Contract-Based Discovery and Composition of Web Services

Gianluigi Zavattaro
zavattar@cs.unibo.it
http://cs.unibo.it/~zavattar

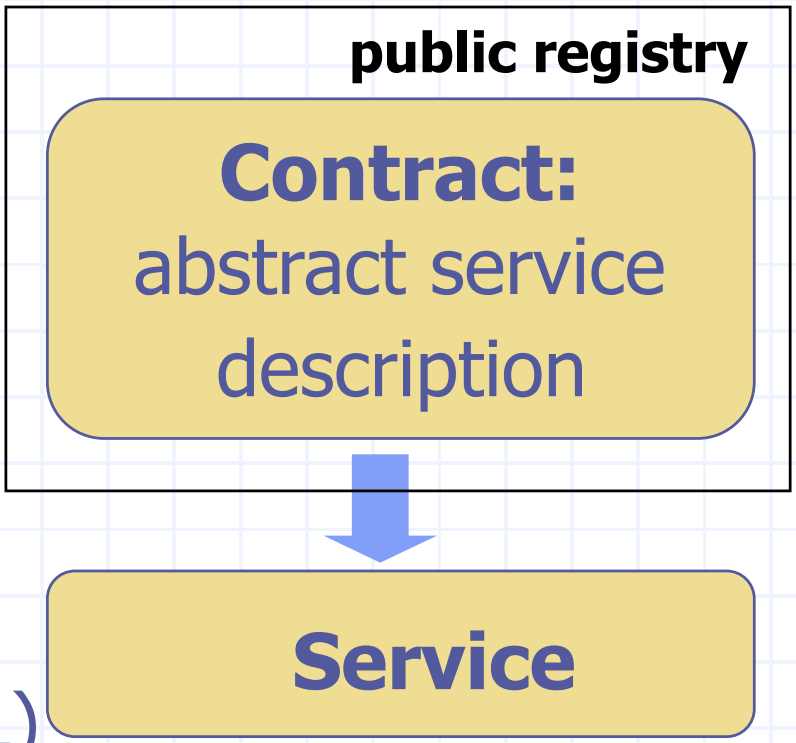
Department of Computer Science
University of Bologna

joint work with **Mario Bravetti**

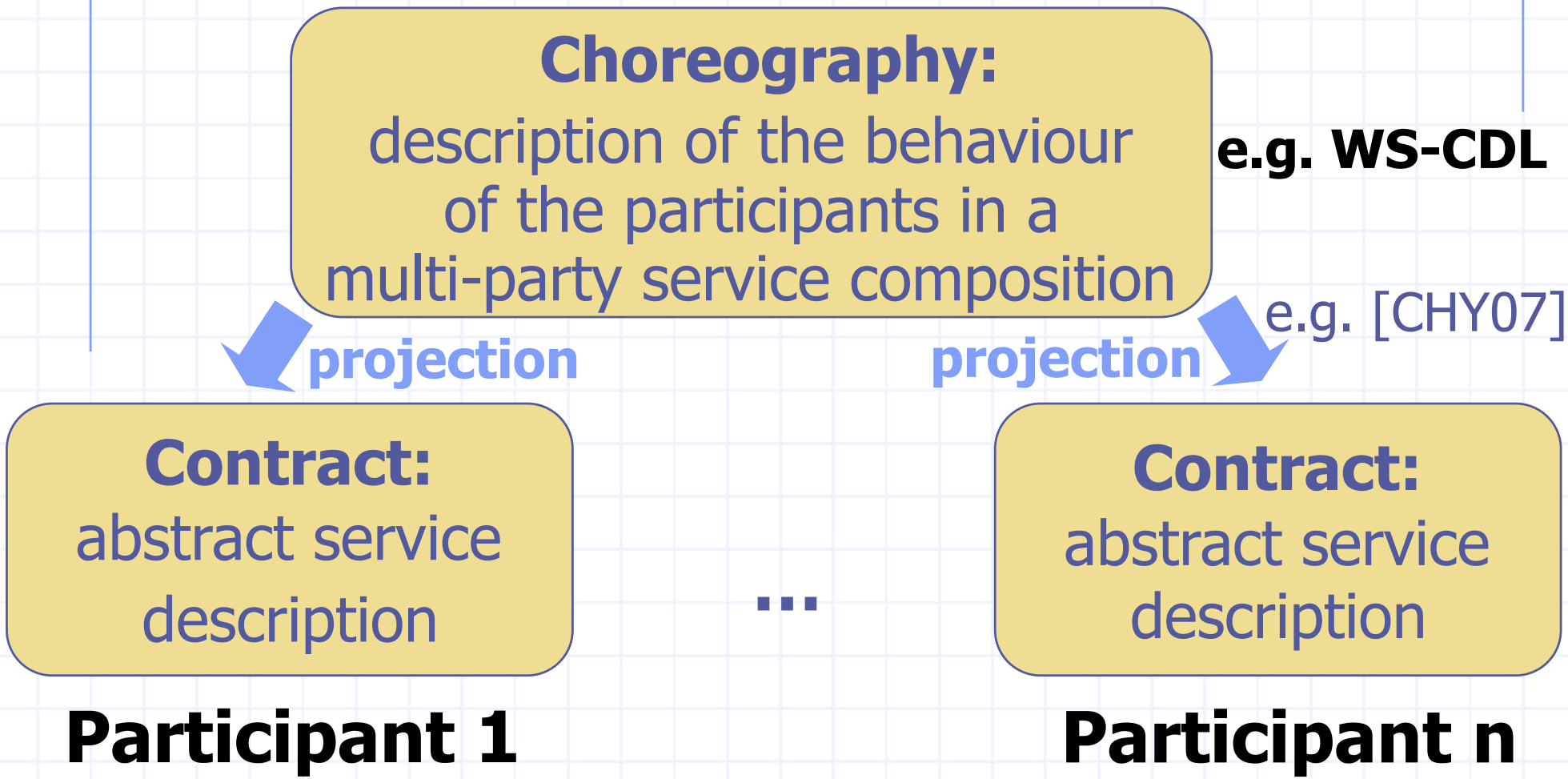
Contracts

[FHRR04][CCLP06]

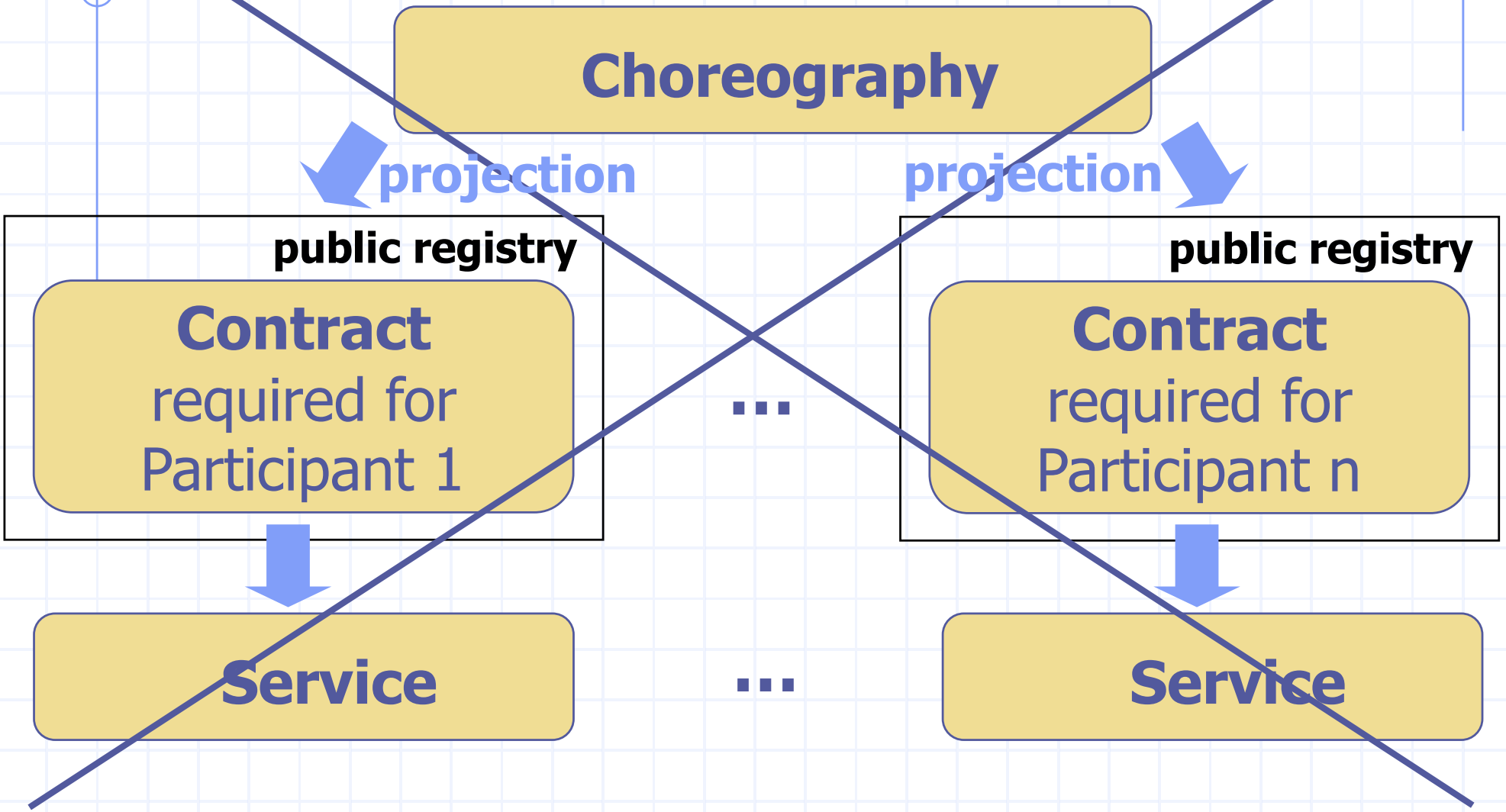
- Contract: service "behavioural interface" that describes
  - not only the signature of the provided operations (as in WSDL)
  - but also the expected order of invoke and receive (as in abstract WS-BPEL)



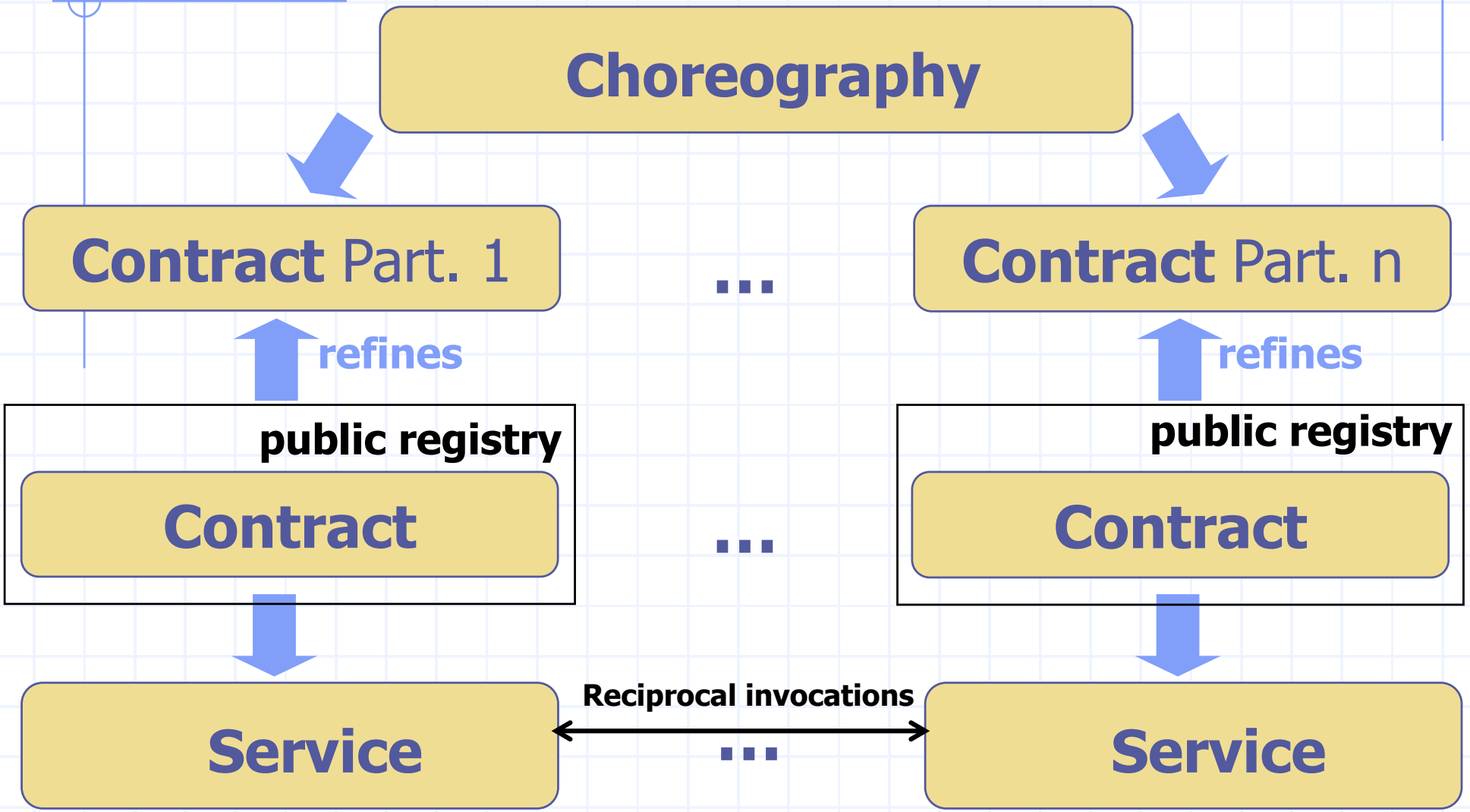
Choreographies and contracts



Discovery of required contracts: directly in registries… ?

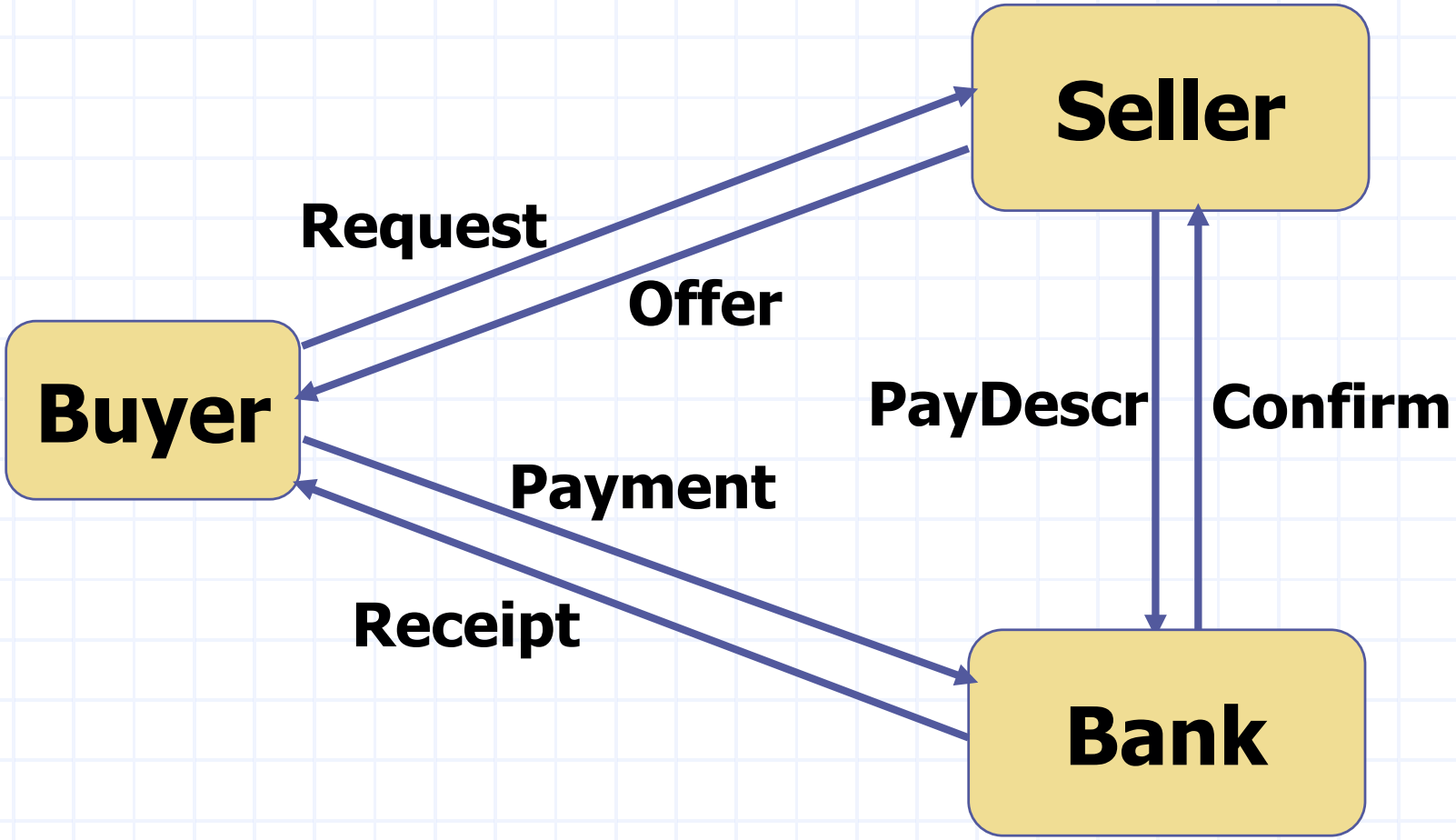


Contract refinement



An example...

- Buyer-Seller-Bank choreography



An example... (more formally)

$$\mathrm{Request}_{\mathrm{Buyer}\rightarrow\mathrm{Seller}} \; ;$$
$$(\; \mathrm{Offer}_{\mathrm{Seller}\rightarrow\mathrm{Buyer}} \; |$$
$$\mathrm{PayDescr}_{\mathrm{Seller}\rightarrow\mathrm{Bank}} \;) \; ;$$
$$\mathrm{Payment}_{\mathrm{Buyer}\rightarrow\mathrm{Bank}} \; ;$$
$$(\; \mathrm{Confirm}_{\mathrm{Bank}\rightarrow\mathrm{Seller}} \; |$$
$$\mathrm{Receipt}_{\mathrm{Bank}\rightarrow\mathrm{Buyer}} \;)$$

Projection of the Choreography on the Single Participants

Buyer: Invoke(Request)@Seller;Receive(Offer);
Invoke(Payment)@Bank;Receive(Receipt)

Seller: Receive(Request);
(Invoke(Offer)@Buyer |
Invoke(PayDescr)@Bank);
Receive(Confirm)

Bank: Receive(PayDescr);Receive(Payment);
(Invoke(Receipt)@Buyer |
Invoke(Confirm)@Seller)

Contract Refinement

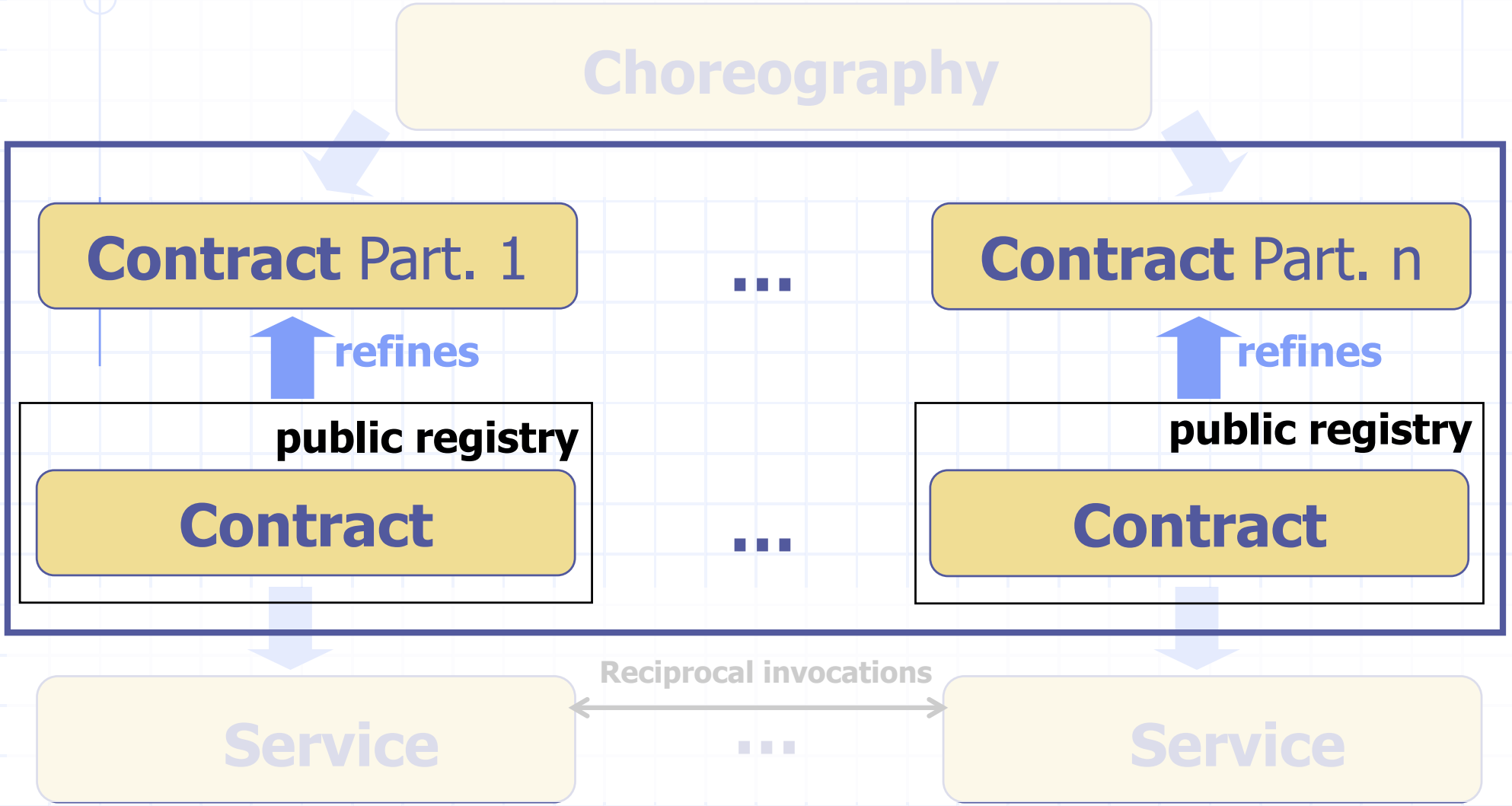
- Can we use a concrete *MonteDeiPaschi* service:

  Receive(PayDescr);Receive(Payment);
  (Invoke(Receipt)@Buyer | Invoke(Confirm)@Seller) +
  Receive(eBookStore); ….

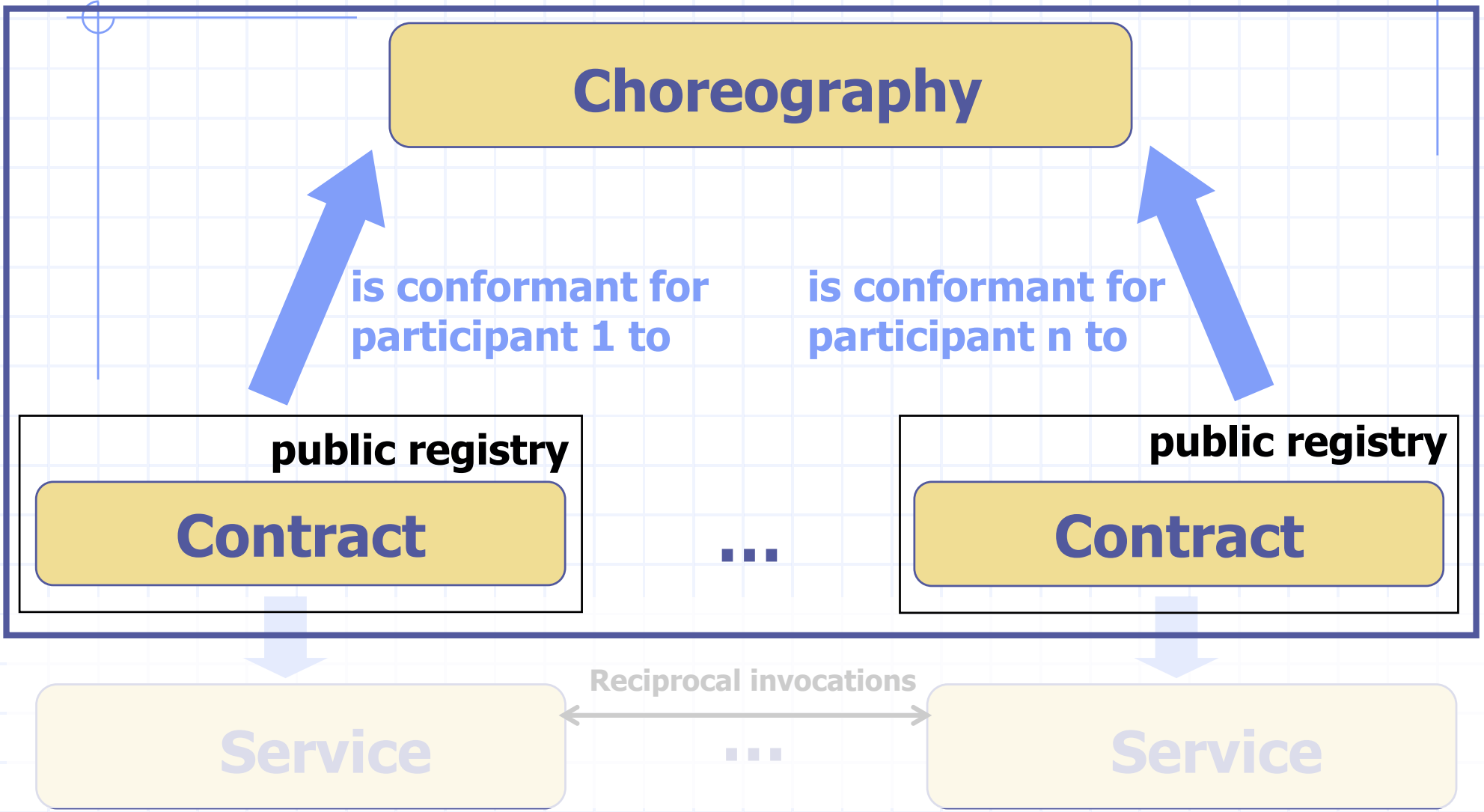
  to implement the *Bank* role?

  Receive(PayDescr);Receive(Payment);
  (Invoke(Receipt)@Buyer | Invoke(Confirm)@Seller)

Contract Refinement



Choreography Conformance



Choreography Conformance

- Can we use a concrete *MonteDeiPaschi* service:

  Receive(PayDescr);Receive(Payment);
  (Invoke(Receipt)@Buyer | Invoke(Confirm)@Seller) +
  Receive(eBookStore); ....

  to implement the *Bank* role in the choreography?

  $Request_{Buyer \rightarrow Seller}$;($Offer_{Seller \rightarrow Buyer}$ | $PayDescr_{Seller \rightarrow Bank}$);
  $Payment_{Buyer \rightarrow Bank}$;($Confirm_{Bank \rightarrow Seller}$ | $Receipt_{Bank \rightarrow Buyer}$)

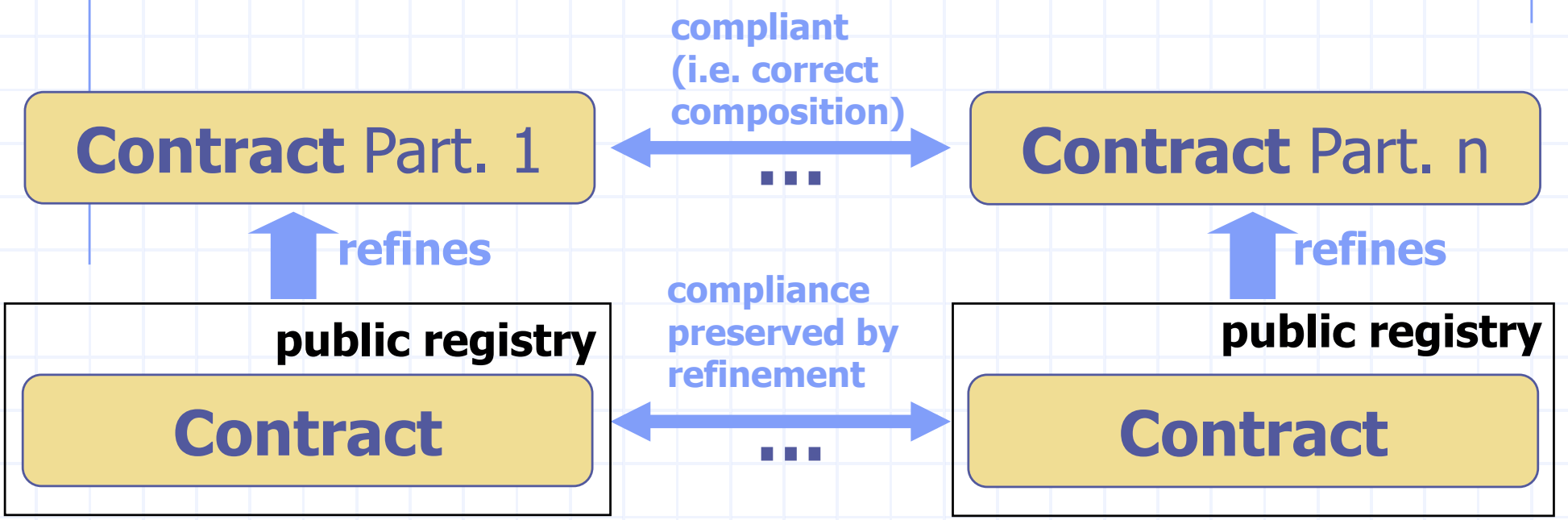
Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- Compliance-preserving refinement
  - General case (negative result)
  - Output persistence, located outputs
- Choreography conformance (negative result)
  - Consonance
- Alternative notions of compliance
  - Strong compliance, queue-based compliance
- Conclusion

Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- **Compliance-preserving refinement**
  - **General case (negative result)**
  - Output persistence, located outputs
- Choreography conformance (negative result)
  - Consonance
- Alternative notions of compliance
  - Strong compliance, queue-based compliance
- Conclusion

Compliance-Preserving Contract Refinement



A formal (language independent) model for contracts

- Services publish their interface expressed in terms of a contract "C" in UDDI-like registries

$$C ::= \mathbf{0} \mid \mathbf{1} \mid \alpha.C \mid C+C \mid X \mid recX.C$$
$$\alpha ::= \boxed{\tau} \mid \boxed{a} \mid \boxed{\overline{a}}$$

- Operational semantics:

$$\mathbf{1} \xrightarrow{\surd} \mathbf{0} \qquad \alpha.C \xrightarrow{\alpha} C$$

$$\frac{C \xrightarrow{\lambda} C'}{C+D \xrightarrow{\lambda} C'} \qquad \frac{C\{recX.C/X\} \xrightarrow{\lambda} C'}{recX.C \xrightarrow{\lambda} C'}$$

Contract composition

- A composition of contracts can be represented by the syntax:

$$P \quad ::= \quad [C] \quad | \quad P \| P \quad | \quad P \backslash\backslash L$$

- Operational semantics:

$$\frac{C \xrightarrow{\lambda} C'}{[C] \xrightarrow{\lambda} [C']} \qquad \frac{P \xrightarrow{\lambda} P' \quad \lambda \neq \surd}{P \| Q \xrightarrow{\lambda} P' \| Q}$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'} \qquad \frac{P \xrightarrow{\surd} P' \quad Q \xrightarrow{\surd} Q'}{P \| Q \xrightarrow{\surd} P' \| Q'} \qquad \frac{P \xrightarrow{\lambda} P' \quad \lambda \notin L}{P \backslash\backslash L \xrightarrow{\lambda} P' \backslash\backslash L}$$

Correct compositions: informally

- The composition $[C_1]|[C_2]$ is correct for the following compliant contracts:

$$C_1 = a + b \qquad C_2 = \tau.\overline{a} + \tau.\overline{b}$$
$$C_1 = a.b \qquad C_2 = \overline{a}.\overline{b}$$
$$C_1 = a + b + c \qquad C_2 = \tau.\overline{a} + \tau.\overline{b}$$
$$C_1 = (a.b) + (b.a) \qquad C_2 = \overline{a}.\overline{b} + \overline{b}.\overline{a}$$
$$C_1 = recX.(a.\overline{b}.(X + \mathbf{1})) \quad C_2 = recX.(\tau.\overline{a}.b.(X + \mathbf{1}))$$

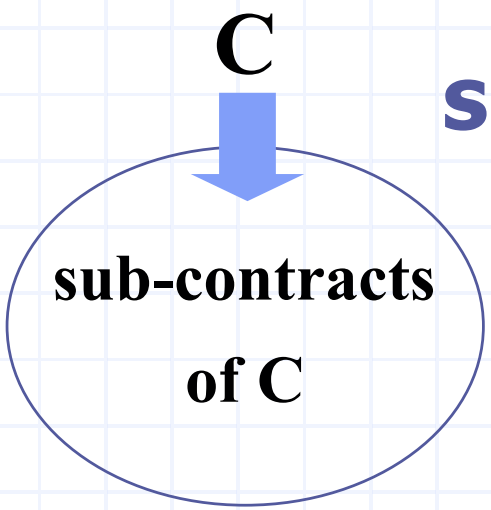
Correct compositions: formally

- A contract composition is correct if all contracts always reach successful termination (under fairness assumption)

Definition 6. (Correct contract composition) *A system P is a correct contract composition, denoted $P\downarrow$, if for every P' such that $P \xrightarrow{\tau}{}^{*} P'$ there exists P'' such that $P' \xrightarrow{\tau}{}^{*} P'' \xrightarrow{\surd}$.*

Refinement: Subcontract Preorder

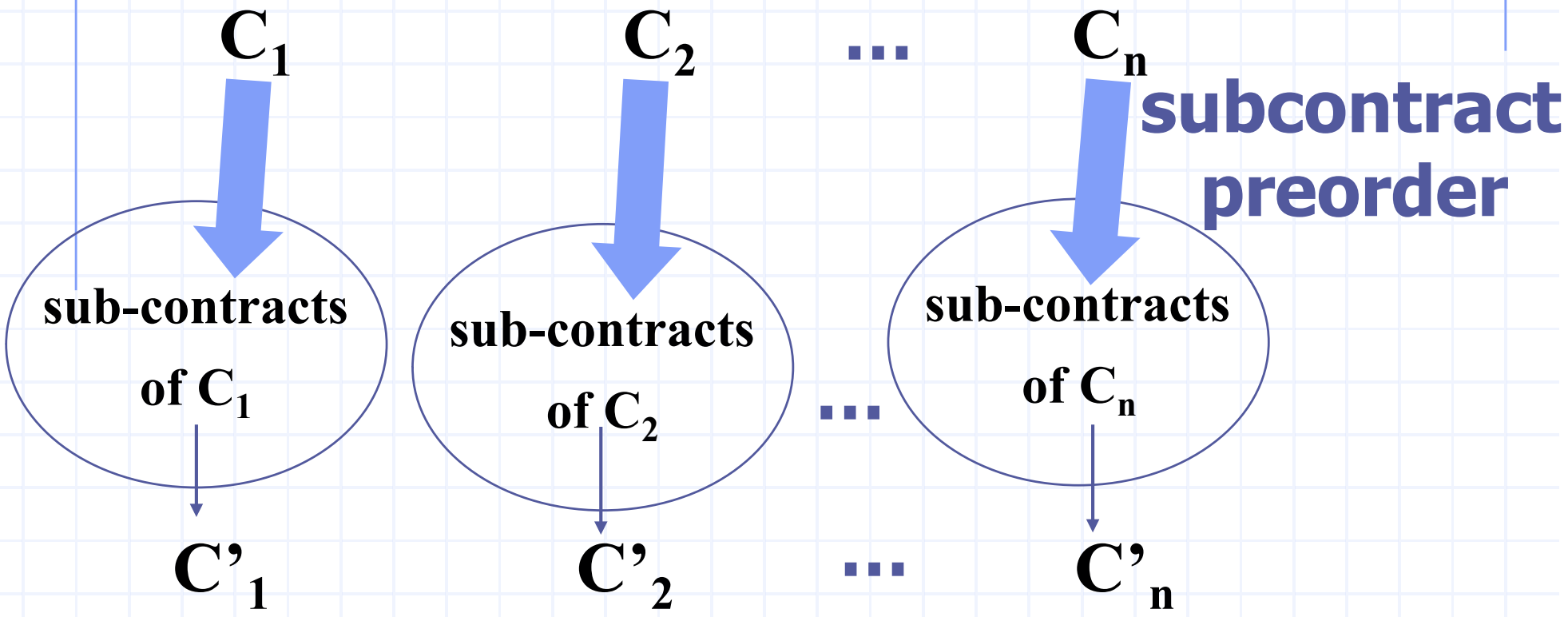
- Preorder ≤ between contracts C:
  - C′ ≤ C means C′ is a subcontract of C



subcontract preorder

Definition of Preorder Induced from Independence Property

Given a correct composition with initial contracts:



The composition is still correct after (independent) replacement of initial contracts with subcontracts

Singular Refinement is Implied

- In particular (since $\leq$ is a preorder, it is reflexive, i.e. $C_i \leq C_i$):

  $C'_i \leq C_i$ implies C'_i must comply with the other initial contracts $C_1,\ldots,C_{i-1},C_{i+1},\ldots,C_n$

Buyer-Seller-Bank example strikes back...

- Remember that for the Bank:
  Receive(PayDescr).P + Receive(eBookStore).Q
  should be a refinement for
  Receive(PayDescr).P
  (this because we assumed that eBookStore was not invoked from Buyer and Seller)
- This kind of refinement requires knowledge about input-output alphabets of the considered contracts!

Exploiting Knowledge About Interface of Initial Contracts

- Formally, we parametrize subcontract preorders with information about what we assume
  - outputs O: operations that can be invoked by the other initial contracts
  - inputs I: operations that can be received by the other initial contracts

  We write $C' \leq_{I,O} C$
- Such a knowledge is usable in independent retrieval because it can be extracted from the initial contracts before starting the independent retrieval phase

Negative result

- We cannot define a "refinement" as the maximal subcontract pre-order!
- Consider the system $[\,\overline{\mathbf{a}}\,] \mid [\,\mathbf{a}\,]$: we could have a preorder $\leq'$ for which

$$\overline{\mathbf{a}} + \mathbf{b}.\mathbf{0} \leq'_{\mathcal{N}\text{-}\{\mathbf{b}\},\mathcal{N}\text{-}\{\mathbf{b}\}} \overline{\mathbf{a}}$$

and a preorder $\leq''$ for which

$$\mathbf{a} + \overline{\mathbf{b}} \leq''_{\mathcal{N}\text{-}\{\mathbf{b}\},\mathcal{N}\text{-}\{\mathbf{b}\}} \mathbf{a}$$

but no preorder $\leq$ could have both

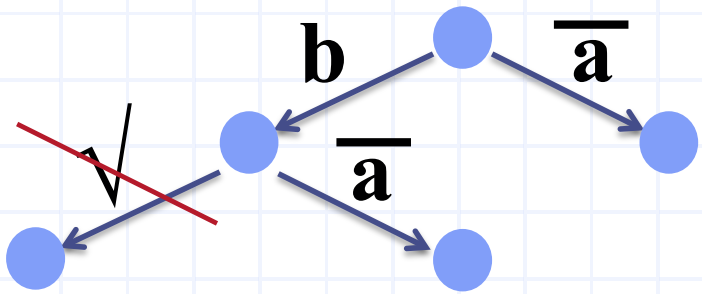
$$\overline{\mathbf{a}} + \mathbf{b}.\mathbf{0} \leq_{\mathcal{N}\text{-}\{\mathbf{b}\},\mathcal{N}\text{-}\{\mathbf{b}\}} \overline{\mathbf{a}} \qquad \mathbf{a} + \overline{\mathbf{b}} \leq_{\mathcal{N}\text{-}\{\mathbf{b}\},\mathcal{N}\text{-}\{\mathbf{b}\}} \mathbf{a}$$

Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- **Compliance-preserving refinement**
  - General case (negative result)
  - **Output persistence**, located outputs
- Choreography conformance (negative result)
  - Consonance
- Alternative notions of compliance
  - Strong compliance, queue-based compliance
- Conclusion

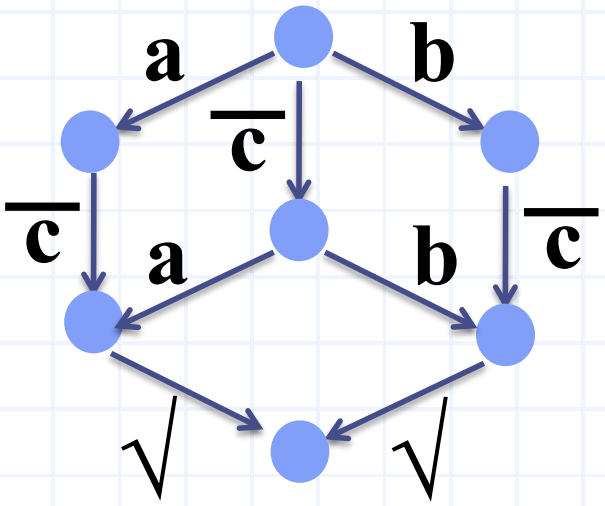
Output persistence

- Assume that services are programmed with languages in which external choices are guarded by inputs (such as in the pick operator of WS-BPEL)
- Their contracts satisfies the following output persistence property:



Example of output persistent contract

- Receive(a)+Receive(b) | Invoke(c)



Output persistence (formally)

Definition 4. (Output persistence) *Let $C \in \mathcal{P}_{con}$ be a contract. It is output persistent if given $C \xrightarrow{w} C'$ with $C' \xrightarrow{\overline{a}}$ then: $C' \not\xrightarrow{\surd}$ and if $C' \xrightarrow{\alpha} C''$ with $\alpha \neq \overline{a}$ then also $C'' \xrightarrow{\overline{a}}$.*

- Main result: If we restrict to output persistent contracts, the union of all subcontract relations turns out to be a subcontract relation
- Consequence: we can define our "refinement" as the maximal subcontract relation!

Main theorem for output persistent contracts

- The preorder $C' \leq^{max}_{I,O} C$ iff for every context P with inputs in I and outputs in O

  $[C] \| P$ is correct implies $[C'] \| P$ is correct
  - is a subcontract preoder
  - includes all subcontract preorders

Subcontracts Cannot Add Reachable Invoke of New Operations

- Due to output persistence subcontracts cannot add reachable outputs on new types
  - otherwise subcontract compliance w.r.t initial contracts is not preserved
- For example: $\mathbf{a} + \overline{\mathbf{c.b}} \leq_{N-\{c\},N-\{c\}} \mathbf{a}$ but not $\mathbf{a} + \overline{\mathbf{a.b}} \leq_{N-\{c\},N-\{c\}} \mathbf{a}$
- Executable additional actions on new types can be just inputs so interaction on new types between subcontracts cannot be generated

Input Knowledge Independence

- If $\mathbf{I'}$,$\mathbf{I''}$ include types of outputs in $\mathbf{C}$:

$$\mathbf{C'} \leq^{max}_{\mathbf{I'},\mathbf{O}} \mathbf{C} \text{ iff } \mathbf{C'} \leq^{max}_{\mathbf{I''},\mathbf{O}} \mathbf{C}$$

  - because subcontracts cannot add reachable outputs on new types

- We just use $\leq^{max}_{\mathbf{O}}$ to stand for $\leq^{max}_{N,\mathbf{O}}$

Output knowledge

- Contrary to inputs, enlarging output types of contexts decreases allowed subcontracts
- In fact $\overline{\mathbf{a}} + \mathbf{b} \leq_{N,N-\{\mathbf{b}\}} \overline{\mathbf{a}}$
  but not $\overline{\mathbf{a}} + \mathbf{b} \leq_{N,N} \overline{\mathbf{a}}$
  Consider for instance the correct system $[\overline{\mathbf{a}}] \mid [\mathbf{a}.\mathbf{b}] \mid [\overline{\mathbf{b}}]$ and the incorrect one $[\overline{\mathbf{a}}+\mathbf{b}] \mid [\mathbf{a}.\mathbf{b}] \mid [\overline{\mathbf{b}}]$

Output Knowledge Allows Extension with Additional Input Types

- Problem reduced to $\leq$ (meaning $\leq_{N,N}$):

  $$C' \leq_{O}^{max} C \text{ iff } C' \setminus\setminus N\text{-}O \leq^{max} C \setminus\setminus N\text{-}O$$

  i.e. to subcontract relation when inputs that cannot be invoked are restricted

- Examples:
  - exploiting knowledge we have $a+b \leq_{\{a\}} a$
  - in addition to $\tau.\overline{a} \leq \tau.\overline{a} + \tau.\overline{b}$ (more deterministic)

Decidible Sound Characterization Based on a Must-like Testing

- Subcontract relation contains a universal quantification over possible contexts
- Sound characterization resorting to a must-testing theory (should-testing [RV05])
  - $\mathbf{C'} \leq_{\mathbf{O}}^{\max} \mathbf{C}$ is implied by

$$\mathcal{NF}(\mathbf{C'} \setminus\setminus \boldsymbol{N}\text{-}\mathbf{O}) \leq_{\text{test}} \mathcal{NF}(\mathbf{C} \setminus\setminus \boldsymbol{N}\text{-}\mathbf{O})$$

  - i.e. $\leq_{\mathbf{O}}$ coarser than testing preorder (and of simulation)

Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- **Compliance-preserving refinement**
  - General case (negative result)
  - Output persistence, **located outputs**
- Choreography conformance (negative result)
  - Consonance
- Alternative notions of compliance
  - Strong compliance, queue-based compliance
- Conclusion

Adoption of a more realistic address-based communication

- Alternative to standard Process Algebra channel-based communication mechanism
  - "$\overline{a}$" can be received by any C doing "a"
- Invokes $\overline{a}_l$ indicate a destination address (location l)
- Every contract C is executed at a distiguished location l, written "$[C]_l$"
  - i.e. from choreography, location = participant

In Addition Output Knowledge Independence !

- If, for every $\mathbf{l}$, $\mathbf{O'}$,$\mathbf{O''}$ include inputs in $\mathbf{C}$:

$$\mathbf{C'} \leq^{max}_{\mathbf{I},\mathbf{O'}} \mathbf{C} \text{ iff } \mathbf{C'} \leq^{max}_{\mathbf{I},\mathbf{O''}} \mathbf{C}$$

  - because compliant contexts of $\mathbf{C}$ cannot perform reachable outputs to $\mathbf{C}$ that it cannot receive

- Now $\overline{\mathbf{a}_\mathbf{l}} + \mathbf{b} \leq_{N,N-\{\mathbf{b}\}} \overline{\mathbf{a}_\mathbf{l}}$
  and also $\overline{\mathbf{a}_\mathbf{l}} + \mathbf{b} \leq_{N,N} \overline{\mathbf{a}_\mathbf{l}}$
  in fact the refined process now cannot intercept the $\mathbf{b}$ message in
  $[\, \overline{\mathbf{a}_\mathbf{l}} + \mathbf{b} \,]_\mathbf{i} \mid [\mathbf{a}.\mathbf{b}]_\mathbf{l} \mid [\, \overline{\mathbf{b}_\mathbf{l}} \,]_\mathbf{j}$

Independence on Output Knowledge allows Extension with Input Types !

- Therefore, from $C' \leq^{max} C$ iff $C' \leq^{max}_{N,I(C)} C$, we can reduce the problem:

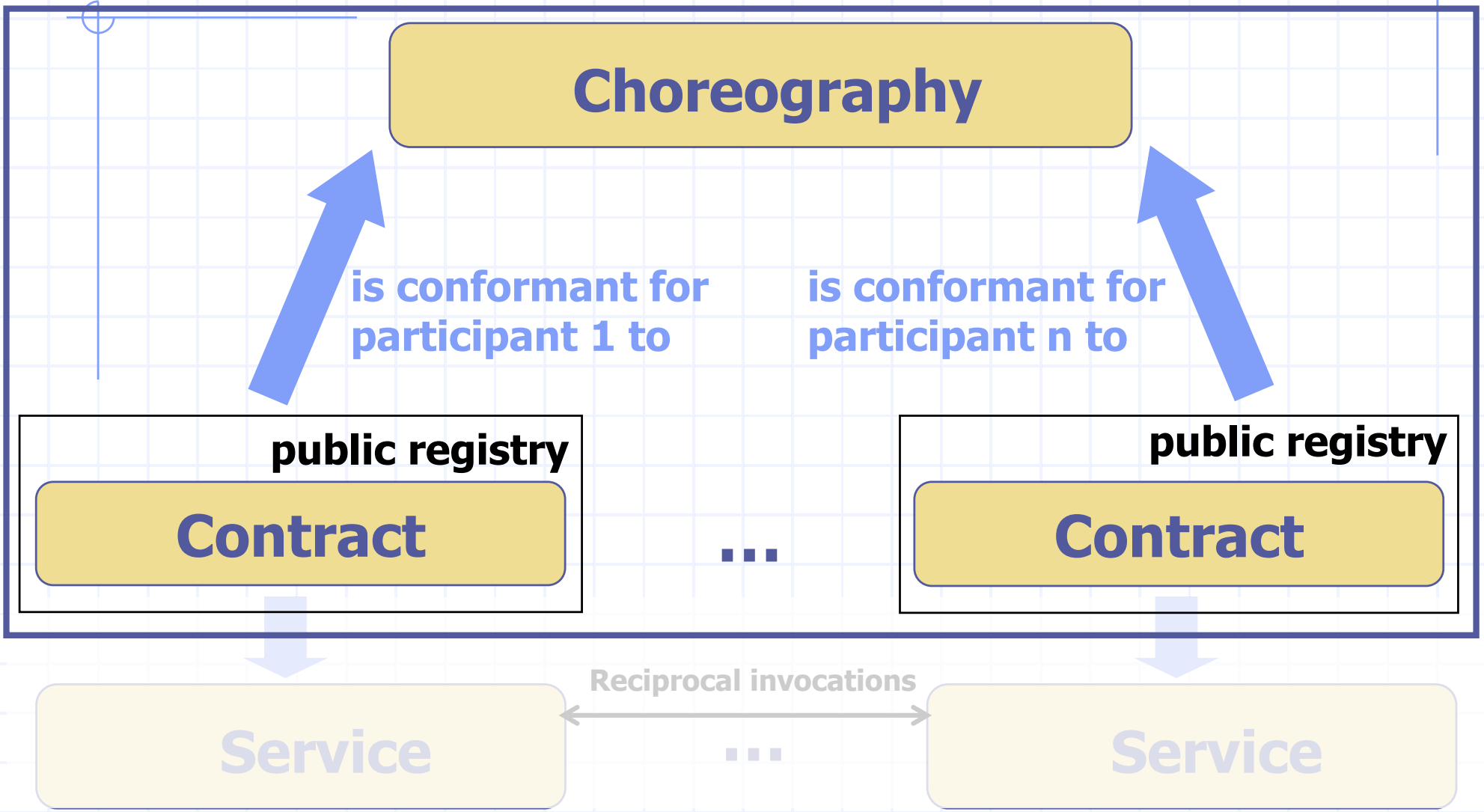
$$C' \leq^{max} C \text{ iff } C' \backslash\backslash N\text{-}I(C) \leq^{max} C$$

  i.e. to subcontract relation when inputs that cannot be invoked are restricted
- Hence knowledge is no longer needed!
- Analogous sound characterization based on a must-like testing scenario

Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- Compliance-preserving refinement
  - General case (negative result)
  - Output persistence, located outputs
- **Choreography conformance** (negative result)
  - Consonance
- Alternative notions of compliance
  - Strong compliance, queue-based compliance
- Conclusion

Choreography Conformance



Choreography Language in Essence

- Choreography **H**:

$$H ::= a_{r \to s} \mid H;H \mid H+H \mid H|H \mid H^*$$

r invokes the operation a of s

Notion of Implementation

- Given a choreography $\mathbf{H}$, a system $\mathbf{P}$ implements $\mathbf{H}$ if:
  - $\mathbf{P}$ is a composition of compliant contracts
  - Each (completed weak) trace of $\mathbf{P}$ has a corresponding (completed) trace of $\mathbf{H}$
    - all computations of $\mathbf{P}$ are correct conversations according to the choreography $\mathbf{H}$

Examples of choreography Implementations:

- Given the choreography:

$$\mathbf{Request}_{\mathbf{Alice}\rightarrow\mathbf{Bob}};\ (\mathbf{Accept}_{\mathbf{Bob}\rightarrow\mathbf{Alice}} + \mathbf{Reject}_{\mathbf{Bob}\rightarrow\mathbf{Alice}})$$

The following are implementations:

Examples of choreography Implementations:

- Given the choreography:

$$\mathbf{Request}_{\mathbf{Alice}\rightarrow\mathbf{Bob}};\ (\mathbf{Accept}_{\mathbf{Bob}\rightarrow\mathbf{Alice}} + \mathbf{Reject}_{\mathbf{Bob}\rightarrow\mathbf{Alice}})$$

The following are implementations:

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}}.(\mathbf{Accept}+\mathbf{Reject})]_{\mathbf{Alice}} \mid$$
$$[\mathbf{Request}.(\tau.\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\tau.\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

Examples of choreography Implementations:

- Given the choreography:

$$\mathbf{Request}_{\mathbf{Alice}\rightarrow\mathbf{Bob}};\ (\mathbf{Accept}_{\mathbf{Bob}\rightarrow\mathbf{Alice}} + \mathbf{Reject}_{\mathbf{Bob}\rightarrow\mathbf{Alice}})$$

The following are implementations:

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}}.(\mathbf{Accept}+\mathbf{Reject})]_{\mathbf{Alice}} \mid [\mathbf{Request}.(\tau.\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\tau.\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}}.(\mathbf{Accept}+\mathbf{Reject}+\mathbf{Retry})]_{\mathbf{Alice}} \mid [\mathbf{Request}.(\tau.\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\tau.\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

Examples of choreography Implementations:

- Given the choreography:

$$\mathbf{Request}_{\mathbf{Alice}\rightarrow\mathbf{Bob}};\ (\mathbf{Accept}_{\mathbf{Bob}\rightarrow\mathbf{Alice}} + \mathbf{Reject}_{\mathbf{Bob}\rightarrow\mathbf{Alice}})$$

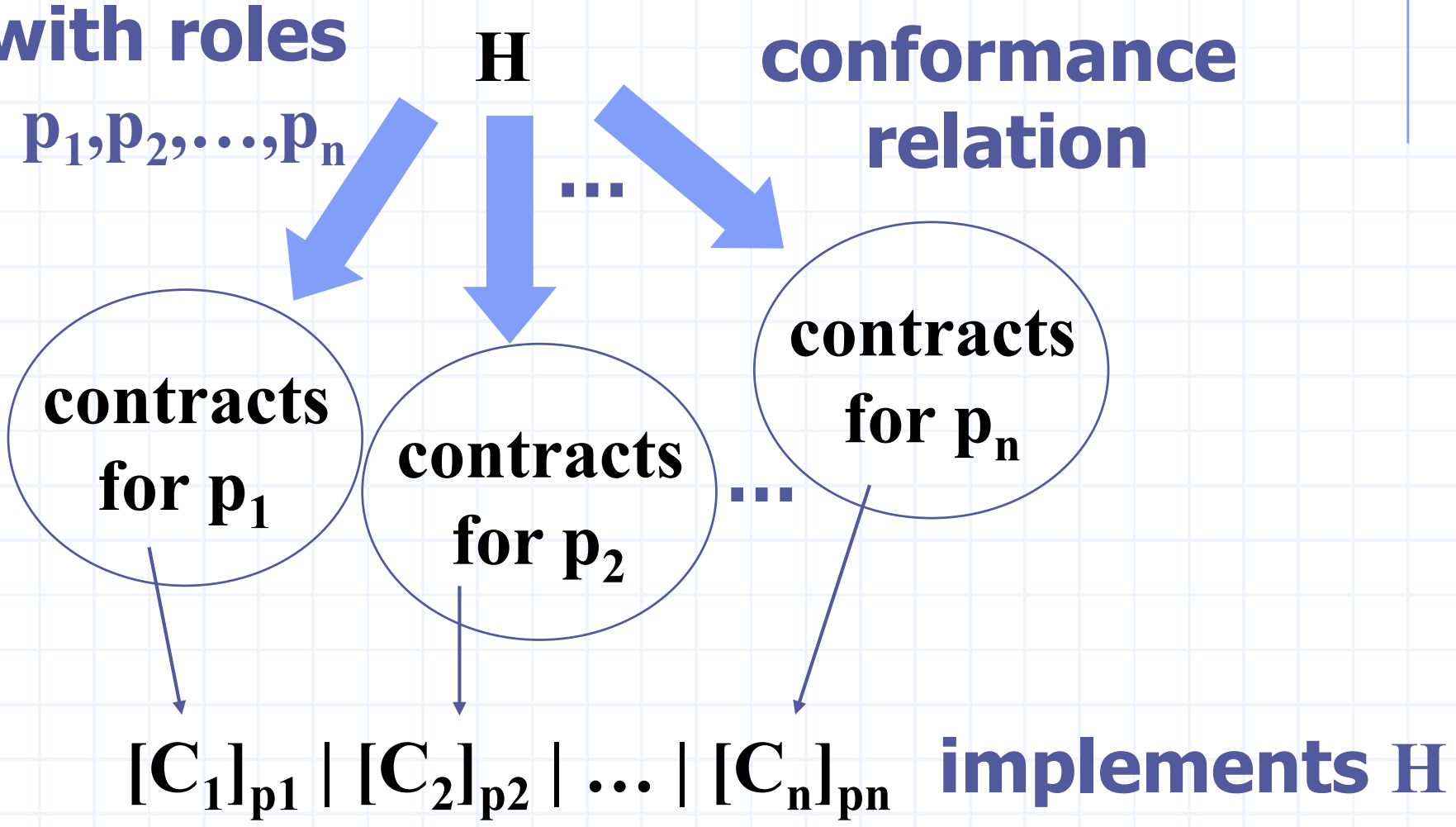
The following are implementations:

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}}.(\mathbf{Accept}+\mathbf{Reject})]_{\mathbf{Alice}} \mid [\mathbf{Request}.(\tau.\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\tau.\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}}.(\mathbf{Accept}+\mathbf{Reject}+\mathbf{Retry})]_{\mathbf{Alice}} \mid [\mathbf{Request}.(\tau.\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\tau.\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}}.(\mathbf{Accept}+\mathbf{Reject}+\mathbf{Retry})]_{\mathbf{Alice}} \mid [\mathbf{Request}.\overline{\mathbf{Accept}}_{\mathbf{Alice}}]_{\mathbf{Bob}}$$

Definition of Relation Induced from Independence Property



No Maximal Choreography Conformance Relation

- Consider the choreography

$$\mathbf{a}_{r\to s} \mid \mathbf{b}_{r\to s}$$

that can be implemented as:

$$[\,\overline{\mathbf{a}}_s.\overline{\mathbf{b}}_s + \overline{\mathbf{b}}_s.\overline{\mathbf{a}}_s]_r \mid [\,\tau.\mathbf{a}.\mathbf{b} + \tau.\mathbf{b}.\mathbf{a}\,]_s$$

$$[\tau.\overline{\mathbf{a}}_s.\overline{\mathbf{b}}_s + \tau.\overline{\mathbf{b}}_s.\overline{\mathbf{a}}_s\,]_r \mid [\mathbf{a}.\mathbf{b} + \mathbf{b}.\mathbf{a}]_s$$

but not as:

$$[\tau.\overline{\mathbf{a}}_s.\overline{\mathbf{b}}_s + \tau.\overline{\mathbf{b}}_s.\overline{\mathbf{a}}_s\,]_r \mid [\,\tau.\mathbf{a}.\mathbf{b} + \tau.\mathbf{b}.\mathbf{a}\,]_s$$

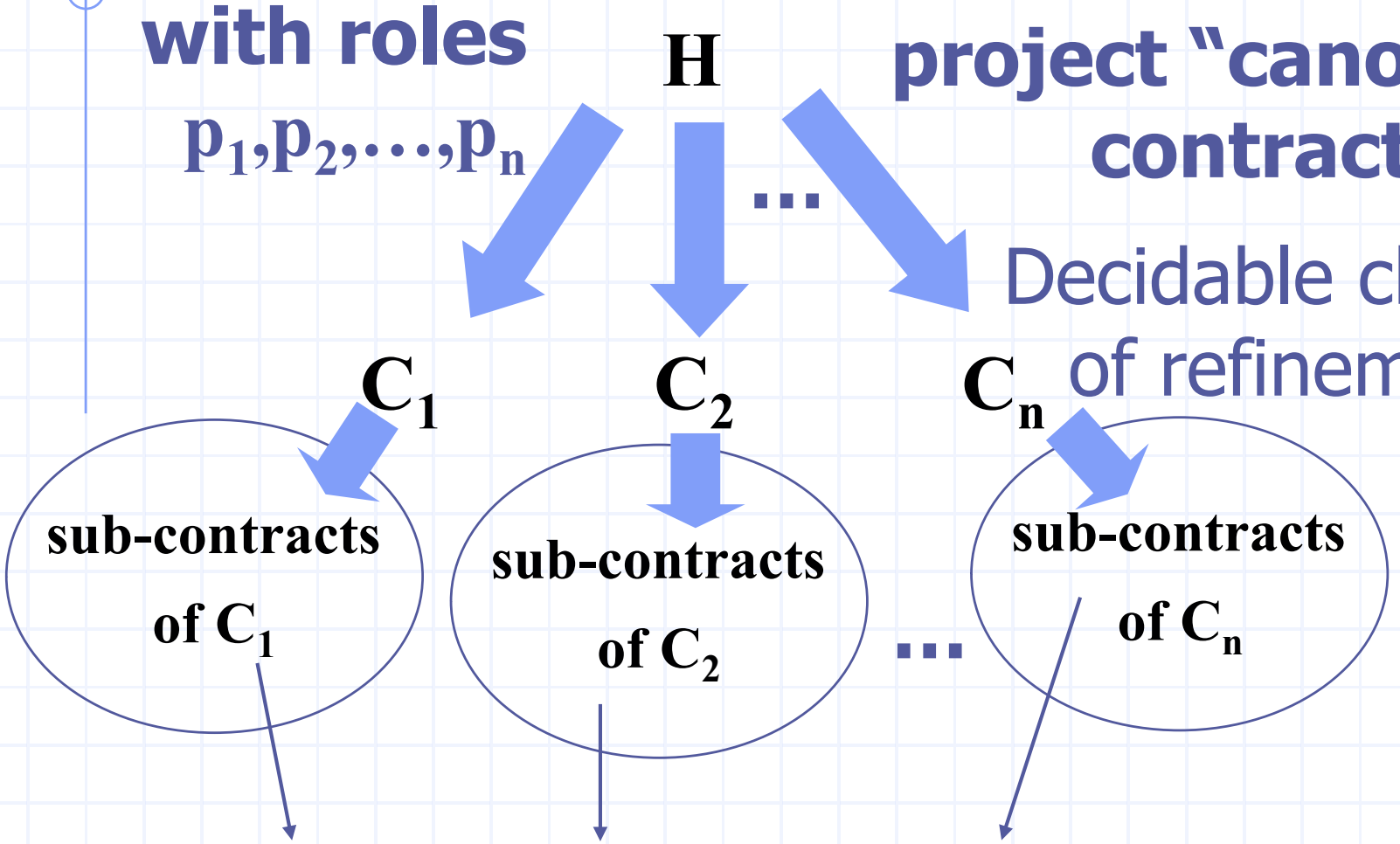
Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- Compliance-preserving refinement
  - General case (negative result)
  - Output persistence, located outputs
- Choreography conformance (negative result)
  - **Consonance**
- Alternative notions of compliance
  - Strong compliance, queue-based compliance
- Conclusion

Consonance:
a "canonical" conformance relation

with roles

$p_1, p_2, \ldots, p_n$



implements H

The "Canonical" Projection

- $\mathbf{H}$ is well-formed if the system $\mathbf{P}$, achieved via canonical projections, implements $\mathbf{H}$
- Canonical projection $[[\ ]]_t$:

$$[[\ a_{r\to s}\ ]]_t = \begin{cases} \tau.\overline{a_s} & \text{if } t=r \\ a & \text{if } t=s \\ 1 & \text{otherwise} \end{cases}$$

$$[[H;H']]_t=[[H]]_t\,;\,[[H']]_t \qquad [[H|H']]_t=[[H]]_t \mid [[H']]_t$$

$$[[H+H']]_t=[[H]]_t + [[H']]_t \qquad [[H^*]]_t=[[H]]_t^*$$

Decidible Sound Characterization Based on a Must-like Testing

- The sound characterization of the consonance relation between a choreography $\mathbf{H}$ and a contract $\mathbf{C}$ playing a role $\mathbf{r}$ given by

$$\mathcal{NF}(\mathbf{C} \setminus\setminus N\text{-}\mathbf{I}([[\mathbf{H}]]_{\mathbf{r}})) \leq_{\text{test}} \mathcal{NF}([[\mathbf{H}]]_{\mathbf{r}})$$

  is a conformance relation

  - proof exploits pre-congruence of testing (trace inclusion is implied by testing)

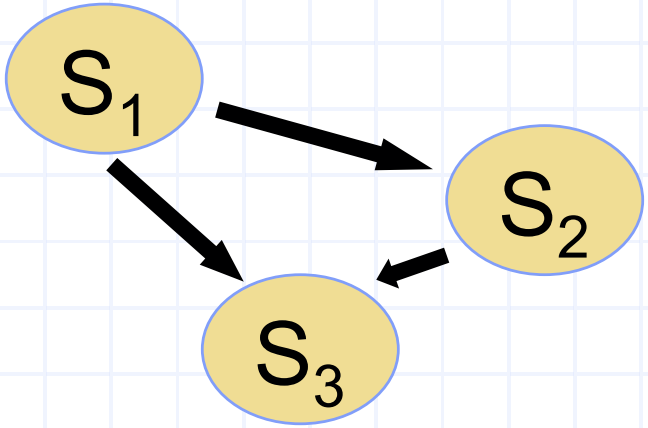
Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- Compliance-preserving refinement
  - General case (negative result)
  - Output persistence, located outputs
- Choreography conformance (negative result)
  - Consonance
- **Alternative notions of compliance**
  - **Strong compliance**, queue-based compliance
- Conclusion

Alternatives to Standard Compliance: Strong Compliance

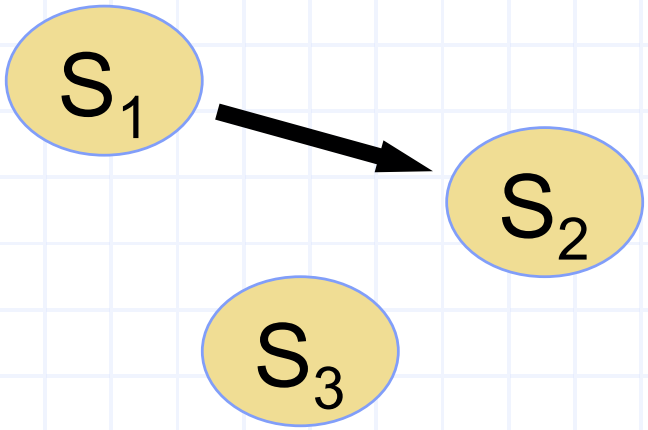
- "Standard" Contract Compliance:
  - **S_1: invoke(a).invoke(b)**
  - **S_2: receive(a).invoke(c)**
  - **S_3: receive(c).receive(b)**

Any doubt?



Alternatives to Standard Compliance: Strong Compliance

- Let us give a more careful look:
  - S_1: **invoke(a).invoke(b)**
  - S_2: **receive(a).invoke(c)**
  - S_3: **receive(c).receive(b)**

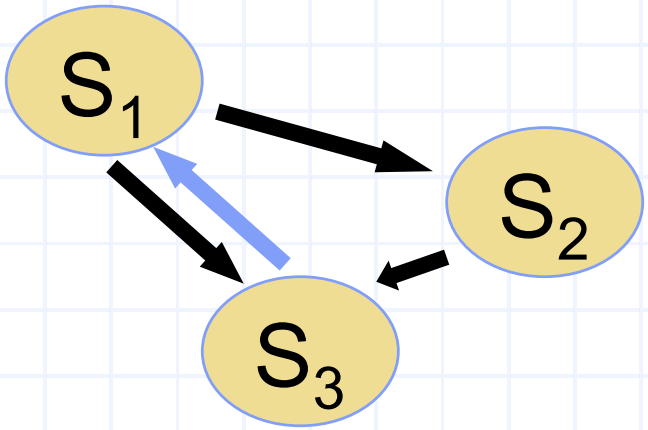


- These services are not **strongly** compliant !!

- **Strong compliance** requires that the receptors should be always ready

Alternatives to Standard Compliance: Strong Compliance

- "Strongly" Compliant Contracts:
  - **S_1: invoke(a).receive(b).invoke(b)**
  - **S_2: receive(a).invoke(c)**
  - **S_3: receive(c).invoke(b).receive(b)**



Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- Compliance-preserving refinement
  - General case (negative result)
  - Output persistence, located outputs
- Choreography conformance (negative result)
  - Consonance
- **Alternative notions of compliance**
  - Strong compliance, **queue-based compliance**
- Conclusion

Alternatives to Standard Compliance: Queue-based Comp.

- Study of asynchronous compliance: each service has a queue for every operation (input)
- Communication involves two events:
  - "$\overline{a_l}$" inserts in queue "a" at location "l"
  - "a" retrieves from local "a" queue
- Example:
  $[\ \overline{a}_s.\overline{b}_s\ ]_r$ is now compliant with $[\ b.a\ ]_s$

Plan of the talk

- Preliminaries
  - Choreography, contract, refinement, conformance
- Compliance-preserving refinement
  - General case (negative result)
  - Output persistence, located outputs
- Choreography conformance (negative result)
  - Consonance
- Alternative notions of compliance
  - Strong compliance, queue-based compliance
- **Conclusion**

Summary of Results

- Direct conformance w.r.t. the whole choreography: maximal relation does not exist (all kinds of compl.)
- Conformance via refinement with knowledge about other initial contracts limited to I/O actions
  (enough to guarantee that refinements that extend the interface are included)
  - "normal" compliance:
    - Unconstrained contracts: maximal relation does not exist
    - Contracts where outputs are internally chosen (output persistence): maximal relation exists and "I" knowledge is irrelevant
    - Output persistent contracts where outputs are directed to a location: maximal relation exists and "I/O" knowledge is irrelevant
  - strong compliance:
    - Unconstrained contracts (where output are directed to a location): maximal relation exists and "I/O" knowledge is irrelevant

Summary of Results

- queue-based compliance:
  - Unconstrained contracts (where output are directed to a location): maximal relation exists and "I/O" knowledge is irrelevant

- Sound characterizations of the relations obtained (apart from the queue based) by resorting to an encoding into (a fair version of) must testing [RV05]
- As a consequence:
  - Algorithm that guarantees compliance
  - Classification of the relations w.r.t. existing pre-orders: coarser than must testing (e.g., they allow external non-determinism on inputs to be added in refinements)

Future work

- Complete characterization of the relations
  - $\tau.a+\tau.b \leq c+c.d$ is not captured by the should-testing based characterization
- Extension with data (types), mobility, sessions/correlation sets, ….

References

- [FHRR04] C. Fournet, C.A.R. Hoare, S.K. Rajamani, and J. Rehof. Stuck-Free Conformance. In CAV'04.
- [CCLP06] S. Carpineti, G. Castagna, C. Laneve, and L. Padovani. A Formal Account of Contracts for Web Services. In WS-FM'06.
- [CHY07] M. Carbone, K. Honda, and N. Yoshida. Structured Communication-Centred Programming for Web Services. In ESOP'07.
- [RV05] A. Rensink and W. Vogler. Fair testing. CTIT Technical Report TR-CTIT-05-64, Dep. Computer Science, Univ. of Twente, 2005.
- [BZ07a] M. Bravetti and G. Zavattaro. Contract based Multi-party Service Composition. In FSEN'07 and "Fundamenta Informaticae".
- [BZ07b] M. Bravetti and G. Zavattaro. Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In SC'07.
- [BZ07c] M. Bravetti and G. Zavattaro. A Theory for Strong Service Compliance. In Coordination'07 and "Mathematical Structures in Computer Science".
- [BZ08] M. Bravetti and G. Zavattaro. Contract Compliance and Choreography Conformance in the Presence of Message Queues. In WS-FM'08.