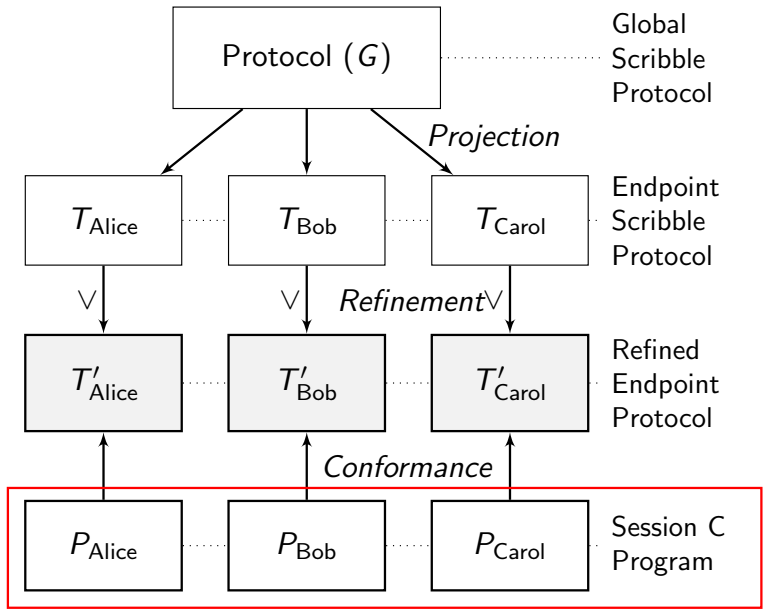
Session C programming: Key reasoning

1. Design protocol in global view
2. Automatic *projection* to endpoint protocol, algorithm preserves safety
3. Write program according to endpoint protocol
4. Check program conforms to protocol
5. $\Rightarrow$ Safe program by design



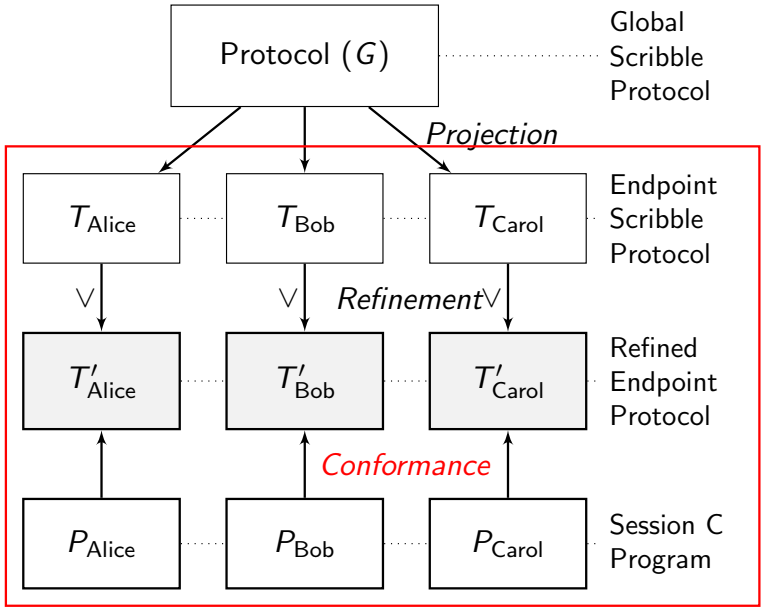
Session C programming: Key reasoning

1. Design protocol in global view
2. Automatic *projection* to endpoint protocol, algorithm preserves safety
3. Write program according to endpoint protocol
4. Check program conforms to protocol
5. $\Rightarrow$ Safe program by design



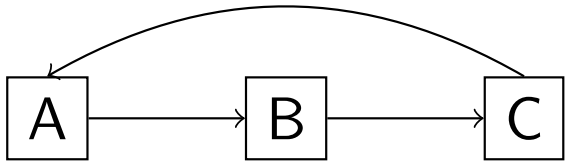
Session C programming: Key reasoning

1. Design protocol in global view
2. Automatic *projection* to endpoint protocol, algorithm preserves safety
3. Write program according to endpoint protocol
4. Check program conforms to protocol
5. $\Rightarrow$ Safe program by design



Asynchronous optimisation

- Asynchronous operations can be **safely** permuted [Mostrous et al., ESOP'09]
- Pipelines inefficient in 'strict' multiparty session types
- Efficient pipelines with asynchronous subtyping of MPST



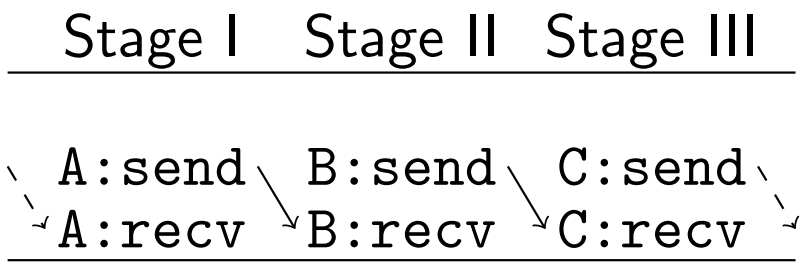
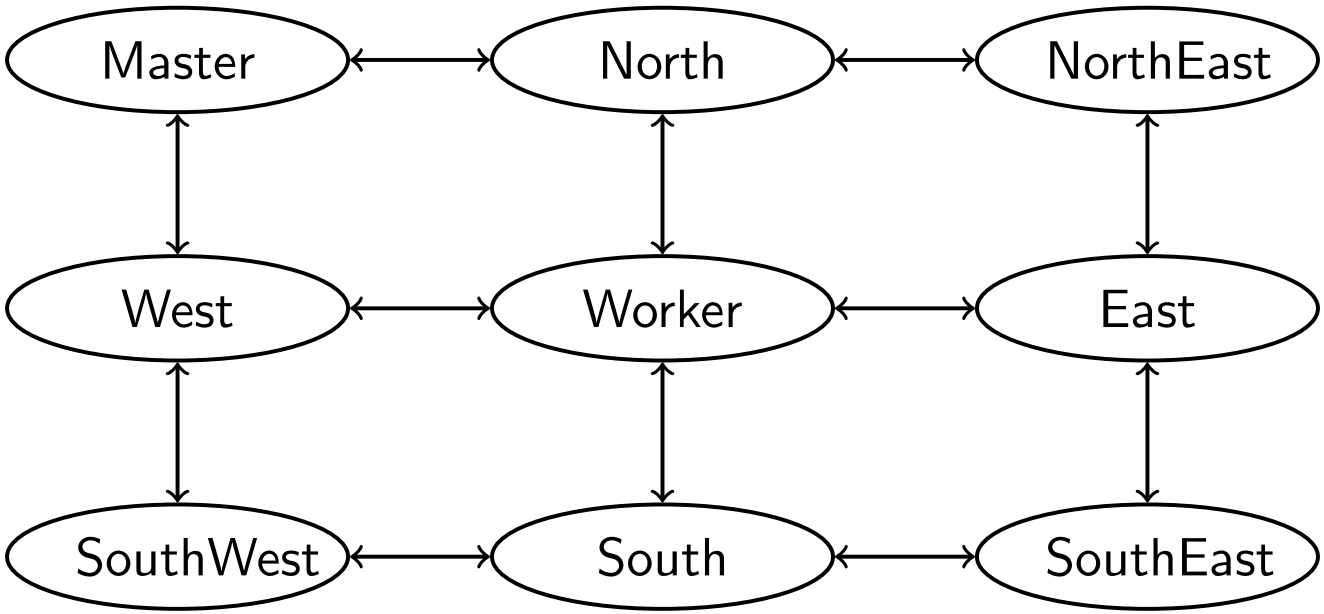


Figure: Asynchronous subtyping.

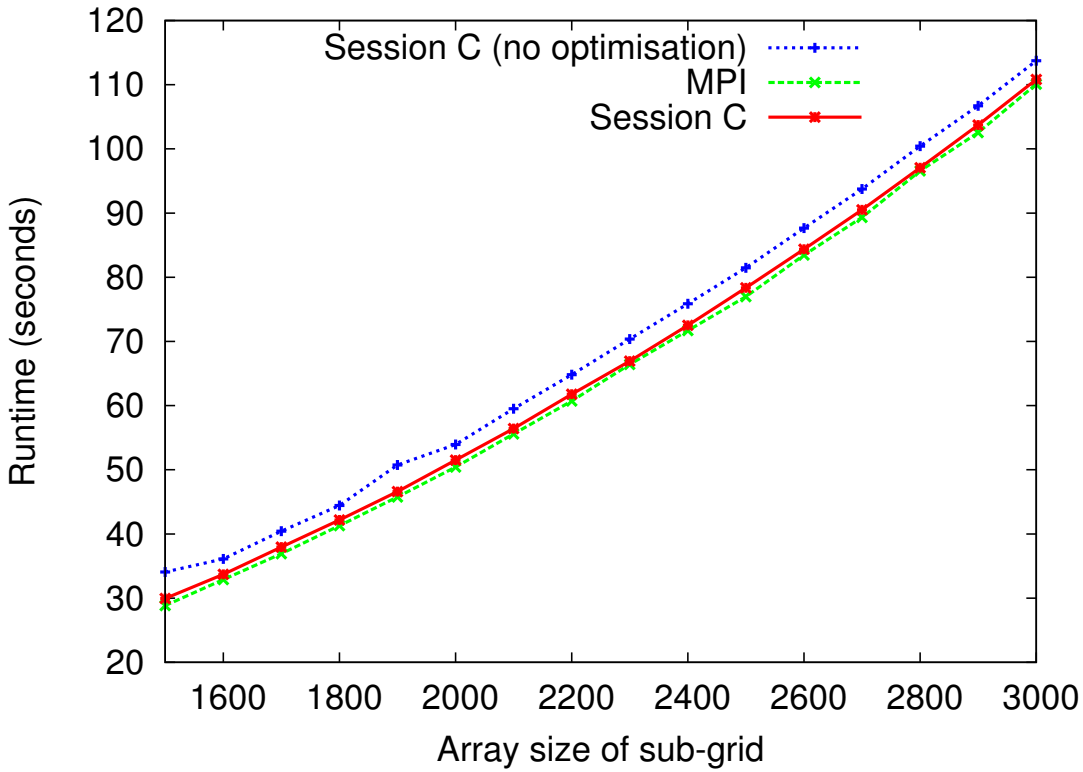
Jacobi solution for the DPE: Mesh topology

- Input segmented to 2D sub-grids
- Edge results exchanged between each neighbours
- Takes full advantage of asynchronous message optimisation



Benchmark results: highlight

- Jacobi method for the Discrete Poisson Equation
- Mesh topology
- Asynchronous optimisation: 8% improvement



2014 PPDP'14 Precise Subtyping (Chen Dezani

③ Precise

2014 – 15 STSM BETTY by Alceste Scalas

2015 LMCS (Chen, Dezani, A.Scalas, NY)

2016 ③ STSM BETTY by Scalas & NY

2016 ⑦ RC Meeting NY meets Bravetti & Zavattaro

2016 ⑨ UNDECIDABLE (Lange & Yoshida)

2016 ⑫ Accepted by FoSSaCs

Theorem 6.1 (Denotational preciseness)**.** *The synchronous and the asynchronous subtyping relations are denotationally precise for the synchronous calculus and the asynchronous calculus, respectively.*

7. Related Work

Preciseness To the best of our knowledge operational preciseness was first defined in [3] for a call-by-value λ-calculus with recursive functions, pairs and sums. In that paper the authors show that the iso-recursive subtyping induced by the Amber rules [5] is incomplete. They propose a new iso-recursive subtyping which they prove to be precise.

Operational and denotational preciseness are shown in [9] for the concurrent λ-calculus with intersection and union types introduced in [10]. In that paper divergence plays the rôle of reduction to `error`.

Preciseness in concurrency is more useful and challenging than in the functional setting, since there are many interesting choices for the syntax, semantics, type errors of the calculi and for the typing systems. A similar situation appears in the study of bisimulations where many labelled transition relations can be defined. It is now common that researchers justify the correctness of labelled transition systems by proving that the bisimulation coincides with the contextual congruence [20, 26]. Our claim is that preciseness should become a sanity check for subtypings.

Choices of subtypings The first branching-selection subtyping for the session types was proposed in [12] and used for example in [6, 7, 31, 36]. That subtyping is the opposite of the current synchronous subtyping, since branch is covariant and selection is contravariant in the set of labels. We can establish the preciseness of the subtyping in [12] for the synchronous calculus if we reverse the ordering both in the preciseness definition and in the extension of subtyping to session environments. We have chosen the subtyping used in [4, 8, 28, 29] since we claim it fits better with the preciseness definition. In fact the approach of [12] corresponds to safe substitutability of channels, while the approach of [29] corresponds to safe substitutability of processes. The subtyping of [4, 8, 28, 29] differs from ours since the types of exchanged values are invariant.

Other completeness results Subtyping of recursive types requires algorithms for checking subtype relations, as discussed in [32, Chapter 21]. These algorithms need to be proved sound and complete with respect to the definition of the corresponding subtyping, as done for example in [7, 12, 33]. Algorithms for checking the synchronous and asynchronous subtypings of the present paper can be easily designed.

Several works on subtyping formulate the errors using typed reductions or type environments (e.g. [17, 33]), and they prove soundness with respect to the typed reductions and their erasure theorems. In contrast with these approaches, our error definitions in Tables 4 and 12 do not rely on any type-case construct or explicit type information, but are defined *syntactically* over untyped terms. Note that once the calculus is annotated by type information or equipped with type case, completeness becomes trivial, since any two processes of incomparable types can be operationally distinguished.

Semantic subtyping In the semantic subtyping approach each type is interpreted as the set of values having that type and subtyping is subset inclusion between type interpretations [11]. This gives a precise subtyping as soon as the calculus allows to distinguish operationally values of different types. Semantic subtyping has been studied in [6] for a π-calculus with a patterned input and in [7] for a session calculus with internal and external choices and typed input. Types are built using a rich set of type constructors including union, intersection and negation: they extend IO-types in [6] and session types in [7]. Semantic subtyping is precise for the calculi of [6, 7, 11], thanks to the type case constructor in [11], and to the blocking of inputs for values of "wrong" types in [6, 7].

Subtyping of Mostrous Note that our subtyping relation differs from that defined in [27] only for the premises $\& \in \mathscr{A}$ and $\& \in T_i$ in rule [SUB-PERM-ASYNC]. As a consequence T is a subtype of S when $T = \mu\mathbf{t}.!l\langle T'\rangle.\mathbf{t}$ and $S = \mu\mathbf{t}.!l\langle T'\rangle.?l'(S').\mathbf{t}$ (see [27, p. 116]). This subtyping is not sound in our system: intuitively T accumulates infinite orphan messages in a queue, while S ensures that the messages are eventually received. The subtyping relation in [27] unexpectedly allows an unsound process (typed by T) to act as if it were a sound process (typed by S). Let $C = (\nu ab)([\,] \mid Q \mid ab\blacktriangleright\varnothing \mid ba\blacktriangleright\varnothing)$ where

$$Q = \mathtt{def}\ Y(x) = b!l\langle x\rangle.b?l'(y).Y\langle x\rangle\ \mathtt{in}\ (\nu cc')(Y\langle c\rangle).$$

Then we can derive $C[a : S] \triangleright \emptyset$. Let

$$P = \mathtt{def}\ Z(z) = a!l\langle z\rangle.Z\langle z\rangle\ \mathtt{in}\ (\nu dd')(Z\langle d\rangle).$$

Then $P \triangleright \{a : T\}$. We get

$$C[P] \to_{\mathsf{a}}^{*} (\nu ab)(\nu cc')(P \mid Q \mid ab\blacktriangleright\varnothing \mid ba\blacktriangleright l\langle c\rangle) \to_{\mathsf{a}} \mathtt{error}$$

by rule [ERR-ORPH-MESS-ASYNC], since $a \notin \varphi(P \mid Q \mid ab\blacktriangleright\varnothing)$ and $\mathsf{fpv}(P \mid Q \mid ab\blacktriangleright\varnothing) = \emptyset$.

The subtyping of [27] is sound for the session calculus defined there, which does not consider orphan messages as errors. However, the subtyping of [27] is not complete, an example being $\mu\mathbf{t}.!l\langle T\rangle.\mathbf{t} \not\leqslant \mu\mathbf{t}.?l'(S).\mathbf{t}$. There is no context C which is safe for all processes with one channel typed by $\mu\mathbf{t}.?l'(S).\mathbf{t}$ and no process P with one channel typed by $\mu\mathbf{t}.!l\langle T\rangle.\mathbf{t}$ such that $C[P]$ deadlocks.

8. Conclusion

This paper gives, as far as we know, the first formulation and proof techniques for the preciseness of subtyping in mobile processes. We consider the synchronous and asynchronous session calculi to investigate the preciseness of the existing subtypings. While the well-known branching-selection subtyping [4, 8, 12] is precise for the synchronous calculus, the subtyping in [27] turns out to be not sound for the asynchronous calculus. We propose a simplification of previous asynchronous subtypings [28, 29] and prove its preciseness. As a matter of fact only soundness is a consequence of subject reduction, while completeness can fail also when subject reduction holds.

Our calculus lacks the session initialisation as well as the communication of expressions which are present in the original calculus [21]. These extensions are straightforward and we can obtain the same preciseness results. First, for the extension to session initialisation, we just need to add a negation of the subtyping relation over the shared channel type ($\langle T, \overline{T}\rangle$ in [21]), and define its corresponding session initialisation process as a characteristic process for a shared channel type. The definitions of deadlock/error processes remain the same. Second, for the extension to expressions (e.g. $\mathtt{succ}(n)$) and to ground types (e.g. `nat`), we require contravariance of input and covariance of output for ground types and we add constructors distinguishing values of different ground types (e.g. `succ` can be applied to a value of type `nat` but not to a value of type `bool`). We then use these constructors in building characteristic processes following [3].

The formulation of preciseness along with the proof methods and techniques could be useful to examine other subtypings and calculi. Our future work includes the applications to higher-order processes [27, 28], polymorphic types [15], fair subtypings [31] and contract subtyping [1]. We plan to use the characteristic processes in typecheckers for session types. More precisely the error messages can show processes of given types when type checking

2014 PPDP'14 Precise Subtyping (Chen, Dezani)

2014 – 15 STSM BE (Unsuccessful) este Scalas

2015 LMCS (Chen, [illegible]ni, A.Scalas, NY)

2016 ③ STSM BETTY by Scalas & NY

2016 ⑦ RC Meeting NY meets Bravetti & Zavattaro

2016 ⑨ UNDECIDABLE (Lange & Yoshida)

2016 ⑫ Accepted by FoSSaCs

③ Precise

2014 PPDP'14 Precise Subtyping (Chen, Dezani)

3 Precise

2014 – 15 STSM BE (Unsuccessful) este Scalas

2015 LMCS (Chen, [illegible], A.Scalas, NY)

STSM BET (Synchronous TACAS'16) las & NY

(Unsuccessful)

RC Meeting (Julien NY) Bravetti & Zavattaro

2016 (9) UNDECIDABLE (Lange & Yoshida)

2016 (12) Accepted by FoSSaCs

Asynchronous Subtyping (② Mostrous vs ③ Precise)

$$\frac{\forall i \in I \quad T_i \leqslant U_i}{\bigoplus_{i \in I} !a_i.\, T_i \leqslant \bigoplus_{i \in I \cup J} !a_i.\, U_i}$$

MORE

$$\frac{\forall i \in I \quad T_i \leqslant U_i}{\&_{i \in I \cup J}\, ?a_i.\, T_i \leqslant \&\, ?a_i.\, U_i}$$

CONTEXT

$$\frac{\forall i \in I \quad T_i \leqslant A[U_i^n]^n \qquad \& \in T_i}{\bigoplus_{i \in I} !a_i.\, T_i \leqslant A[\bigoplus_{I \cup J_n} !a_i.\, U_i^n]^n}$$

$$\frac{}{\text{end} \leqslant \text{end}}$$

CONTEXT $\quad A = [\,]^n \mid \&_{i \in I}\, ?a_i.\, A_i$

Asynchronous Subtyping (② Mostrous vs ③ Precise)

$$\frac{\forall i \in I \quad T_i \leqslant U_i}{\bigoplus_{i \in I} !a_i.\, T_i \leqslant \bigoplus_{i \in I \cup J} !a_i.\, U_i} \qquad \frac{\forall i \in I \quad T_i \leqslant U_i}{\&_{i \in I \cup J} ?a_i.\, T_i \leqslant \&_{i \in I \cup J} ?a_i.\, U_i}$$

MORE

② No Condition

$$\frac{\forall i \in I \quad T_i \leqslant A[U_i^n]^n}{\bigoplus_{i \in I} !a_i.\, T_i \leqslant A[\bigoplus_{I \cup J_n} !a_i.\, U_i^n]^n} \qquad \frac{}{\mathrm{end} \leqslant \mathrm{end}}$$

CONTEXT

$$\text{CONTEXT} \quad A = [\,]^n \mid \&_{i \in I} ?a_i.\, A_i$$

Mostrous Subtyping ②

$$\mu t.\, !\langle T\rangle.\, t \;\leqslant\; \mu t.\, !\langle T\rangle.\, ?\langle T'\rangle.\, t$$

No Input

Precise Subtyping ③

$$\mu t.\, !\langle T\rangle.\, t \;\not\leqslant\; \mu t.\, !\langle T\rangle.\, ?\langle T\rangle.\, t$$

preserving orphan message freedom

Our Paper [FoSSaCS'17] via CFSM [1980]

STEP 1 Introduce Asynchronous Duplex System which strictly includes Half Duplex System [2005]

STEP 2 Define Compatible Relation $M_1 \asymp M_2$ on CFSMs and show $\asymp$ is soundly and completely characterise safe AD

STEP 3 Prove AD is Turing Complete

STEP 4 Relate $\asymp$ and $\lesssim$

$$M_1 \asymp M_2 \iff T(M_1) \lesssim \overline{T(M_2)}$$

Our Paper [FoSSaCS'17] via CFSM [1980]

STEP 1 Introduce Asynchronous Duplex System which strictly includes Half Duplex System [2005]

STEP 2 Define Compatible Relation $M_1 \asymp M_2$ on CFSMs and show $\asymp$ is soundly and completely characterise safe AD

STEP 3 Prove AD is Turing Complete

STEP 4 Relate $\asymp$ and $\lesssim$

$$M_1 \asymp M_2 \iff T(M_1) \lesssim \overline{T(M_2)}$$

Safe ① No Deadlock
② No Orphan Messages

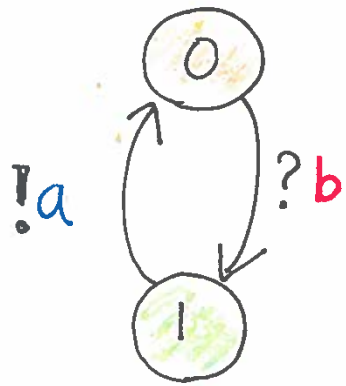
General CFSMs

Session CFSMs
- Deterministic
- No Mixed
- Directed

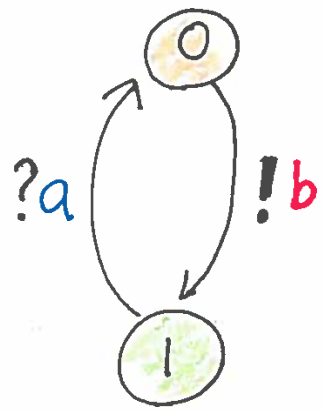
✓ Our approach gives Semantics for $\leqslant_s \leqslant_a \leqslant_M$

✓ Extendable to CFSMs without Session Syntax Limitation

HD

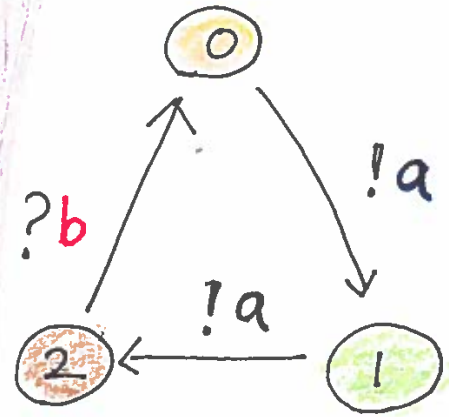


M_2

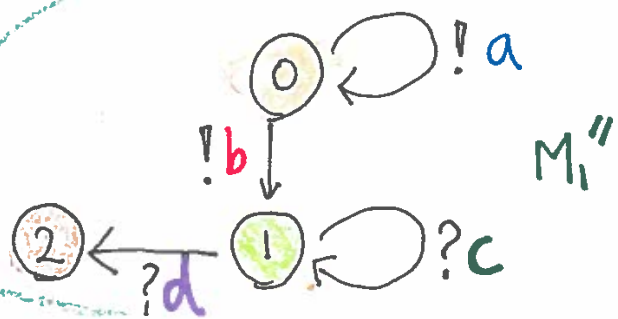


M_1

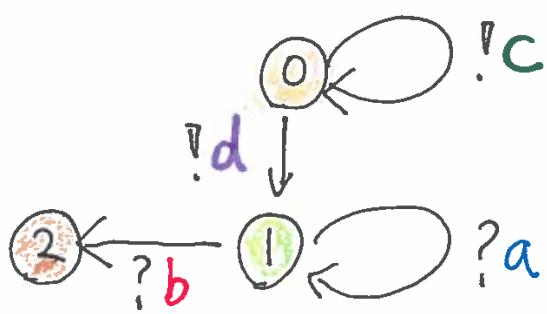
AD



M_2'



M_1''



M_2''

NOT AD

Side Theorem. (Transitivity of $\asymp$ in CFSMs)

$$M_1 \asymp M \;\wedge\; M \asymp M_2 \;\Rightarrow\; M_1 \asymp M_2$$

because

$$M_1 \leq \overline{M} \;\wedge\; \overline{M} \leq M_2 \;\Rightarrow\; M_1 \leq M_2 \quad \text{by [PPDP'14]}$$