The impact of asynchronous communication on the theory of contracts and session types

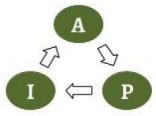
Gianluigi Zavattaro

Department of Computer Science
University of Bologna

INRIA research team FOCUS

Based on joint work with:
Mario Bravetti, Marco Carbone, Nobuko Yoshida, Julien Lange

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 778233



BEHAPI
Behavioural Application Program Interfaces

The impact of asynchronous communication on the theory of contracts and session types

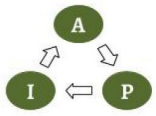
Gianluigi Zavattaro

Department of Computer Science
University of Bologna

INRIA research team FOCUS

Based on joint work with:
Mario Bravetti, Marco Carbone, Nobuko Yoshida, Julien Lange

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 778233



BEHAPI
Behavioural Application Program Interfaces

Structure of the course

- Session types: subtyping
- Behavioural contracts: CFSMs, BPA
- The synchronous case
  - Session subtyping ≈ Contract refinement
- Asynchronous communication
  - Asynchronous session subtyping
  - Async. subtyping ≈ Async. refinement
  - Positive and Negative implications

Session types [ESOP98]

Language Primitives and Type Discipline for Structured Communication-Based Programming

KOHEI HONDA*, VASCO T. VASCONCELOS†, AND MAKOTO KUBO‡

- A type discipline for communication-based **concurrent** programming

$$S ::= \mathbf{nat} \mid \mathbf{bool} \mid \langle \alpha, \overline{\alpha} \rangle \mid s \mid \mu s.S$$

$$\alpha ::= \downarrow[\tilde{S}];\alpha \mid \downarrow[\alpha];\beta \mid \&\{l_1 : \alpha_1, \ldots, l_n : \alpha_n\} \mid \mathbf{1} \mid \bot$$
$$\mid \uparrow[\tilde{S}];\alpha \mid \uparrow[\alpha];\beta \mid \oplus\{l_1 : \alpha_1, \ldots, l_n : \alpha_n\} \mid t \mid \mu t.\alpha$$

(Simple) session types

- A **minimal** syntax for session types:

$$T \quad ::= \quad \oplus\{l_i : T_i\}_{i \in I} \quad | \quad \&\{l_i : T_i\}_{i \in I} \quad | \quad \mu \mathbf{t}.T \quad | \quad \mathbf{t} \quad | \quad \mathbf{end}$$

(Simple) session types

- A **minimal** syntax for session types:

$$T \quad ::= \quad \oplus\{l_i : T_i\}_{i \in I} \quad | \quad \&\{l_i : T_i\}_{i \in I} \quad | \quad \mu \mathbf{t}.T \quad | \quad \mathbf{t} \quad | \quad \mathbf{end}$$

Selection among **outputs**

(Simple) session types

- A **minimal** syntax for session types:

$$T ::= \oplus\{l_i : T_i\}_{i \in I} \mid \&\{l_i : T_i\}_{i \in I} \mid \mu \mathbf{t}.T \mid \mathbf{t} \mid \mathbf{end}$$

Selection among **outputs**

Branching among **inputs**

(Simple) session types

- A **minimal** syntax for session types:

$$T ::= \oplus\{l_i : T_i\}_{i \in I} \mid \&\{l_i : T_i\}_{i \in I} \mid \mu \mathbf{t}.T \mid \mathbf{t} \mid \mathbf{end}$$

Selection among **outputs**

Recursion

Branching among **inputs**

(Simple) session types

- A **minimal** syntax for session types:

$$T ::= \oplus\{l_i : T_i\}_{i \in I} \mid \&\{l_i : T_i\}_{i \in I} \mid \mu \mathbf{t}.T \mid \mathbf{t} \mid \mathbf{end}$$

Selection among **outputs**

Recursion

Termination

Branching among **inputs**

(Simple) example

- A session type for a service:

$$S_{service} = \mu \mathbf{t}.\&\{\text{add\_to\_cart} : \mathbf{t}\ ,\ \text{pay} : \mathbf{end}\}$$

(Simple) example

- A session type for a service:

$$S_{service} = \mu \mathbf{t}.\&\{\text{add\_to\_cart} : \mathbf{t}\ ,\ \ \text{pay} : \mathbf{end}\}$$

- A session type for a client:

$$T_{client} = \oplus\{\text{add\_to\_cart} : \oplus\{\text{pay} : \mathbf{end}\}\}$$

(Simple) example

- A session type for a service:

$$S_{service} = \mu \mathbf{t}.\&\{\text{add\_to\_cart} : \mathbf{t}\ ,\ \ \text{pay} : \mathbf{end}\}$$

- A session type for a client:

$$T_{client} = \oplus\{\text{add\_to\_cart} : \oplus\{\text{pay} : \mathbf{end}\}\}$$

Are the client protocol and the service protocol compatible ?

Compatibility between dual types

- A protocol is always compatible with its dual (i.e. the **specular** protocol)

$$\overline{\oplus\{l_i : T_i\}_{i\in I}} = \&\{l_i : \overline{T}_i\}_{i\in I}$$
$$\overline{\&\{l_i : T_i\}_{i\in I}} = \oplus\{l_i : \overline{T}_i\}_{i\in I}$$
$$\overline{\mu \mathbf{t}.T} = \mu \mathbf{t}.\overline{T},\ \overline{\mathbf{t}} = \mathbf{t},\ \overline{\mathbf{end}} = \mathbf{end}$$

$$S = \mu \mathbf{t}.\ \&\{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}$$

$$\overline{S} = \mu \mathbf{t}.\ \oplus\{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}$$

Liskov substitution principle [ToPLaS94]

A Behavioral Notion of Subtyping

BARBARA H. LISKOV
MIT Laboratory for Computer Science
and
JEANNETTE M. WING
Carnegie Mellon University

Subtype Requirement: Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T.

- All protocols with a session type which is **subtype** of the dual are compatible

Session subtyping [ActaInf05]

Simon Gay · Malcolm Hole

Subtyping for session types in the pi calculus

Definition (Synchronous Subtyping, $\leq$). *$\mathcal{R}$ is a synchronous subtyping relation whenever $(T, S) \in \mathcal{R}$ implies that:*

1. *if $T = \mathbf{end}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \mathbf{end}$;*
2. *if $T = \oplus\{l_i : T_i\}_{i \in I}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \oplus\{l_j : S_j\}_{j \in J}$, $I \subseteq J$ and $\forall i \in I. (T_i, S_i) \in \mathcal{R}$;*
3. *if $T = \&\{l_i : T_i\}_{i \in I}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \&\{l_j : S_j\}_{j \in J}$, $J \subseteq I$ and $\forall j \in J. (T_j, S_j) \in \mathcal{R}$;*
4. *if $T = \mu \mathbf{t}.T'$ then $(T'\{T/\mathbf{t}\}, S) \in \mathcal{R}$.*

T is a synchronous subtype of S, written $T \leq S$, if there is a synchronous subtyping relation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$.

Output Covariance and Input Contravariance

- Output Covariance
  - subtype may have a **subset of outputs**
  - example:

$$\oplus\{l_1 : T_1\} \leq \oplus\{l_1 : T_1, l_2 : T_2\}$$

- Input Contravariance
  - subtype may have a **superset of inputs**
  - example:

$$\&\{l_1 : T_1, l_2 : T_2\} \leq \&\{l_1 : T_1\}$$

Session subtyping [ActaInf05]

Simon Gay · Malcolm Hole

Subtyping for session types in the pi calculus

Definition (Synchronous Subtyping, $\leq$). *$\mathcal{R}$ is a synchronous subtyping relation whenever $(T, S) \in \mathcal{R}$ implies that:*

1. *if* $T = \mathbf{end}$ *then* $\exists n \geq 0$ *such that* $\mathsf{unfold}^n(S) = \mathbf{end}$;
2. *if* $T = \oplus\{l_i : T_i\}_{i \in I}$ *then* $\exists n \geq 0$ *such that* $\mathsf{unfold}^n(S) = \oplus\{l_j : S_j\}_{j \in J}$, $I \subseteq J$ *and* $\forall i \in I.\, (T_i, S_i) \in \mathcal{R}$;
3. *if* $T = \&\{l_i : T_i\}_{i \in I}$ *then* $\exists n \geq 0$ *such that* $\mathsf{unfold}^n(S) = \&\{l_j : S_j\}_{j \in J}$, $J \subseteq I$ *and* $\forall j \in J.\, (T_j, S_j) \in \mathcal{R}$;
4. *if* $T = \mu \mathbf{t}.T'$ *then* $(T'\{T/\mathbf{t}\}, S) \in \mathcal{R}$.

T is a synchronous subtype of S, written $T \leq S$, if there is a synchronous subtyping relation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$.

Our simple example

$$\overline{S} = \mu \mathbf{t}. \oplus \{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}$$
$$T = \oplus\{\text{add\_to\_cart} : \oplus\{\text{pay} : \mathbf{end}\}\}$$

? $T \leq \overline{S}$?

Our simple example

$$\overline{S} = \mu \mathbf{t}. \oplus \{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}$$
$$T = \oplus\{\text{add\_to\_cart} : \oplus\{\text{pay} : \mathbf{end}\}\}$$

? $T \leq \overline{S}$?

- A synchronous subtyping relation:

$$\{ \ (\oplus\{\text{add\_to\_cart} : \oplus\{\text{pay} : \mathbf{end}\}\} \ , \ \mu \mathbf{t}. \ \oplus \{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}),$$
$$(\oplus\{\text{pay} : \mathbf{end}\} \ , \ \mu \mathbf{t}. \ \oplus \{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}),$$
$$(\mathbf{end} \ , \ \mathbf{end}) \ \}$$

Our simple example

$$\overline{S} = \mu \mathbf{t}. \oplus \{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}$$
$$T = \oplus\{\text{add\_to\_cart} : \oplus\{\text{pay} : \mathbf{end}\}\}$$

? $T \leq \overline{S}$?

- A synchronous subtyping relation:

$$\{ \ (\oplus\{\text{add\_to\_cart} : \oplus\{\text{pay} : \mathbf{end}\}\} \ , \ \mu \mathbf{t}. \ \oplus \{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}),$$
$$(\oplus\{\text{pay} : \mathbf{end}\} \ , \ \mu \mathbf{t}. \ \oplus \{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}),$$
$$(\mathbf{end} \ , \ \mathbf{end}) \ \}$$

  - unfold of the supertype:

$$\oplus\{\text{add\_to\_cart} : (\mu \mathbf{t}. \ \oplus \{\text{add\_to\_cart} : \mathbf{t}, \text{pay} : \mathbf{end}\}), \text{pay} : \mathbf{end}\}$$

The subtyping simulation game

$$\frac{T \leqslant U \in \Sigma}{\Sigma \vdash T \leqslant U}\ \text{AS-Assump} \qquad \frac{}{\Sigma \vdash \mathsf{end} \leqslant \mathsf{end}}\ \text{AS-End}$$

$$\frac{\Sigma, \mu X.T \leqslant U \vdash T\{\mu X.T/X\} \leqslant U}{\Sigma \vdash \mu X.T \leqslant U}\ \text{AS-RecL}$$

$$\frac{\Sigma, T \leqslant \mu X.U \vdash T \leqslant U\{\mu X.U/X\}}{\Sigma \vdash T \leqslant \mu X.U}\ \text{AS-RecR}$$

$$\frac{\Sigma \vdash \widetilde{T} \leqslant \widetilde{U} \quad \Sigma \vdash V \leqslant W}{\Sigma \vdash ?[\widetilde{T}].V \leqslant ?[\widetilde{U}].W}\ \text{AS-InS} \qquad \frac{\Sigma \vdash \widetilde{U} \leqslant \widetilde{T} \quad \Sigma \vdash V \leqslant W}{\Sigma \vdash ![\widetilde{T}].V \leqslant ![\widetilde{U}].W}\ \text{AS-OutS}$$

$$\frac{m \leqslant n \quad \forall i \in \{1, \ldots, m\}.(\Sigma \vdash S_i \leqslant T_i)}{\Sigma \vdash \&\langle l_i : S_i \rangle_{1 \leqslant i \leqslant m} \leqslant \&\langle l_i : T_i \rangle_{1 \leqslant i \leqslant n}}\ \text{AS-Branch}$$

$$\frac{m \leqslant n \quad \forall i \in \{1, \ldots, m\}.(\Sigma \vdash S_i \leqslant T_i)}{\Sigma \vdash \oplus\langle l_i : S_i \rangle_{1 \leqslant i \leqslant n} \leqslant \oplus\langle l_i : T_i \rangle_{1 \leqslant i \leqslant m}}\ \text{AS-Choice}$$

Fig. 11 Algorithmic subtyping rules

Structure of the course

- Session types: subtyping
- Behavioural contracts: CFSMs, BPA
- The synchronous case
  - Session subtyping ≈ Contract refinement
- Asynchronous communication
  - Asynchronous session subtyping
  - Async. subtyping ≈ Async. refinement
  - Positive and Negative implications

Behavioural contracts

- An **operational** model for concurrent communicating processes

Behavioural contracts

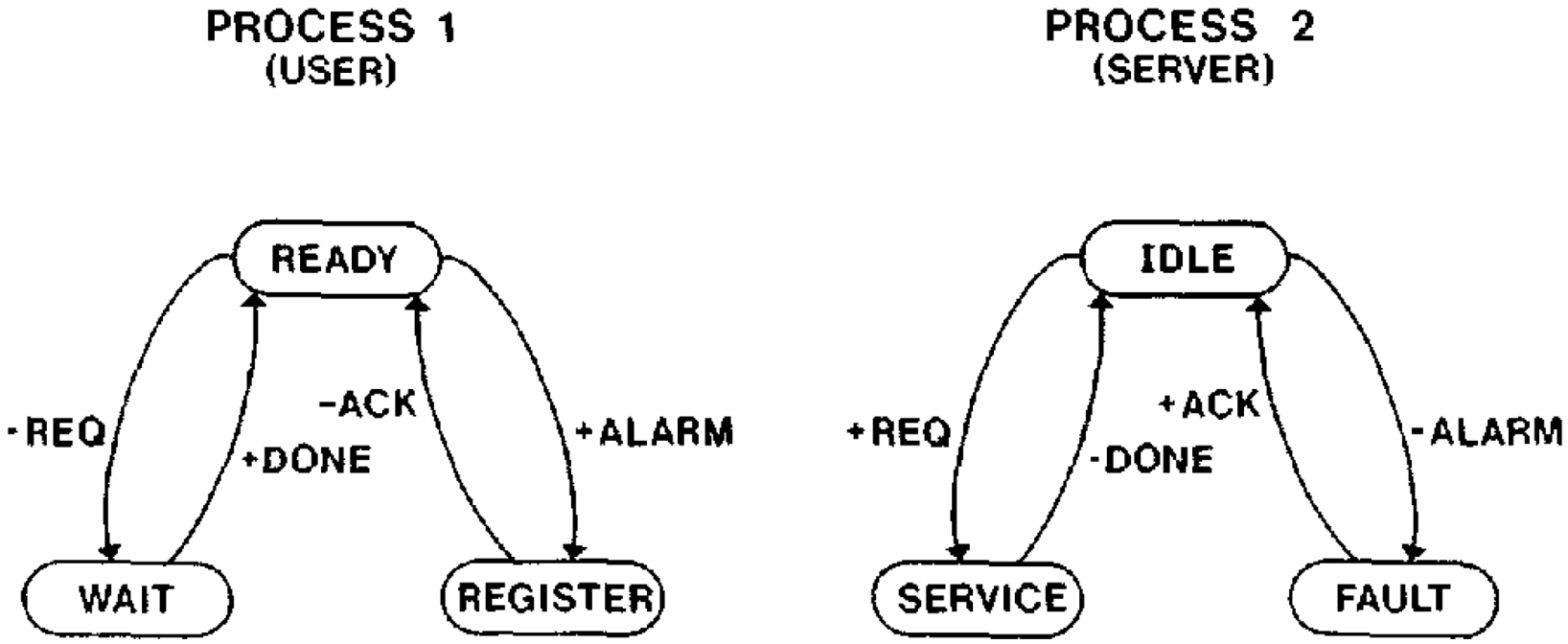
- An **operational** model for concurrent communicating processes

Communicating Finite-State Machines (CFSMs) [JACM83]

On Communicating Finite-State Machines

DANIEL BRAND AND PITRO ZAFIROPULO

IBM Zurich Research Laboratory, Ruschlikon, Switzerland



Process Algebraic Contracts [CAV04]

Stuck-Free Conformance

Cédric Fournet, Tony Hoare, Sriram K. Rajamani, and Jakob Rehof

Microsoft Research
{fournet,thoare,sriram,rehof}@microsoft.com

- **CCS** used to model behavioural contracts

$$C ::= \mathbf{1} \mid \sum_{i \in I} \alpha_i.C_i \mid X \mid recX.C$$

$$\alpha ::= a \mid \overline{a}$$

Basic Process Algebra (BPA)

$$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1}$$

$$recX.\left(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1}\right)$$

- **CCS** used to model behavioural contracts

$$C \;::=\; \mathbf{1} \;\mid\; \textstyle\sum_{i\in I} \alpha_i.C_i \;\mid\; X \;\mid\; recX.C$$

$$\alpha \;::=\; a \;\mid\; \overline{a}$$

Basic Process Algebra (BPA)

$$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1}$$

$$recX.\left(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1}\right)$$

- Operational semantics defined as a **labeled transition system** (LTS):

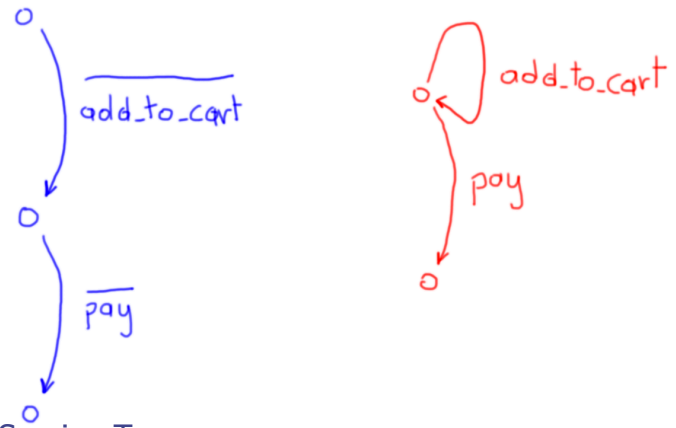
$$\frac{j \in I}{\sum_{i \in I} \alpha_i.C_i \xrightarrow{\alpha_j} C_j} \qquad \frac{C\{recX.C/X\} \xrightarrow{\lambda} C'}{recX.C \xrightarrow{\lambda} C'}$$

Basic Process Algebra (BPA)

$$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1}$$

$$recX.(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1})$$

- Operational semantics defined as a **labeled transition system** (LTS):



Coincidence between BPA and CFSMs

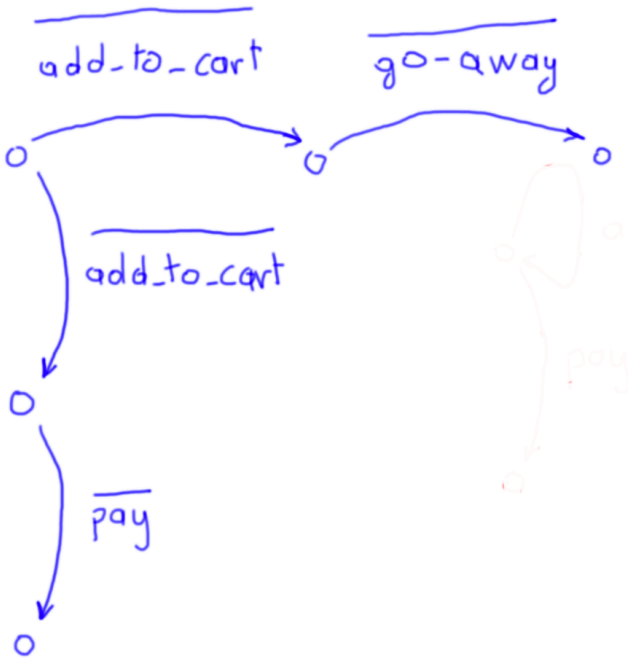
- The operational semantics of a **process algebraic contract** is a finite LTS, that can be interpreted as a **CFSM**
- Given a **CFSM**, it can be interpreted as a finite LTS, and it is possible to synthesize a **process algebraic contract** whose operational semantics is isomorphic with it

I have "cheated".. ;-)

- Actually, CFSMs are **deterministic** !

$$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.1 \;+\; \overline{\text{add\_to\_cart}}.\overline{\text{go\_away}}.1$$

Two transitions from the same state with the same label



Not admitted by CFSMs

Contract composition

- Behavioural contracts communicate by performing **complementary** actions
  - We use $C \| D$ to denote the (parallel) **composition** of two contracts
  - The **operational semantics** of contract compositions is given by a transition system:

$$\frac{C \xrightarrow{\overline{a}} C' \quad D \xrightarrow{a} D'}{C \| D \longrightarrow C' \| D'}$$

Contract composition

$$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1} \| recX.\left(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1}\right)$$

$$\overline{\text{pay}}.\mathbf{1} \| recX.\left(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1}\right)$$

Successful completion

$$\mathbf{1} \| \mathbf{1}$$

$$\frac{C \xrightarrow{\overline{a}} C' \quad D \xrightarrow{a} D'}{C \| D \longrightarrow C' \| D'}$$

Contract compliance

- Two contracts are **compliant** when their composition is correct:
  - i.e. all possible computations
    - terminates with **successful completion**, or
    - **never** terminate

Contract compliance

- Two contracts are **compliant** when their composition is correct:
  - i.e. all possible computations
    - terminates with **successful completion**, or
    - **never** terminate
- **Question time**: are

$$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1} \;+\; \overline{\text{add\_to\_cart}}.\overline{\text{go\_away}}.\mathbf{1}$$

$$recX.\left(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1}\right)$$

compliant ?

Contract compliance

- Two contracts are **compliant** when their composition is correct:
  - i.e. all possible computations
    - terminates with **successful completion**, or
    - **never** terminate
- **Question time**: are

$$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1} \;+\; \overline{\text{add\_to\_cart}}.\overline{\text{go\_away}}.\mathbf{1}$$

$$recX.\left(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1}\right)$$

compliant ?

NO !

Contract compliance

- Two contracts are **compliant** when their composition is correct:
  - i.e. all possible computations
    - terminates with **successful completion**, or
    - **never** terminate
- **Question time (2)**: are

$$\overline{\text{add\_to\_cart}}.(\overline{\text{pay}}.\mathbf{1} + \overline{\text{go\_away}}.\mathbf{1})$$
$$recX.(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1})$$

compliant ?

Contract compliance

- Two contracts are **compliant** when their composition is correct:
  - i.e. all possible computations
    - terminates with **successful completion**, or
    - **never** terminate
- **Question time (2)**: are

$$\overline{\text{add\_to\_cart}}.(\overline{\text{pay}}.\mathbf{1} + \overline{\text{go\_away}}.\mathbf{1})$$
$$recX.(\text{add\_to\_cart}.X + \text{pay}.\mathbf{1})$$

compliant ?

YES !

Structure of the course

- Session types: subtyping
- Behavioural contracts: CFSMs, BPA
- The synchronous case
  - Session subtyping ≈ Contract refinement
- Asynchronous communication
  - Asynchronous session subtyping
  - Async. subtyping ≈ Async. refinement
  - Positive and Negative implications

Contract refinement

- A contract C′ refines a contract C if it is a **compliance-preserving** substitute:

$$\forall D.\ \big(C \text{ is compliant with } D\big) \Rightarrow \big(C' \text{ is compliant with } D\big)$$

Contract refinement

- A contract C′ refines a contract C if it is a **compliance-preserving** substitute:

$$\forall D.\ \big(C \text{ is compliant with } D\big) \Rightarrow \big(C' \text{ is compliant with } D\big)$$

- **Question time**: is

$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1}$ a refinement of

$recX.\big(\overline{\text{add\_to\_cart}}.X + \overline{\text{pay}}.\mathbf{1}\big)$?

Contract refinement

- A contract C′ refines a contract C if it is a **compliance-preserving** substitute:

$$\forall D.\ \big(C \text{ is compliant with } D\big) \Rightarrow \big(C' \text{ is compliant with } D\big)$$

- **Question time**: is

$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1}$ a refinement of

$recX.\big(\overline{\text{add\_to\_cart}}.X + \overline{\text{pay}}.\mathbf{1}\big)$?

NO !

Contract refinement ≠ Session subtyping

- We have that $\overline{\mathrm{add\_to\_cart}}.\overline{\mathrm{pay}}.\mathbf{1}$ is **not a refinement** of $recX.\left(\overline{\mathrm{add\_to\_cart}}.X + \overline{\mathrm{pay}}.\mathbf{1}\right)$
  - Consider the **discriminating** contract $\mathrm{pay}.\mathbf{1}$ which is compliant with the latter but not with the former
- .. while we know that the session type $\oplus\{\mathrm{add\_to\_cart} : \oplus\{\mathrm{pay} : \mathbf{end}\}\}$ is **subtype** of $\mu\mathbf{t}.\ \oplus\{\mathrm{add\_to\_cart} : \mathbf{t}, \mathrm{pay} : \mathbf{end}\}$

Observation

- In session types
  - output internal choices $\oplus$ are treated **differently** from input external choices $\&$
  - see e.g. output **covariance** and input **contravariance**
- While in BPA input and output are treated **specularly**
  - .. we need a **different operational model** for contracts !

Session contracts [MSCS14]

Math. Struct. in Comp. Science (2016), *vol.* 26, *pp.* 510–560.
doi:10.1017/S0960129514000243 First published online 10 November 2014

Modelling session types using contracts

GIOVANNI BERNARDI and MATTHEW HENNESSY†

- **No-mixed** deterministic choices

$$C \;::=\; \mathbf{1} \;\mid\; \sum_{i\in I} a^i.C_i \;\mid\; \sum_{i\in I} \overline{a^i}.C_i \;\mid\; X \;\mid\; recX.C$$

Session contracts [MSCS14]

$$\frac{j \in I}{\sum_{i \in I} a^i.C_i \xrightarrow{a^j} C_j} \qquad \frac{C\{recX.C/X\} \xrightarrow{\lambda} C'}{recX.C \xrightarrow{\lambda} C'}$$

$$\frac{j \in I}{\sum_{i \in I} \overline{a}^i.C_i \longrightarrow \left(\overline{a}^j.C_j\right) \xrightarrow{\overline{a}^j} C_j}$$

- **No-mixed** deterministic choices, with **output** selection

$$C \;::=\; \mathbf{1} \;\mid\; \sum_{i \in I} a^i.C_i \;\mid\; \sum_{i \in I} \overline{a^i}.C_i \;\mid\; X \;\mid\; recX.C$$

What about the counter-example?

- **Question time**: now that we restricted to session contracts, reconsider the two contracts

$$\overline{\text{add\_to\_cart}}.\overline{\text{pay}}.\mathbf{1}$$

$$recX.\left(\overline{\text{add\_to\_cart}}.X + \overline{\text{pay}}.\mathbf{1}\right)$$

  - is it possible to find a **discrimating** contract?
    - compatible with the latter and not with the former

What about the counter-example?

- **Question time**: now that we restricted to session contracts, reconsider the two contracts

$$\overline{\mathrm{add\_to\_cart}}.\overline{\mathrm{pay}}.\mathbf{1}$$

$$recX.\left(\overline{\mathrm{add\_to\_cart}}.X + \overline{\mathrm{pay}}.\mathbf{1}\right)$$

  - is it possible to find a **discrimating** contract?
    - compatible with the latter and not with the former

NO !

(Session) Contract refinement ≈ Session subtyping

- Natural **mapping** from session types to session contracts

$$[\![T = \oplus\{l_i : T_i\}_{i\in I}]\!] = \sum_{i\in I} \overline{l_i}.[\![T_i]\!]; \quad [\![T = \&\{l_i : T_i\}_{i\in I}]\!] = \sum_{i\in I} l_i.[\![T_i]\!]$$
$$[\![\mu \mathbf{t}.T]\!] = rec\,\mathbf{t}.[\![T]\!]; \quad [\![\mathbf{t}]\!] = \mathbf{t}; \quad [\![\mathbf{end}]\!] = \mathbf{1}.$$

- Theorem:
  - Given two session types T and S :
    T is a **subtype** of S if and only if
    $[[S]]$ is a session contract **refinement** of $[[T]]$

Conclusion for the synchronous case

- Session contracts give an **operational interpretation** to session types
- Session **subtyping** coincides with compliance-**refinement** for session contracts
- The Gay-Hole **algorithm** can be used to check contract refinement on session contracts

Structure of the course

- Session types: subtyping
- Behavioural contracts: CFSMs, BPA
- The synchronous case
  - Session subtyping ≈ Contract refinement
- Asynchronous communication
  - Asynchronous session subtyping
  - Async. subtyping ≈ Async. refinement
  - Positive and Negative implications

Asynchronous Subtyping [ESOP09]

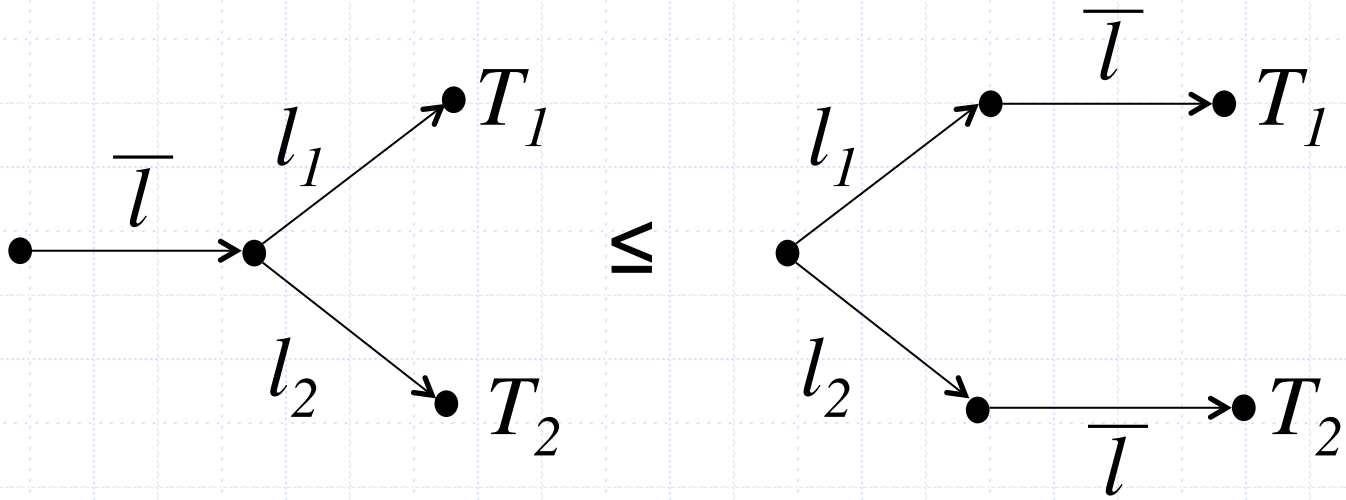
Global Principal Typing in Partially Commutative Asynchronous Sessions

Dimitris Mostrous[1], Nobuko Yoshida[1], and Kohei Honda[2]

- Besides output covariance / input contravariance a subtype may have outputs **anticipated** w.r.t. inputs

Asynchronous Subtyping [ESOP09]

$$\oplus\{l : \&\{l_1 : T_1, l_2 : T_2\}\} \leq \&\{l_1 : \oplus\{l : T_1\}, l_2 : \oplus\{l : T_2\}\}$$



- Besides output covariance / input contravariance a subtype may have outputs **anticipated** w.r.t. inputs

Synchronous Session Subtyping

Definition (Synchronous Subtyping, $\leq$). *$\mathcal{R}$ is a synchronous subtyping relation whenever $(T, S) \in \mathcal{R}$ implies that:*

1. *if $T = \mathbf{end}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \mathbf{end}$;*
2. *if $T = \oplus\{l_i : T_i\}_{i \in I}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \oplus\{l_j : S_j\}_{j \in J}$, $I \subseteq J$ and $\forall i \in I. (T_i, S_i) \in \mathcal{R}$;*
3. *if $T = \&\{l_i : T_i\}_{i \in I}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \&\{l_j : S_j\}_{j \in J}$, $J \subseteq I$ and $\forall j \in J. (T_j, S_j) \in \mathcal{R}$;*
4. *if $T = \mu \mathbf{t}.T'$ then $(T'\{T/\mathbf{t}\}, S) \in \mathcal{R}$.*

T is a synchronous subtype of S, written $T \leq S$, if there is a synchronous subtyping relation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$.

Input Context [I&C15]

Session typing and asynchronous subtyping for the higher-order π-calculus

Dimitris Mostrous [a,*], Nobuko Yoshida [b]

Definition (Input Context). *An input context $\mathcal{A}$ is a session type with multiple holes defined by the syntax:*

$$\mathcal{A} ::= \quad []^n \quad | \quad \&\{l_i : \mathcal{A}_i\}_{i \in I}$$

- Using input contexts:

$\mathcal{A}[T_k]^{k \in \{1,\ldots,m\}}$ *denotes the type obtained by filling each hole k in $\mathcal{A}$ with T_k*

Asynchronous Session Subtyping

Definition (Asynchronous Subtyping, $\leq$). *$\mathcal{R}$ is an asynchronous subtyping relation whenever $(T, S) \in \mathcal{R}$ implies that:*

1. *if $T = \mathbf{end}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \mathbf{end}$;*
2. *if $T = \oplus\{l_i : T_i\}_{i \in I}$ then $\exists n \geq 0, \mathcal{A}$ such that*
   - $\mathsf{unfold}^n(S) = \mathcal{A}[\oplus\{l_j : S_{kj}\}_{j \in J_k}]^{k \in \{1,\ldots,m\}}$,
   - $\forall k \in \{1, \ldots, m\}. I \subseteq J_k$ *and*
   - $\forall i \in I. (T_i, \mathcal{A}[S_{ki}]^{k \in \{1,\ldots,m\}}) \in \mathcal{R}$;
3. *if $T = \&\{l_i : T_i\}_{i \in I}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \&\{l_j : S_j\}_{j \in J}$, $J \subseteq I$ and $\forall j \in J. (T_j, S_j) \in \mathcal{R}$;*
4. *if $T = \mu \mathbf{t}.T'$ then $(T'\{T/\mathbf{t}\}, S) \in \mathcal{R}$.*

T is an asynchronous subtype of S, written $T \leq S$, if there is an asynchronous subtyping relation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$.

Our previous example

$$\oplus\{l : \&\{l_1 : T_1, l_2 : T_2\}\} \leq \&\{l_1 : \oplus\{l : T_1\}, l_2 : \oplus\{l : T_2\}\}$$

- How to **prove** that these two types are actually in asynchronous subtyping relation?

Our previous example

$$\oplus\{l : \&\{l_1 : T_1, l_2 : T_2\}\} \leq \&\{l_1 : \oplus\{l : T_1\}, l_2 : \oplus\{l : T_2\}\}$$

- How to **prove** that these two types are actually in asynchronous subtyping relation?

$$\{ \quad (\oplus\{l : \&\{l_1 : T_1, l_2 : T_2\}\}, \&\{l_1 : \oplus\{l : T_1\}, l_2 : \oplus\{l : T_2\}\}) \ ,$$
$$(\&\{l_1 : T_1, l_2 : T_2\}\}, \&\{l_1 : T_1, l_2 : T_2\}\}) \ ,$$
$$\ldots \quad \}$$

  - the **input context** to be used is $\&\{l_1 : [\,]^1, l_2 : [\,]^2\}$ with the holes filled with $\oplus\{l : T_1\}$ and $\oplus\{l : T_2\}$, respectively

Question time

- Are

$$\oplus\{op : \&\{resp : \oplus\{huge\_data : \mathbf{end}\}\}\}$$

$$\oplus\{op : \oplus\{huge\_data : \&\{resp : \mathbf{end}\}\}\}$$

in **subtyping** relation?
Which is the **subtype**?

Question time

- Are

  $\oplus\{op : \&\{resp : \oplus\{huge\_data : \mathbf{end}\}\}\}$

  $\oplus\{op : \oplus\{huge\_data : \&\{resp : \mathbf{end}\}\}\}$

  in **subtyping** relation?

  Which is the **subtype**?

YES !

Question time

- Are

$$\oplus\{op : \&\{resp : \oplus\{huge\_data : \mathbf{end}\}\}\}$$

$$\oplus\{op : \oplus\{huge\_data : \&\{resp : \mathbf{end}\}\}\}$$

in **subtyping** relation?

Which is the **subtype**?

YES !

The latter

Question time

- Are

$$\oplus\{op : \&\{resp : \oplus\{huge\_data : \mathbf{end}\}\}\}$$

$$\oplus\{op : \oplus\{huge\_data : \&\{resp : \mathbf{end}\}\}\}$$

Can we **automatize** this check?

A Sound Algorithm for Asynchronous Session Subtyping [CONCUR19]

Mario Bravetti
University of Bologna / INRIA FoCUS Team

Marco Carbone
IT University of Copenhagen

Julien Lange
University of Kent

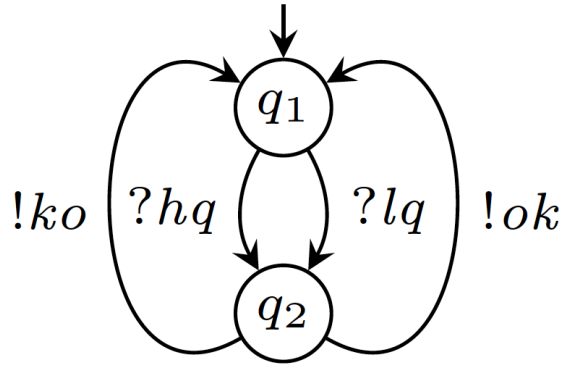
Nobuko Yoshida
Imperial College London

Gianluigi Zavattaro
University of Bologna / INRIA FoCUS Team

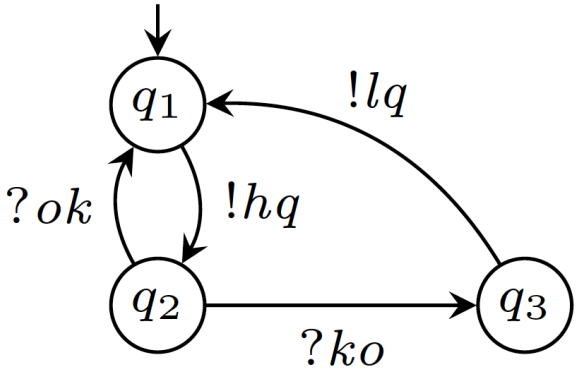
Demo time!

Question time

- Consider a **server** protocol:



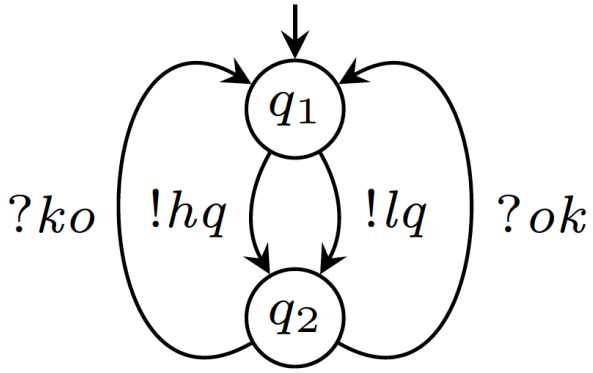
and a **client** protocol:
(! : outputs, ? : inputs)

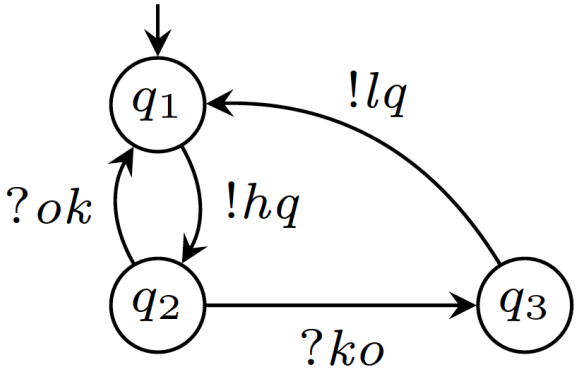


  - are they **compatible** considering asynchronous communication?

Solution

- Usual approach
  Consider the dual of
  the **server** protocol:

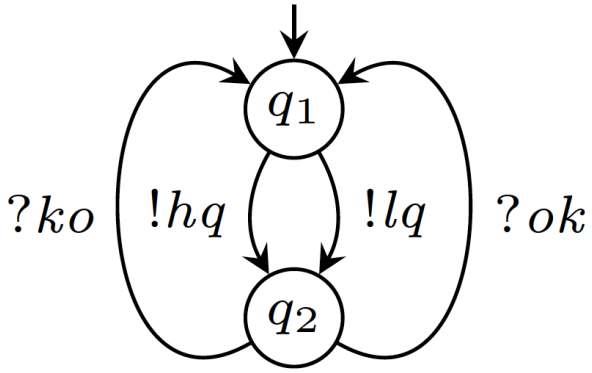


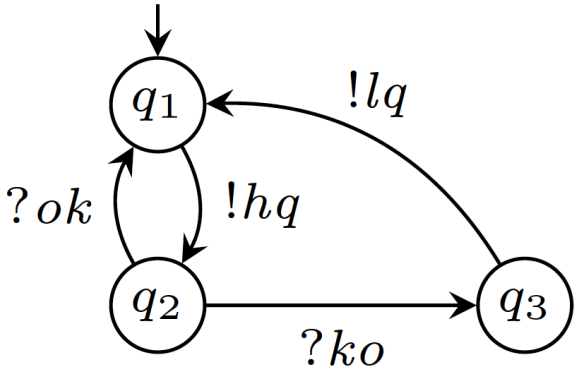


and check whether the
client protocol is
an **asynchronous** subtype
(we can try to use the CONCUR19 tool)

Solution

- Usual approach
  Consider the dual of
  the **server** protocol:



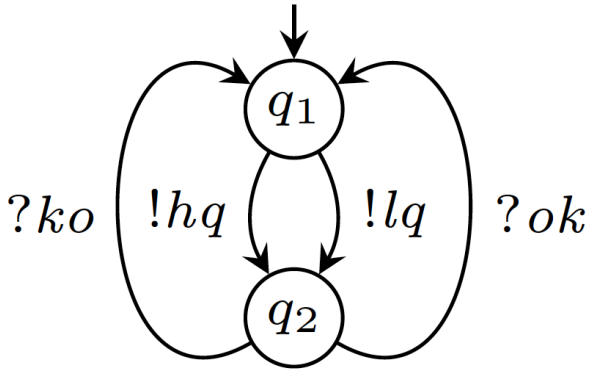


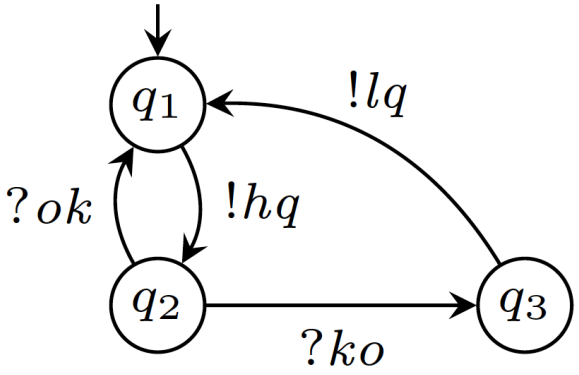
and check whether the
client protocol is
an **asynchronous** subtype
(we can try to use the CONCUR19 tool)

Demo time!

Solution

- Usual approach
  Consider the dual of the **server** protocol:





and check whether the client protocol is an **asynchronous** subtype

Demo time!

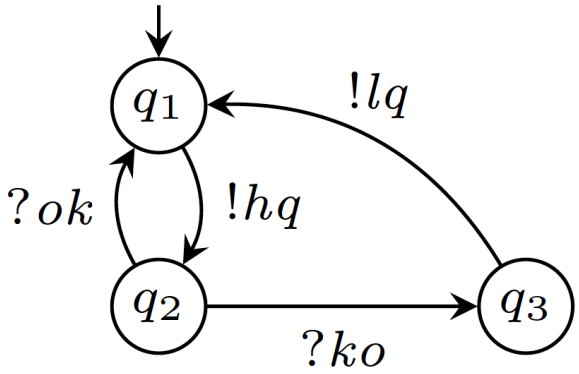
The tool allows us to reply positively !

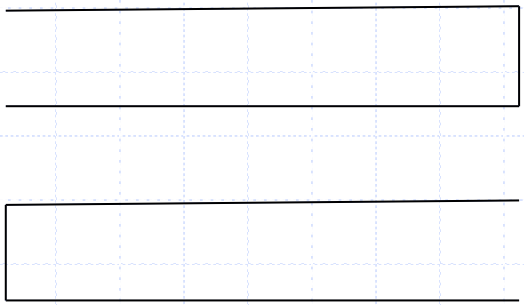
Structure of the course

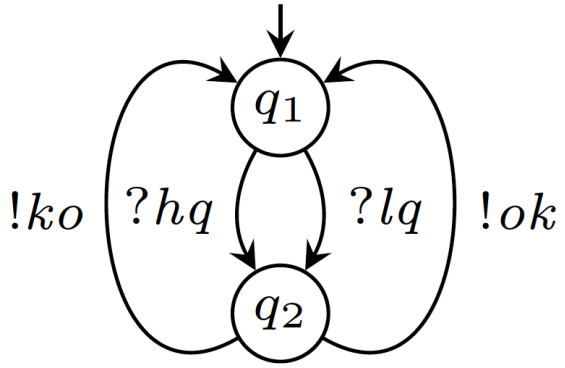
- Session types: subtyping
- Behavioural contracts: CFSMs, BPA
- The synchronous case
  - Session subtyping ≈ Contract refinement
- Asynchronous communication
  - Asynchronous session subtyping
  - Async. subtyping ≈ Async. refinement
  - Positive and Negative implications

Asynchronous contracts

- CFSMs communicate via **FIFO queues**

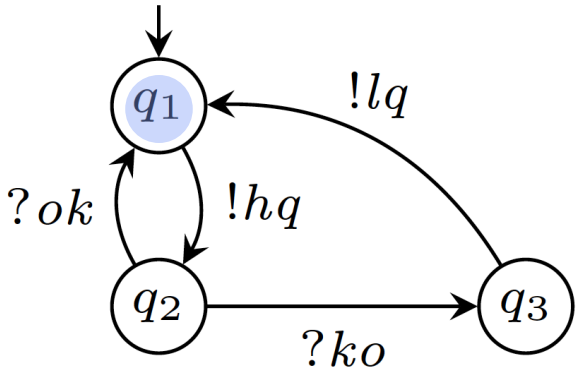


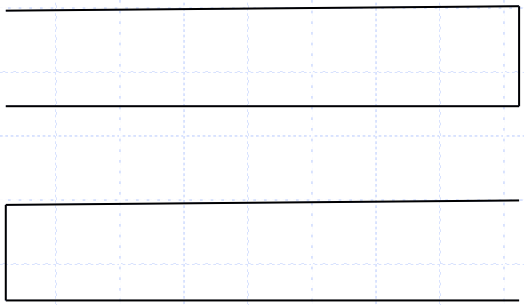


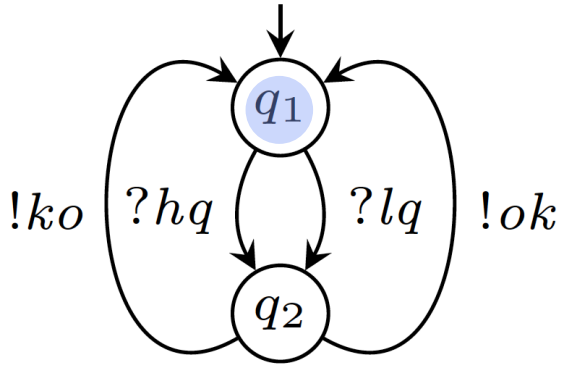


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

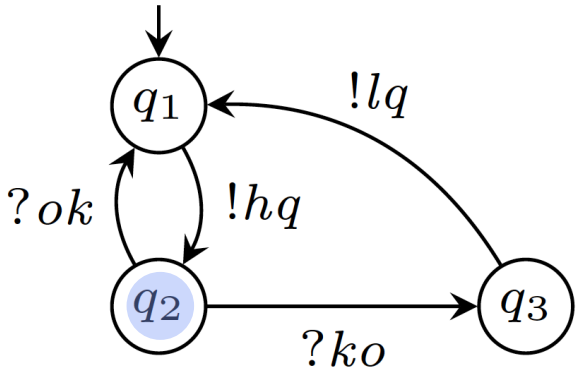


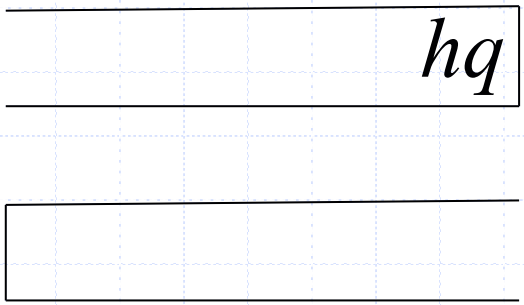


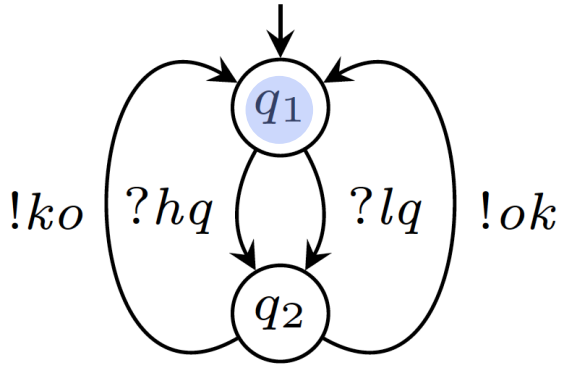


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

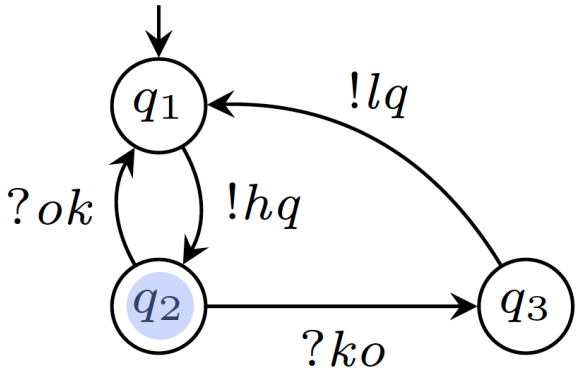


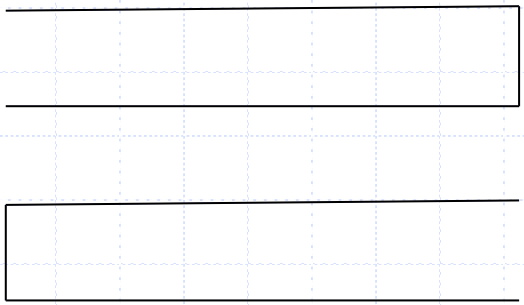


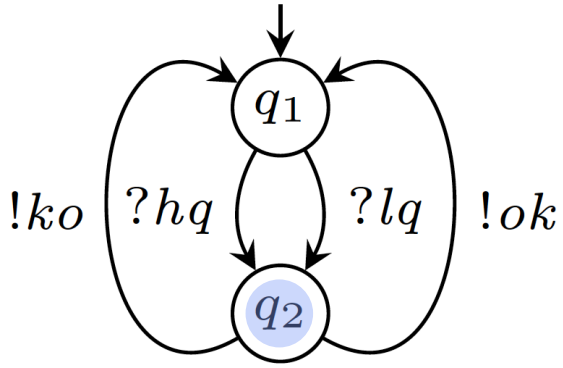


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

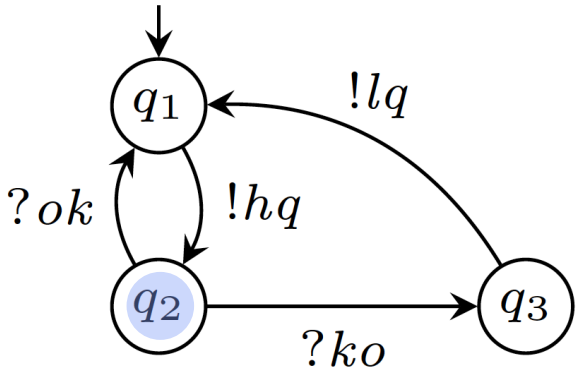


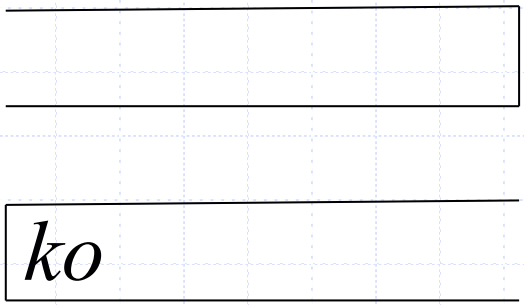


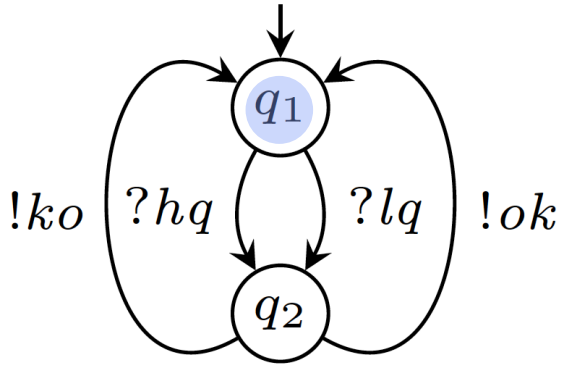


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

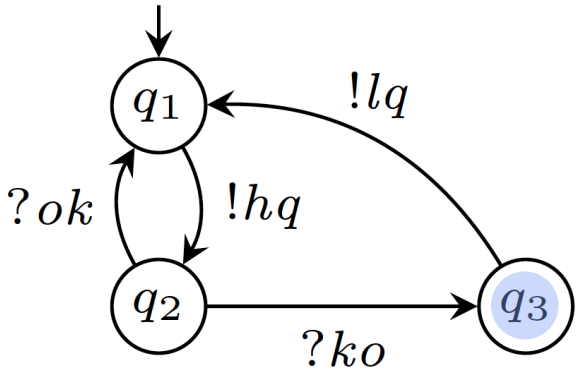


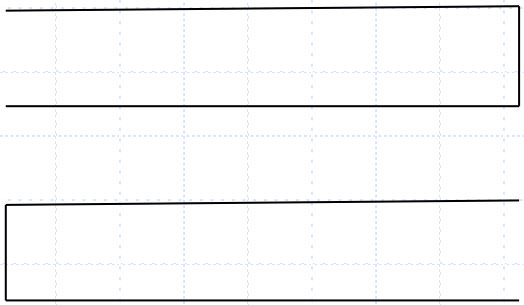


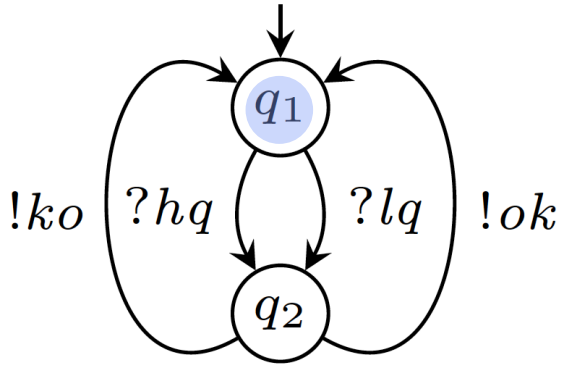


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

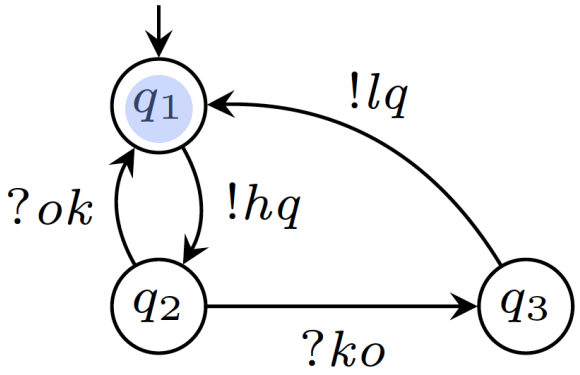


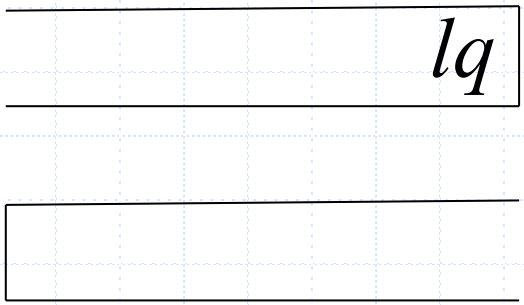


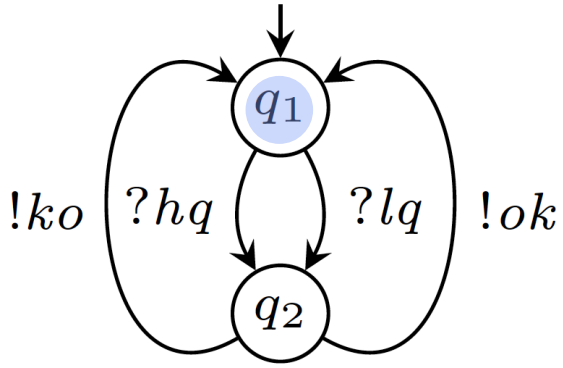


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

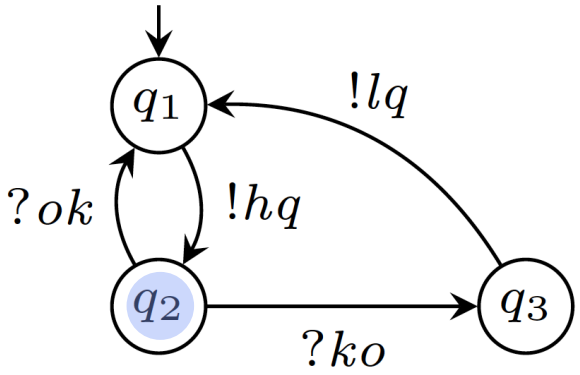


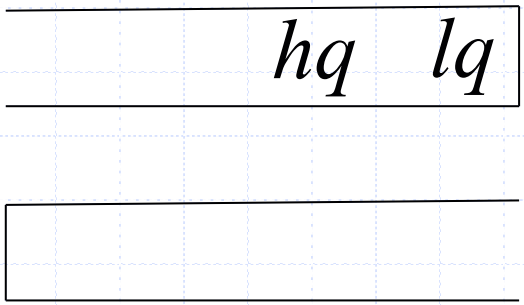


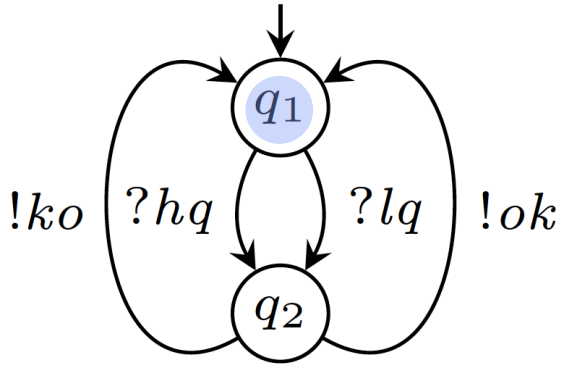


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

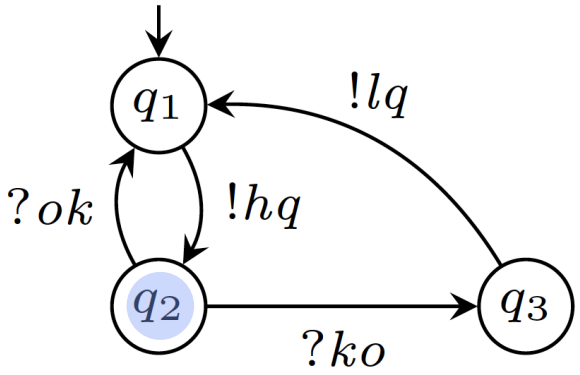


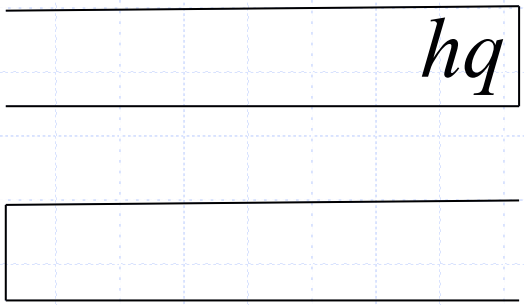


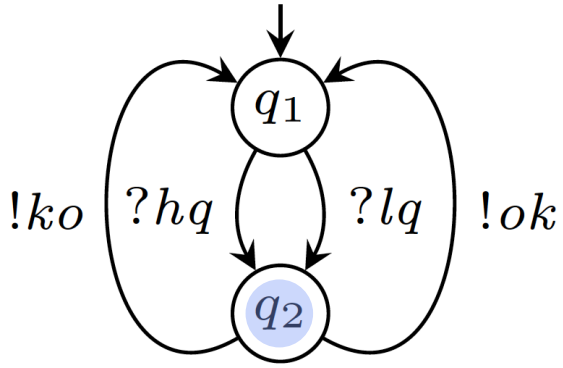


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

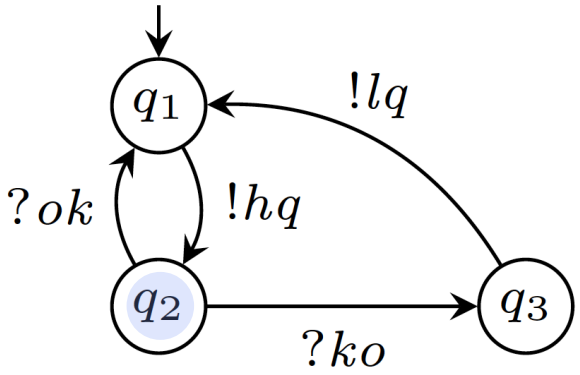


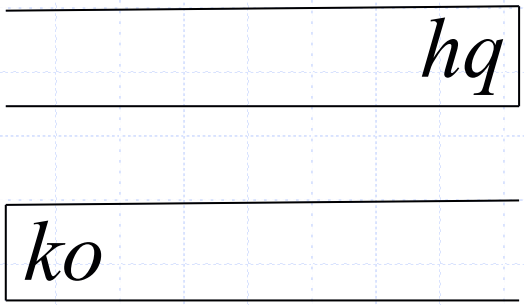


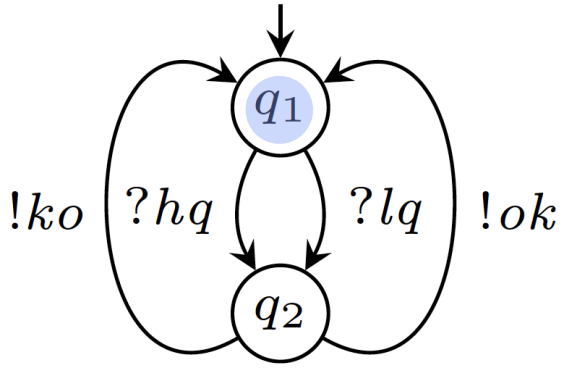


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

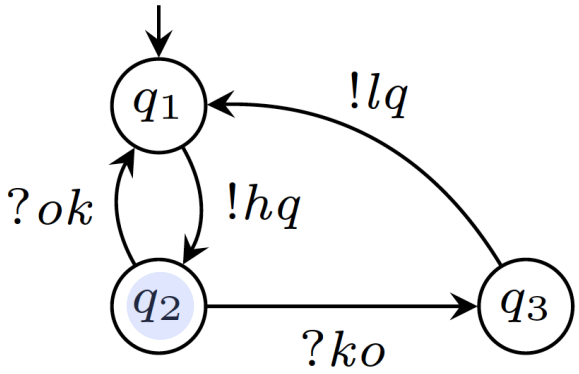


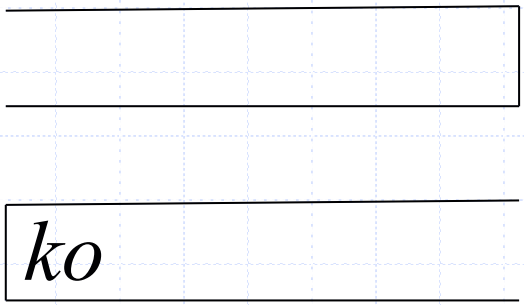


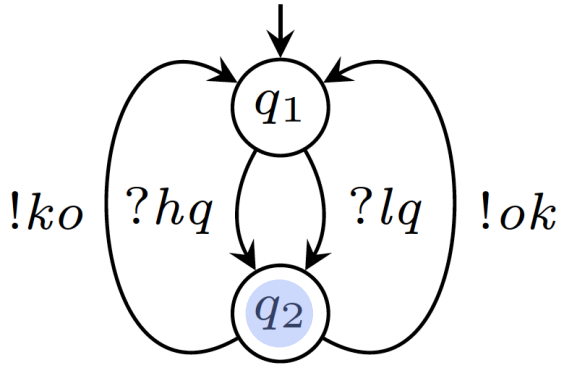


Asynchronous contracts

- CFSMs communicate via **FIFO queues**

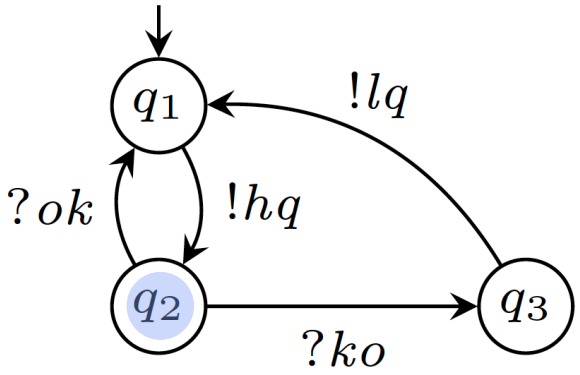


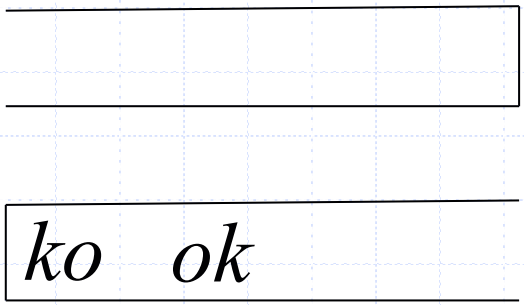


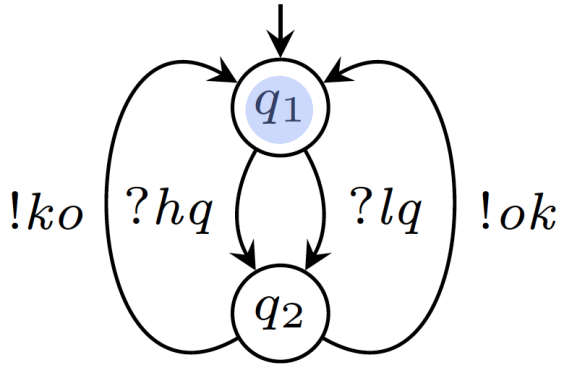


Asynchronous contracts

- CFSMs communicate via **FIFO queues**







FIFO Process Algebra

- PA **enriched** with FIFO queues:

$$\begin{array}{rcl} P & ::= & [C, \mathcal{Q}] \| [D, \mathcal{Q}] \\ \mathcal{Q} & ::= & \epsilon \quad | \quad a :: \mathcal{Q} \end{array}$$

  - Rules for **consuming** messages

$$\frac{C \xrightarrow{a} C'}{[C, a :: \mathcal{Q}] \| [D, \mathcal{Q}'] \longrightarrow [C', \mathcal{Q}] \| [D, \mathcal{Q}']}$$

and for **sending** messages

$$\frac{C \xrightarrow{\overline{a}} C'}{[C, \mathcal{Q}] \| [D, \mathcal{Q}'] \longrightarrow [C', \mathcal{Q}] \| [D, \mathcal{Q}' :: a]}$$

Remember: Contract compliance

- Two contracts are **compliant** when their composition is correct:
  - i.e. all possible computations
    - terminates with **successful completion**, or
    - **never** terminate

Remember: Contract compliance

- Two contracts are **compliant** when their composition is correct:
  - i.e. all possible computations
    - terminates with **successful completion**, or
    - **never** terminate
- Re-define **composition** and **successful completion** for the asynchronous case:
  - Contract composition: $[C, \epsilon] \| [D, \epsilon]$
  - Successful completion: $[\mathbf{1}, \epsilon] \| [\mathbf{1}, \epsilon]$

Question time

- Are

$$recX.(\mathrm{but1}.\overline{\mathrm{coffe}}.X + \mathrm{but2}.\overline{\mathrm{tea}}.X)$$

$$recX.\overline{\mathrm{but1}}.\overline{\mathrm{but2}}.(\mathrm{coffe}.X + \mathrm{tea}.X)$$

compliant ?

Question time

- Are

$$recX.(\mathrm{but1}.\overline{\mathrm{coffe}}.X + \mathrm{but2}.\overline{\mathrm{tea}}.X)$$

$$recX.\overline{\mathrm{but1}}.\overline{\mathrm{but2}}.(\mathrm{coffe}.X + \mathrm{tea}.X)$$

compliant ?

YES !

Remember: Contract refinement

- A contract C′ refines a contract C if it is a **compliance-preserving** substitute:

$$\forall D.\ (C \text{ is compliant with } D) \Rightarrow (C' \text{ is compliant with } D)$$

- This definition applies also to the **asynchronous** case

Question time

- Is $recX.\overline{\text{but1}}.\overline{\text{but2}}.(\text{coffe}.X + \text{tea}.X)$ a **refinement** of $recX.(\overline{\text{but1}}.\text{coffe}.X + \overline{\text{but2}}.\text{coffee}.X)$?
  - is it impossible to find a **discrimating** contract?
    - compatible with the latter and not with the former

Question time

- Is $recX.\overline{\text{but1}}.\overline{\text{but2}}.(\text{coffe}.X + \text{tea}.X)$ a **refinement** of $recX.(\overline{\text{but1}}.\text{coffe}.X + \overline{\text{but2}}.\text{coffee}.X)$?
  - is it impossible to find a **discrimating** contract?
    - compatible with the latter and not with the former

YES !

Demo time!

Pay attention !

- We have used a (safe) algorithm for **session subtyping**, not an algorithm to check **contract refinement**!
  - But.. do they **coincide** also in the asynchronous case?

Async. contract refinement ≈ Async. session subtyping [SEFM19]

Relating Session Types and Behavioural Contracts: the Asynchronous Case

Mario Bravetti Gianluigi Zavattaro

Department of Computer Science and Engineering & Focus Team, INRIA
University of Bologna, Italy

- Additional **restriction** (besides session contracts):
  - Session types/contracts **cannot have infinite sequences of input**

Async. contract refinement ≈ Async. session subtyping [SEFM19]

- The type $\mu \mathbf{t}. \oplus \{l : \&\{l_1 : \mathbf{t}, l_2 : \mathbf{t}\}\}$ is **not** an asynchrononous subtype of $\mu \mathbf{t}.\&\{l_1 : \mathbf{t}, l_2 : \oplus\{l : \mathbf{t}\}\}$ because there exists **no input context** for the latter (infinite sequence of l_1 input choices)
- Additional **restriction** (besides session contracts):
  - Session types/contracts **cannot have infinite sequences of input**

Async. contract refinement ≈ Async. session subtyping [SEFM19]

- If we restrict to session types and session contracts **without infinite input** loops, we have the following ..
- Theorem:
  - Given two session types T and S: T is an **asynchronous subtype** of S if and only if $[[S]]$ is an **asynchronous** contract **refinement** of $[[T]]$

$$[\![T = \oplus\{l_i : T_i\}_{i\in I}]\!] = \sum_{i\in I} \overline{l_i}.[\![T_i]\!]; \quad [\![T = \&\{l_i : T_i\}_{i\in I}]\!] = \sum_{i\in I} l_i.[\![T_i]\!]$$
$$[\![\mu \mathbf{t}.T]\!] = rec\,\mathbf{t}.[\![T]\!]; \quad [\![\mathbf{t}]\!] = \mathbf{t}; \quad [\![\mathbf{end}]\!] = \mathbf{1}.$$

Structure of the course

- Session types: subtyping
- Behavioural contracts: CFSMs, BPA
- The synchronous case
  - Session subtyping ≈ Contract refinement
- Asynchronous communication
  - Asynchronous session subtyping
  - Async. subtyping ≈ Async. refinement
  - Positive and Negative implications

Positive implications

- Session contracts give an **operational interpretation** to session types also in the asynchronous case ..
  - .. under some additional restriction
  - Asynchronous session **subtyping** coincides with compliance-**refinement** for asynchronous session contracts
  - Async. session subtyping definition **can be used** to prove contract refinement

Negative implications [I&C17]

- Asynchronous session subtyping (hence also contract refinement) is **undecidable!**

Undecidability of asynchronous session subtyping

Mario Bravetti [a], Marco Carbone [b], Gianluigi Zavattaro [a,*]

[a] *University of Bologna, Department of Computer Science and Engineering/FOCUS INRIA, Mura Anteo Zamboni 7, 40126 Bologna, Italy*
[b] *Department of Computer Science, IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen, Denmark*

- No general **algorithm** to check asynchronous session subtyping/contract refinement

Safe algorithmic characterization [CONCUR19]

A Sound Algorithm for Asynchronous Session Subtyping

Mario Bravetti
University of Bologna / INRIA FoCUS Team

Marco Carbone
IT University of Copenhagen

Julien Lange
University of Kent

Nobuko Yoshida
Imperial College London

Gianluigi Zavattaro
University of Bologna / INRIA FoCUS Team

- Subtype check that replies with **True**, **False**, or **Maybe**

Demo time!

Algorithms for restricted cases [TCS18]

On the boundary between decidability and undecidability of asynchronous session subtyping

Mario Bravetti [a,*], Marco Carbone [b], Gianluigi Zavattaro [a]

[a] *University of Bologna, Department of Computer Science and Engineering/INRIA FOCUS, Italy*
[b] *Department of Computer Science, IT University of Copenhagen, Denmark*

- Algorithms for restricted cases:
  - **Bounded** buffers
  - **Single-out**: the output choices have always only one label
    - **No internal choices**, as in a reactive service

Related work

- **Orphan-message** free subtyping [LMCS17]
  - Messages are **eventually consumed**

ON THE PRECISENESS OF SUBTYPING IN SESSION TYPES*

TZU-CHUN CHEN [a], MARIANGIOLA DEZANI-CIANCAGLINI [a], ALCESTE SCALAS [c], AND NOBUKO YOSHIDA [d]

[a] Dept. of Computer Science, TU Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany
e-mail address: tzu-chun.chen@dsp.tu-darmstadt.de

[b] Dip. di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy
e-mail address: dezani@di.unito.it

[c,d] Dept. of Computing, Imperial College London, 180 Queen's Gate, London SW7 2AZ, UK
e-mail address: {alceste.scalas, n.yoshida}@imperial.ac.uk

  - Also this subtyping is **undecidable** [FOSSACS17]

On the Undecidability of Asynchronous Session Subtyping

Julien Lange(✉) and Nobuko Yoshida

Future work

- Design algorithms that **better** approximate asynchronous subtyping
- **Apply** asynchronous subtyping into session type tools/languages
- Consider the **multiparty** case:
  - **Synchronous** case recently studied

Precise subtyping for synchronous multiparty sessions [JLAMP19]

Silvia Ghilezan [a], Svetlana Jakšić [a,c], Jovanka Pantović [a], Alceste Scalas [b], Nobuko Yoshida [b]

[a] *Univerzitet u Novom Sadu, Serbia*
[b] *Imperial College London, UK*
[c] *Høgskulen på Vestlandet, Norway*

Thank you!

- .. for your patience ;-)