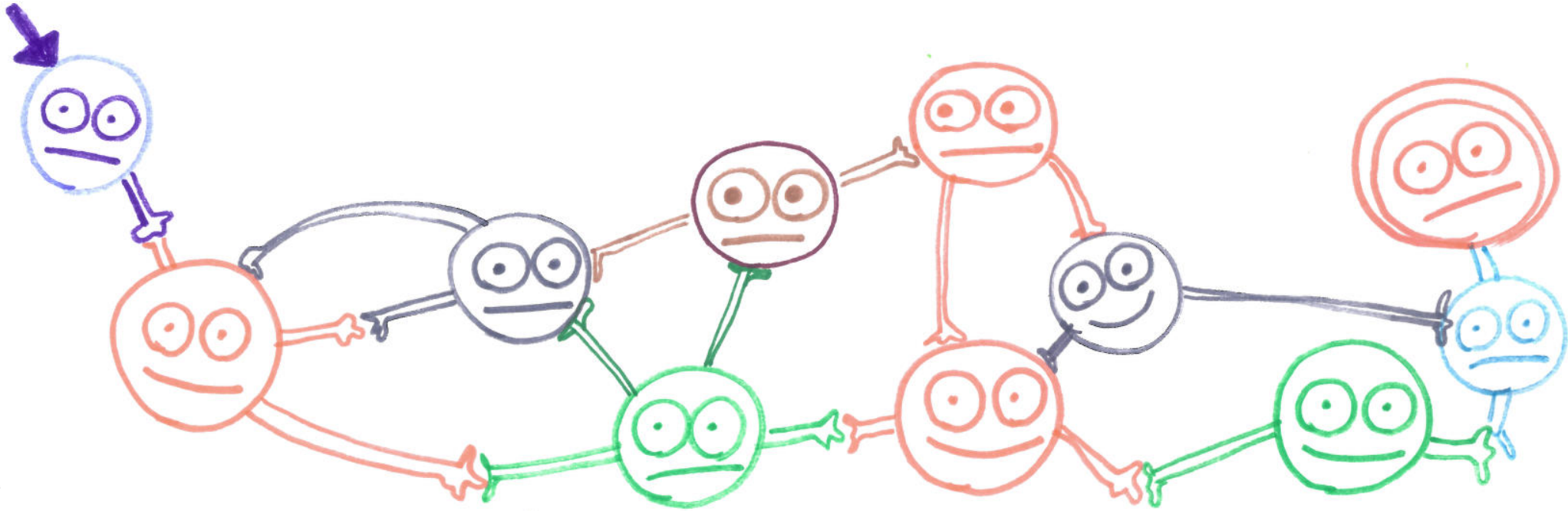
Multiparty Compatibility in Communicating Automata

Synthesis and Characterisation of Multiparty Session Types

Nobuko Yoshida

Pierre-Malo Deniélou



The Kohei Honda Prize for Distributed Systems Queen Mary, University of London

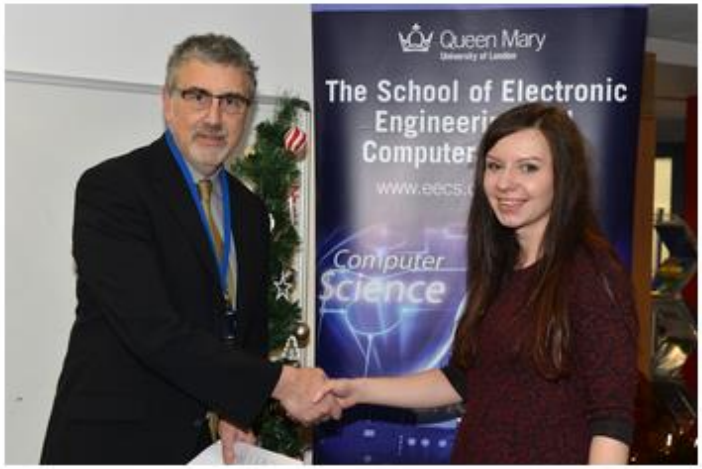
Posted with permission from QMUL on 17th Dec 2013. Original article written by Edmund Robinson.

This prize was instituted in 2013 and is awarded annually to one undergraduate student and one postgraduate student in recognition of their achievement in applying the highest quality scientific and engineering principles in the broad area of Distributed Systems. This is the area in which Dr Honda concentrated most of his teaching, and it is also the area in which he conducted his research. Its primary funding comes from a donation from his family, who wished to commemorate Dr Honda in this way. Additional funding has come from Dr Honda's own ETAPS Award. This prize is sponsored by Springer Verlag, and awarded annually by the ETAPS committee in recognition of an individual's research contribution. Dr Honda received the first such award posthumously, and the awarding panel expressed a wish that the funding be used to supplement this prize fund. The laudation for this award, written by Dr Honda's colleague, Prof Vladimiro Sassone is included later.

About Dr Honda

Kohei Honda was born and lived the first part of his life in Japan. Like many scientists he was fascinated by the idea of finding basic explanatory theories, like the physicists looking for grand unified theories of the universe. Kohei, though, was passionately interested in finding the right basic explanatory theory for the process of computation. Most academics agree that the basic theory

Winners 2013



Ms Anna Pawlicka
2013 winner (Undergraduate) source: QMUL

Mr. Valdmir Negacevshi
2013 winner (Postgraduate) source: QMUL

Selected Publications 2015/2016

- **[FPL'16]** Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk : EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- **[ECOOP'16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC'16]** Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- **[FASE'16]** Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- **[TACAS'16]** Julien Lange, NY: Characteristic Formulae for Session Types.
- **[ESOP'16]** Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- **[POPL'16]** Dominic Orchard, NY: Effects as sessions, sessions as effects .
- **[FSTTCS'15]** Romain Demangeon, NY: On the Expressiveness of Multiparty Session Types.
- **[OOPSLA'15]** Hugo A. López, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco Thudichum Vasconcelos, NY: Protocol-Based Verification of Message-Passing Parallel Programs .
- **[CONCUR'15]** Dimitrios Kouzapas, Jorge A. Pérez, NY: Characteristic Bisimulations for Higher-Order Session Processes .
- **[CONCUR'15]** Laura Bocchi, Julien Lange, NY: Meeting Deadlines Together.
- **[CONCUR'15]** Marco Carbone, Fabrizio Montesi, Carsten Schürmann, NY: Multiparty Session Types as Coherence Proofs.
- **[CC'15]** Nicholas Ng, Jose G.F. Coutinho, NY: Protocols by Default: Safe MPI Code Generation based on Session Types.
- **[PPoPP'15]** Tiago Cogumbreiro, Raymond Hu, Francisco Martins, NY: Dynamic deadlock verification for general barrier synchronisation.
- **[POPL'15]** Julien Lange, Emilio Tuosto, NY: From communicating machines to graphical choreographies.

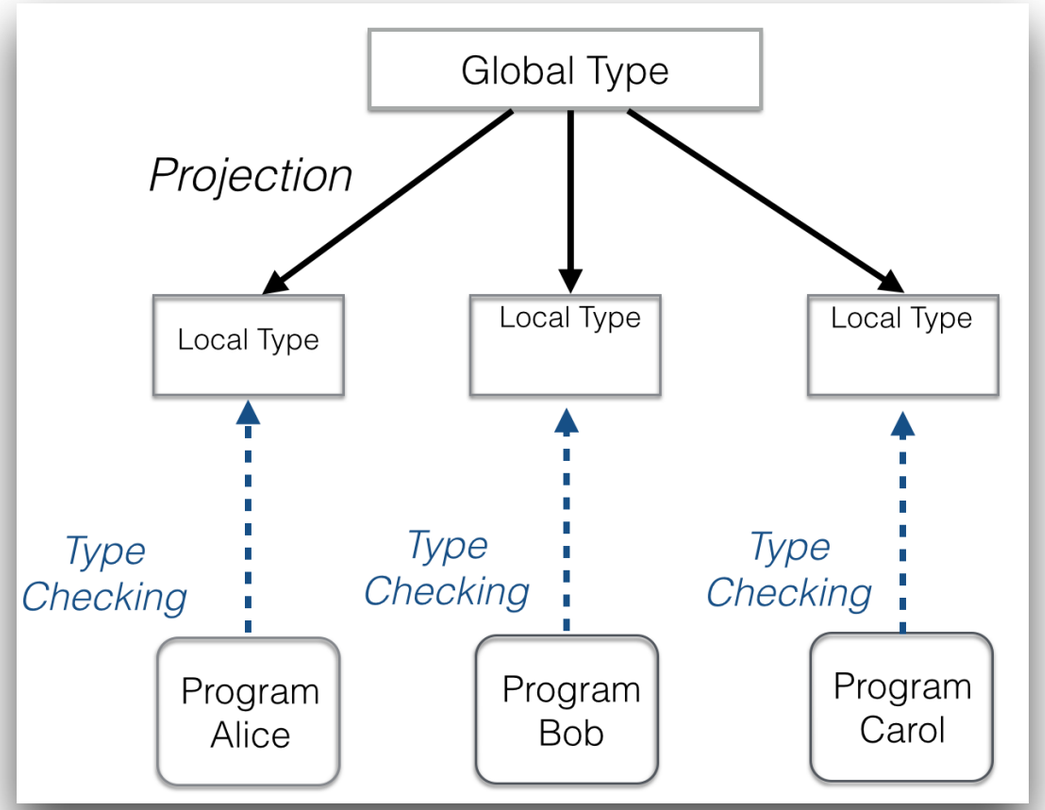
Selected Publications 2015/2016

- **[FPL'16]** Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk : EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- **[ECOOP'16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC'16]** Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- **[FASE'16]** Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- **[TACAS'16]** Julien Lange, NY: Characteristic Formulae for Session Types.
- **[ESOP'16]** Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- **[POPL'16]** Dominic Orchard, NY: Effects as sessions, sessions as effects.
- **[FSTTCS'15]** Romain Demangeon, NY: On the Expressiveness of Multiparty Session Types.
- **[OOPSLA'15]** Hugo A. López, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco Thudichum Vasconcelos, NY: Protocol-Based Verification of Message-Passing Parallel Programs .
- **[CONCUR'15]** Dimitrios Kouzapas, Jorge A. Pérez, NY: Characteristic Bisimulations for Higher-Order Session Processes.
- **[CONCUR'15]** Laura Bocchi, Julien Lange, Nobuko Yoshida: Meeting Deadlines Together.
- **[CONCUR'15]** Marco Carbone, Fabrizio Montesi, Carsten Schürmann, NY: Multiparty Session Types as Coherence Proofs.
- **[CC'15]** Nicholas Ng, Jose G.F. Coutinho, NY: Protocols by Default: Safe MPI Code Generation based on Session Types.
- **[PPoPP'15]** Tiago Cogumbreiro, Raymond Hu, Francisco Martins, NY: Dynamic deadlock verification for general barrier synchronisation.
- **[POPL'15]** Julien Lange, Emilio Tuosto, NY: From communicating machines to graphical choreographies.

OOI Collaboration

- **TCS'16:** Monitoring Networks through Multiparty Session Types. Laura Bocchi , Tzu-Chun Chen , Romain Demangeon , Kohei Honda , Nobuko Yoshida
- **LMCS'16**: Multiparty Session Actors. Rumyana Neykova, Nobuko Yoshida
- **FMSD'15:** Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon , Kohei Honda , Raymond Hu , Rumyana Neykova , Nobuko Yoshida
- **TGC'13:** The Scribble Protocol Language. Nobuko Yoshida , Raymond Hu , Rumyana Neykova , Nicholas Ng

Session Types Overview



- Global session type

$$G = A \to B : \langle U_1 \rangle . B \to C : \langle U_2 \rangle . C \to A : \langle U_3 \rangle$$

- Local session type
  - Slice of global protocol relevant to one role
  - Mechanically derived from a global protocol

$$T_A = !\langle B, U_1 \rangle . ?\langle C, U_3 \rangle$$

- Process language
  - Execution model of I/O actions by session participants
  - Mechanically derived from a global protocol

$$P_A = a[A](x) . x!\langle B, u_1 \rangle . x?(C, y)$$

- (Static) type checking for communication safety and progress

www.scribble.org

Home Getting Started Downloads Documentation Community

Scribble: Describing Multi Party Protocols

Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.

Describe

Scribble is a language for describing multiparty protocols from a global, or endpoint neutral, perspective.

Verify

Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and do not suffer from deadlocks or livelocks.

Project

Endpoint projection is the term used for identifying the responsibility of a particular role (or endpoint) within a protocol.

Implement

Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c) statically verify the code against the projection.

Monitor

Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of a particular endpoint, to ensure it correctly implements the expected behaviour.

Online tool : http://scribble.doc.ic.ac.uk/

```
1  module examples;
2
3  global protocol HelloWorld(role Me, role World) {
4    hello() from Me to World;
5    choice at World {
6      goodMorning1() from World to Me;
7    } or {
8      goodMorning1() from World to Me;
9    }
10 }
11
```

Load a sample | Check | Protocol: examples.HelloWorld | Role: Me | Project | Generate Graph

Interactions with Industries

Adam Bowen @adamnbowen · Sep 15

I didn't even know that session types existed an hour ago, but thanks to Nobuko Yoshida's great talk at **#pwlconf**, I want to learn more.

Nobuko Yoshida

Imperial College, London

DoC researcher to speak at Golang UK conference

by Vicky Kapogianni
20 July 2016

DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research

Click here to add content

.@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16

Interactions with Industries

Distributed Systems vs. Compositionality

Dr. Roland Kuhn
@rolandkuhn — *CTO of Actyx*

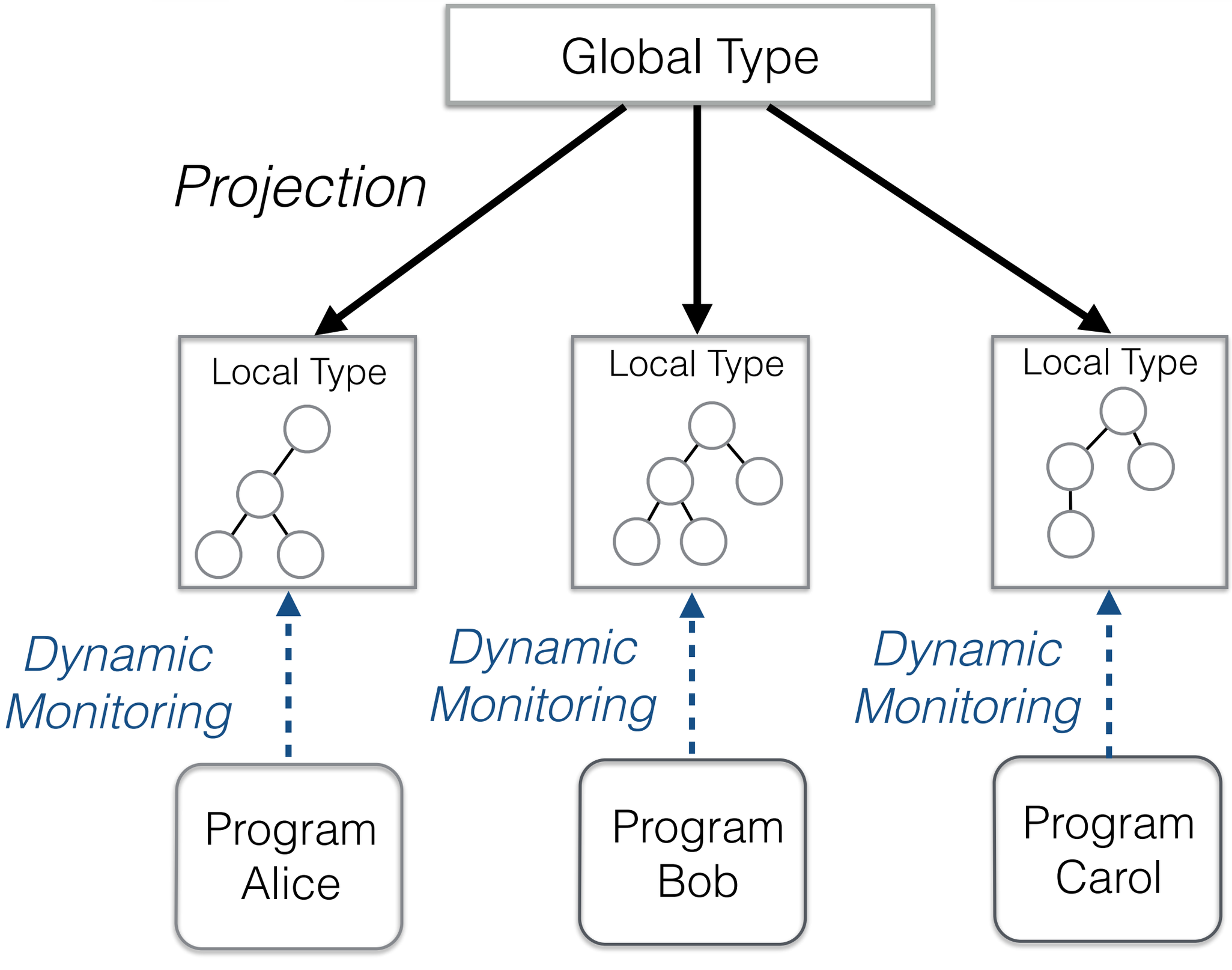
actyx

Current State

- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)

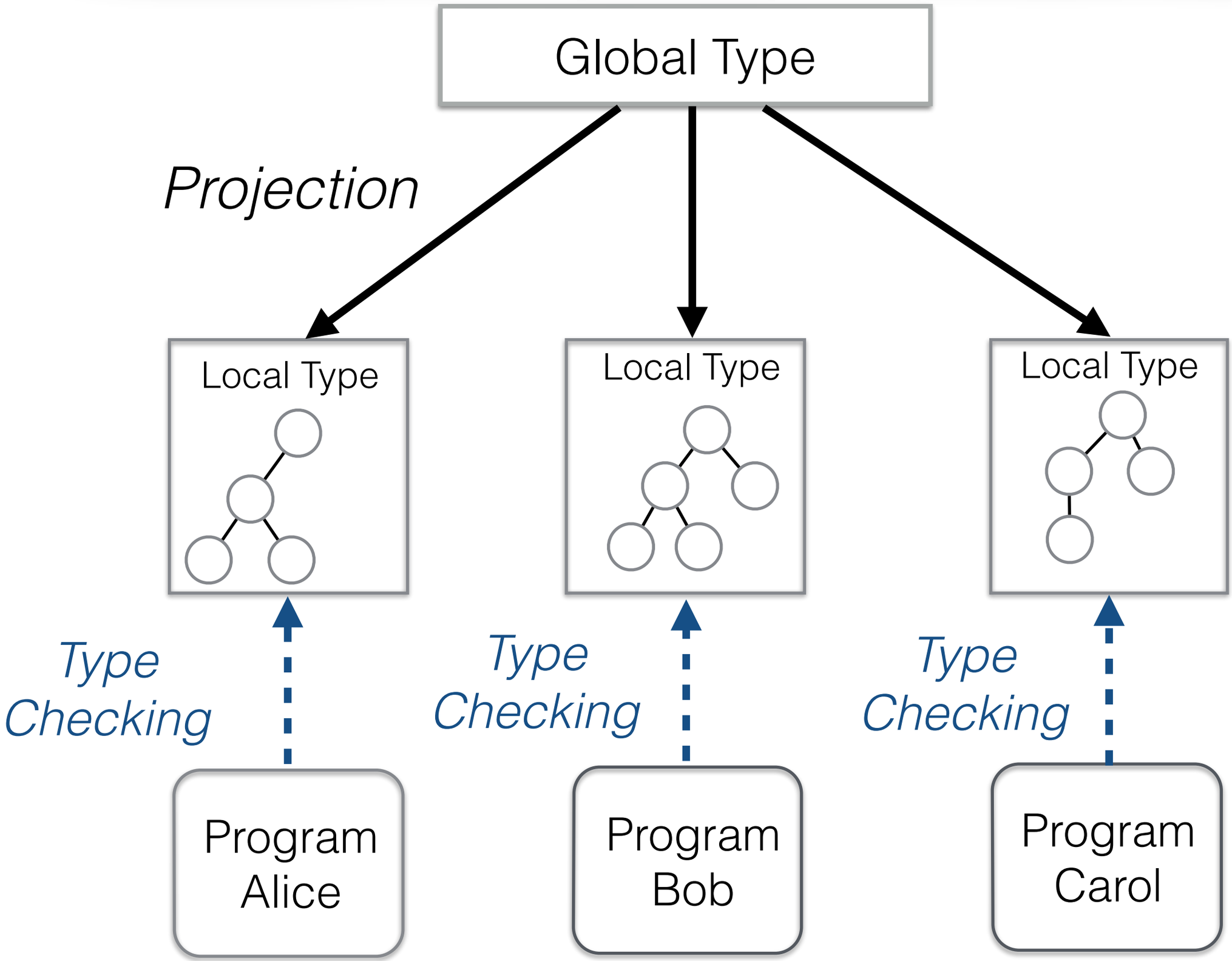
Dynamic Monitoring

[RV'13, COORDINATION'14, FMSD'15]

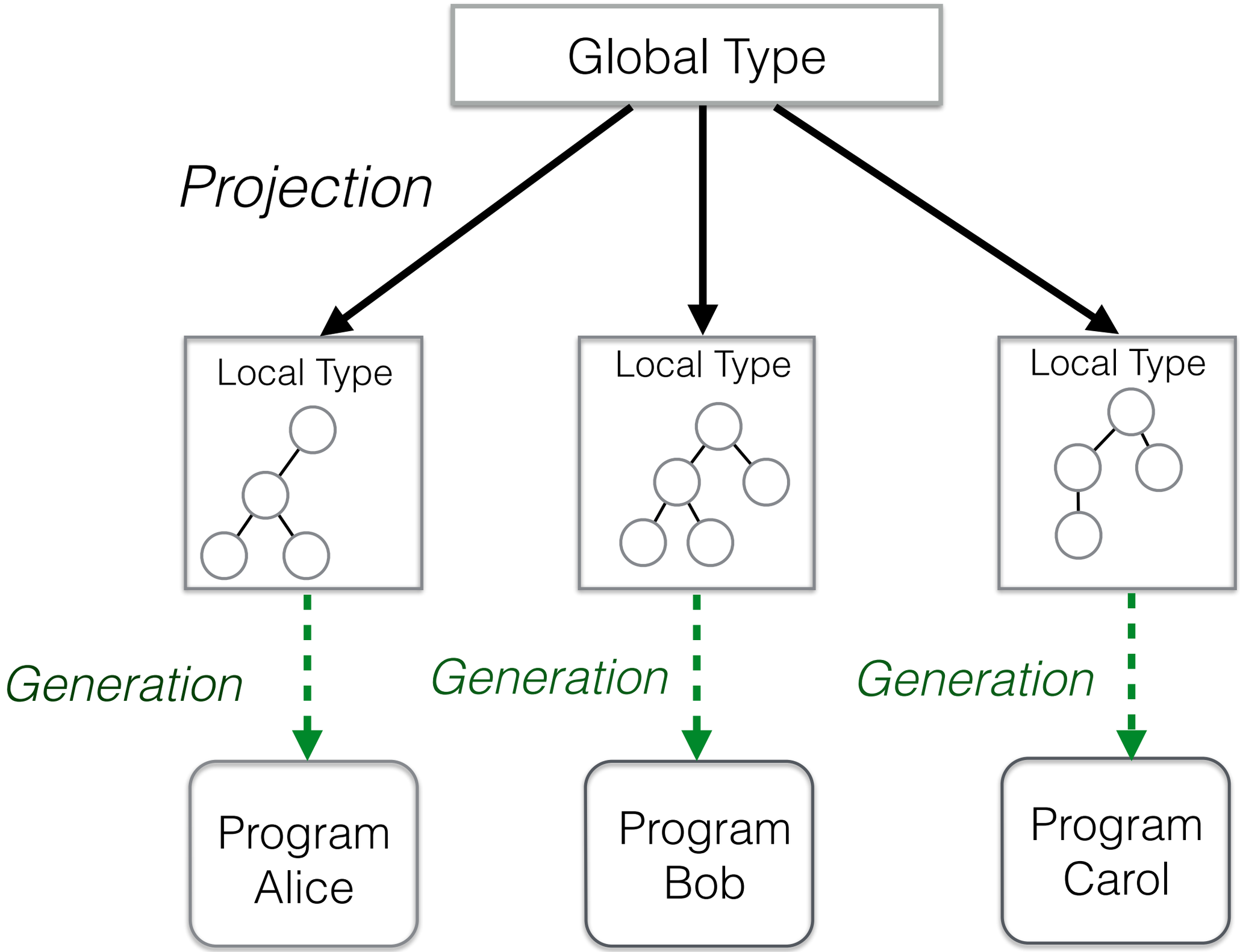


Type Checking

[ECOOP'16, OOPSLA'15, POPL'16]

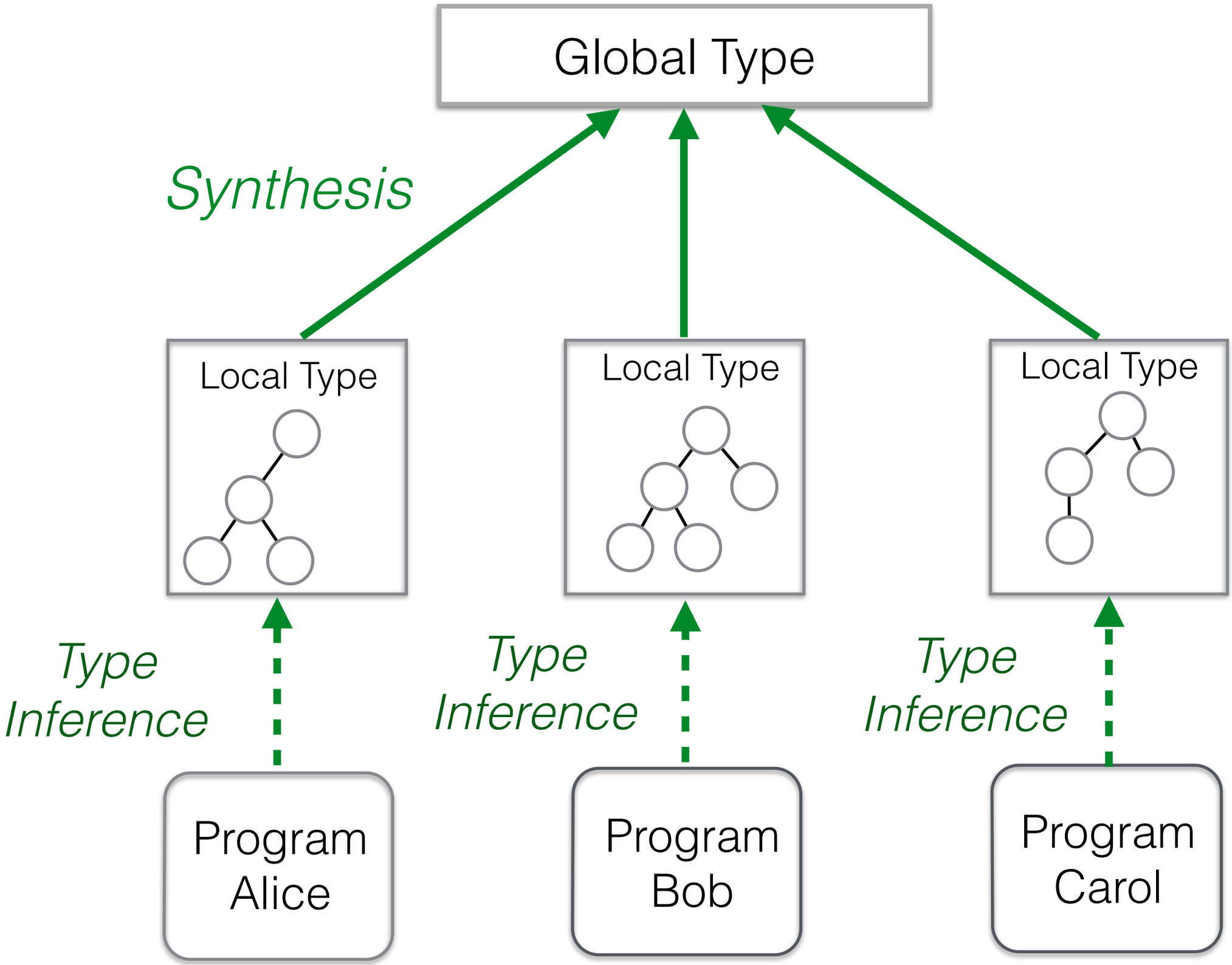


Code Generation [CC'15, FASE'16]



Synthesis

[ICALP'13, POPL'15, CONCUR'15, TACAS'16, CC'16]

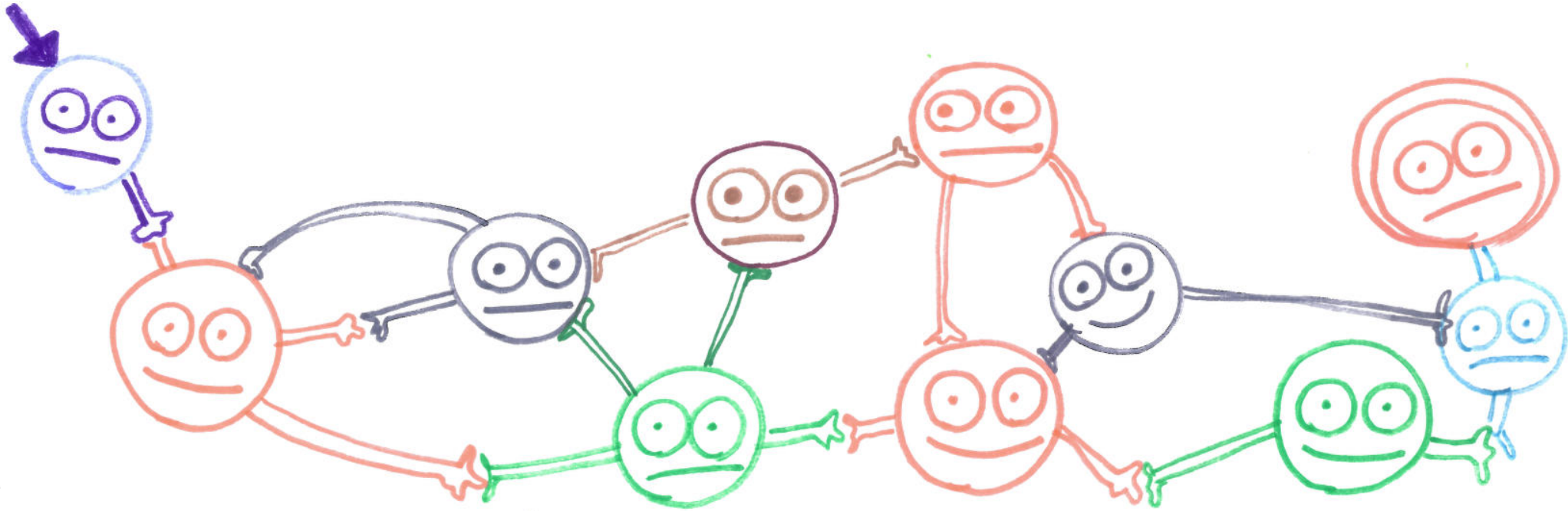


Multiparty Compatibility in Communicating Automata

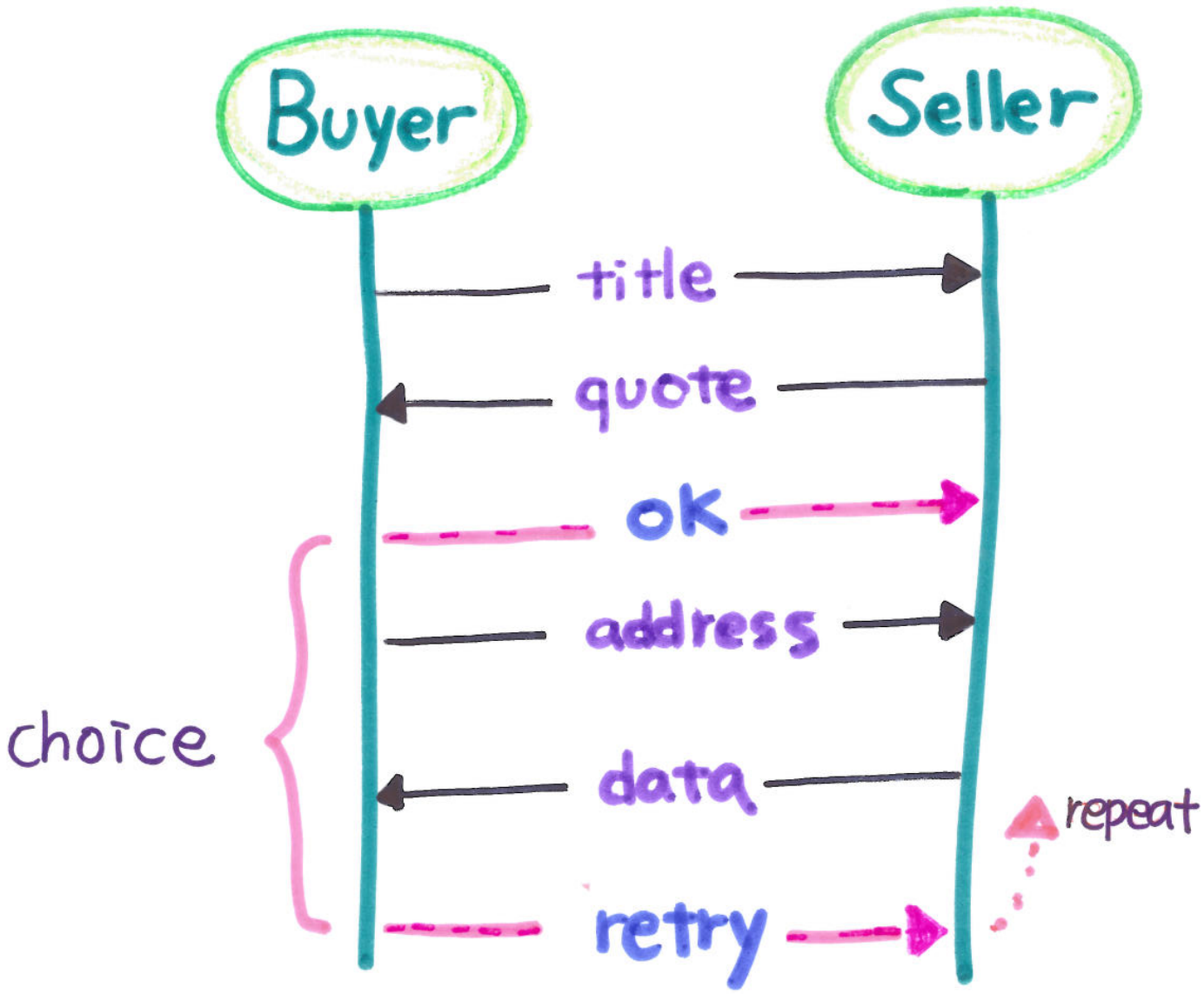
Synthesis and Characterisation of Multiparty Session Types

Nobuko Yoshida

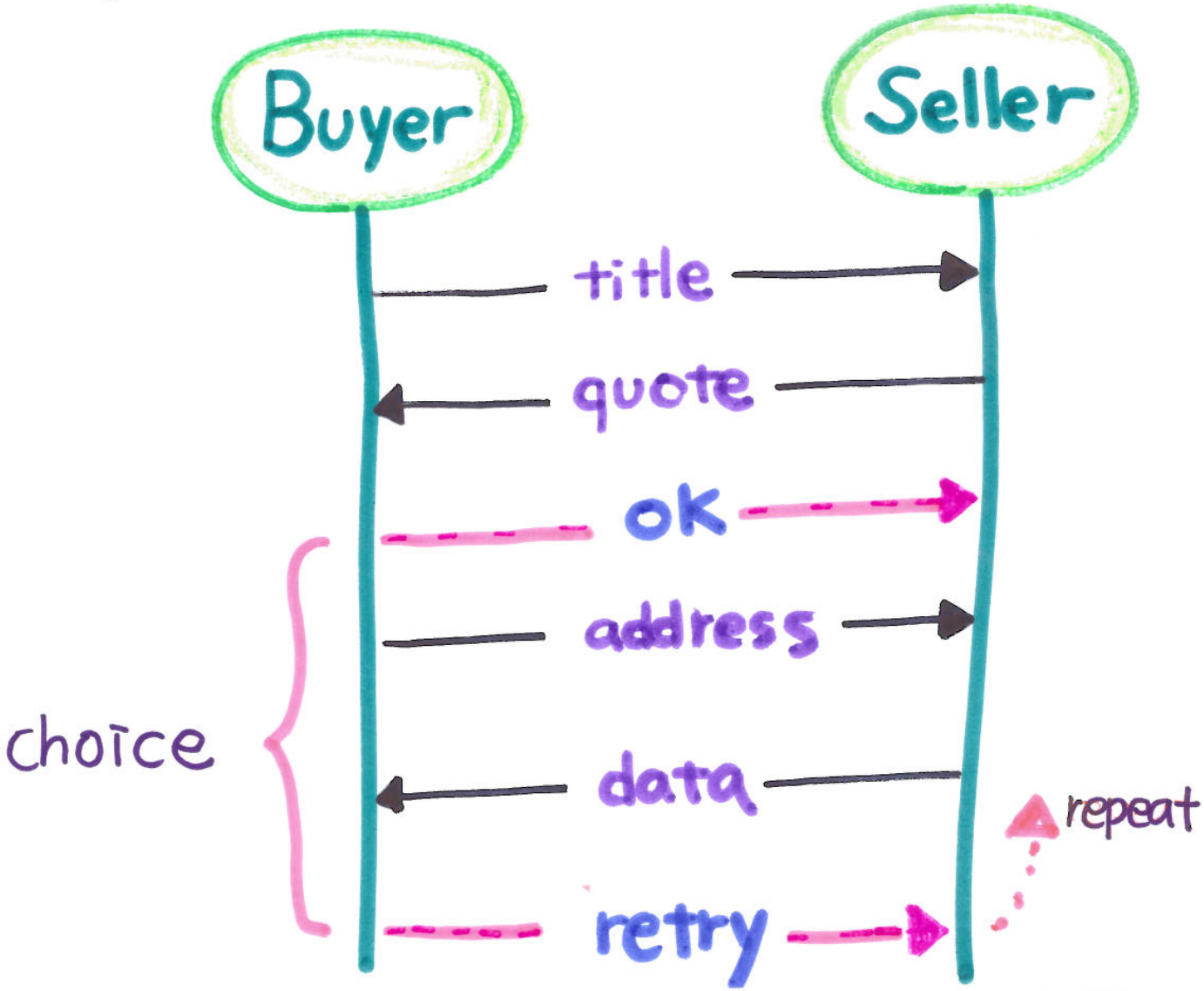
Pierre-Malo Deniélou



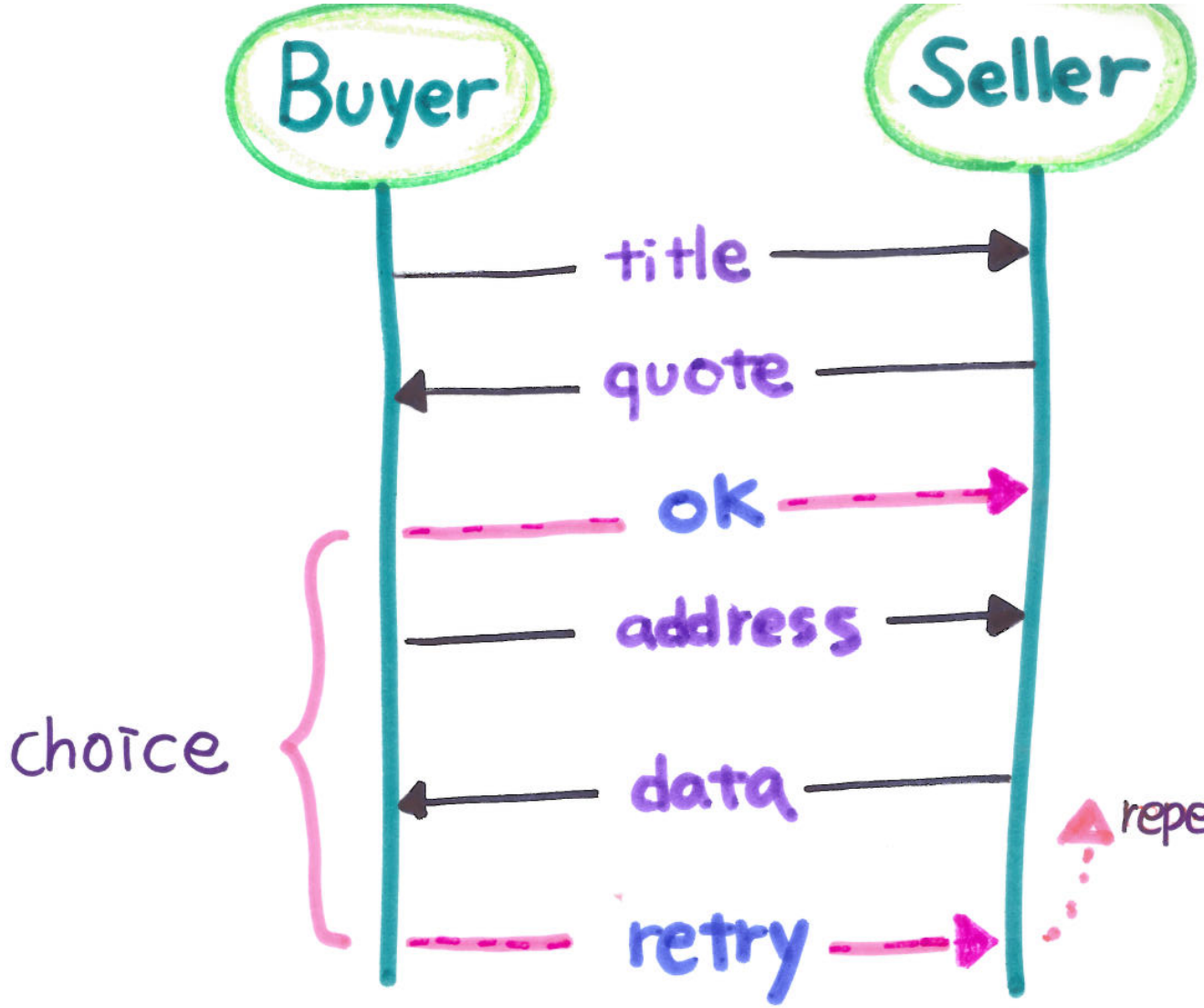
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



μt. ! Title ; ? Quote ; ! { ok : ! Add ; ? Date , retry : t }



P has T

Q has $\overline{T}$

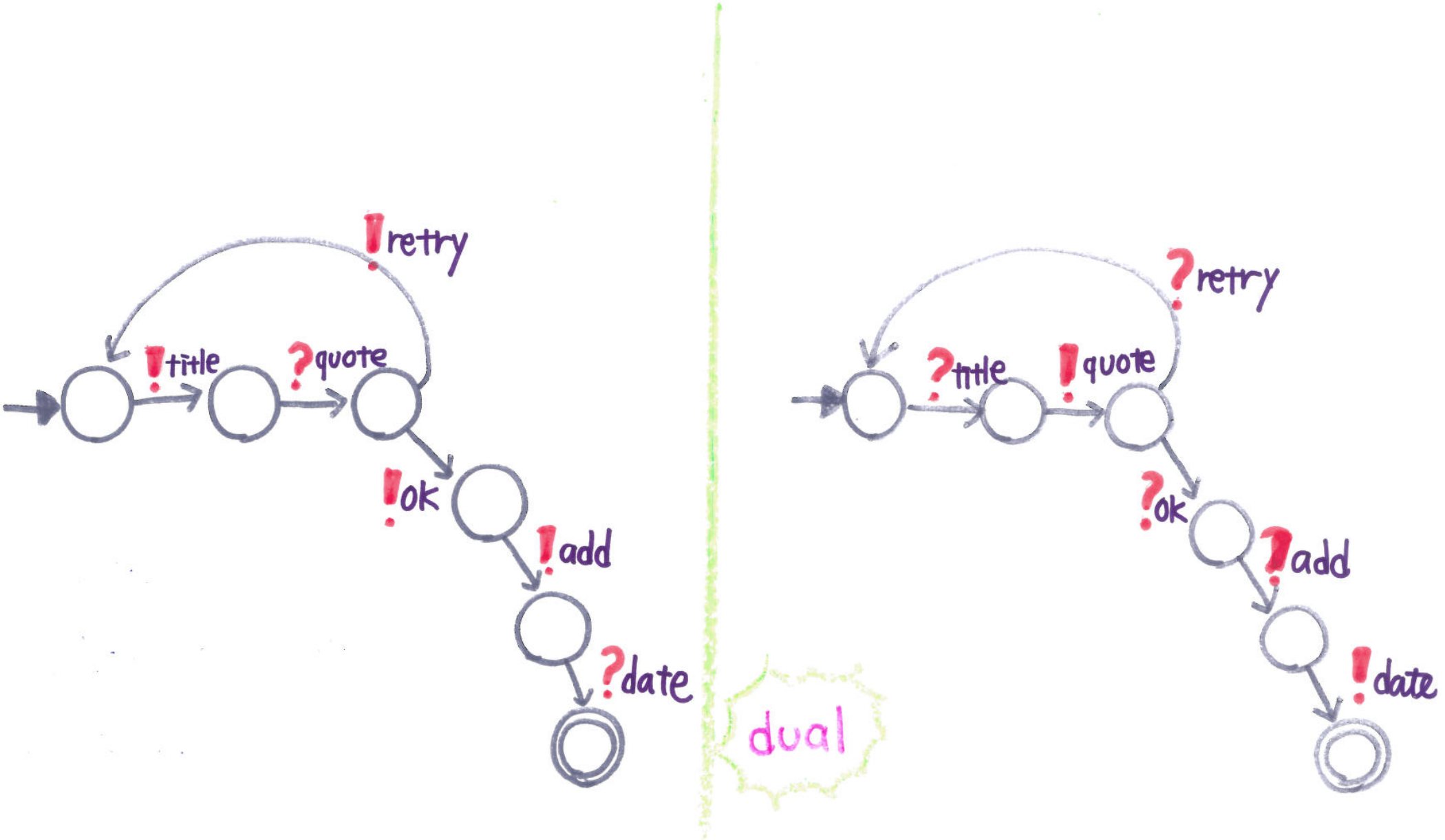
dual

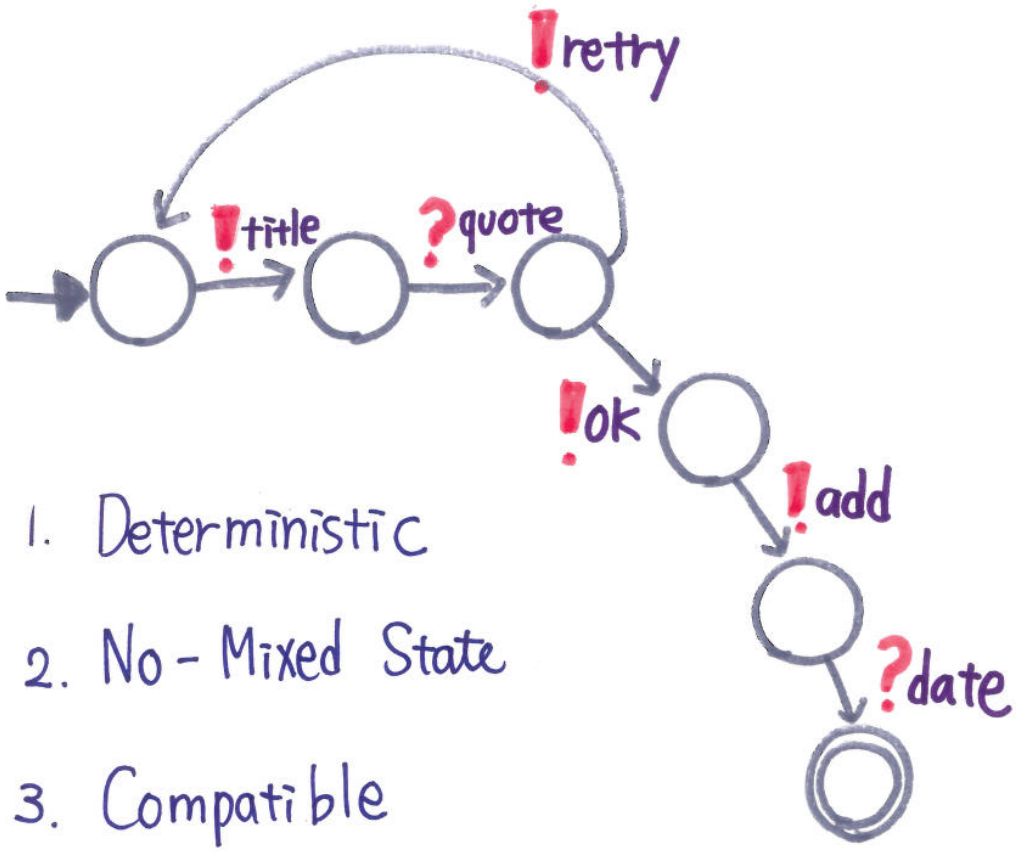
P | Q typable

$\mu t.!\ \text{Title}\ ;?\ \text{Quote}\ ;!\ \{\text{ok}:!\ \text{Add}\ ;?\ \text{Date}\ ,\ \text{retry}:t\}$

$\mu t.?\ \text{Title}\ ;!\ \text{Quote}\ ;?\ \{\text{ok}:?\ \text{Add}\ ;!\ \text{Date}\ ,\ \text{retry}:t\}$

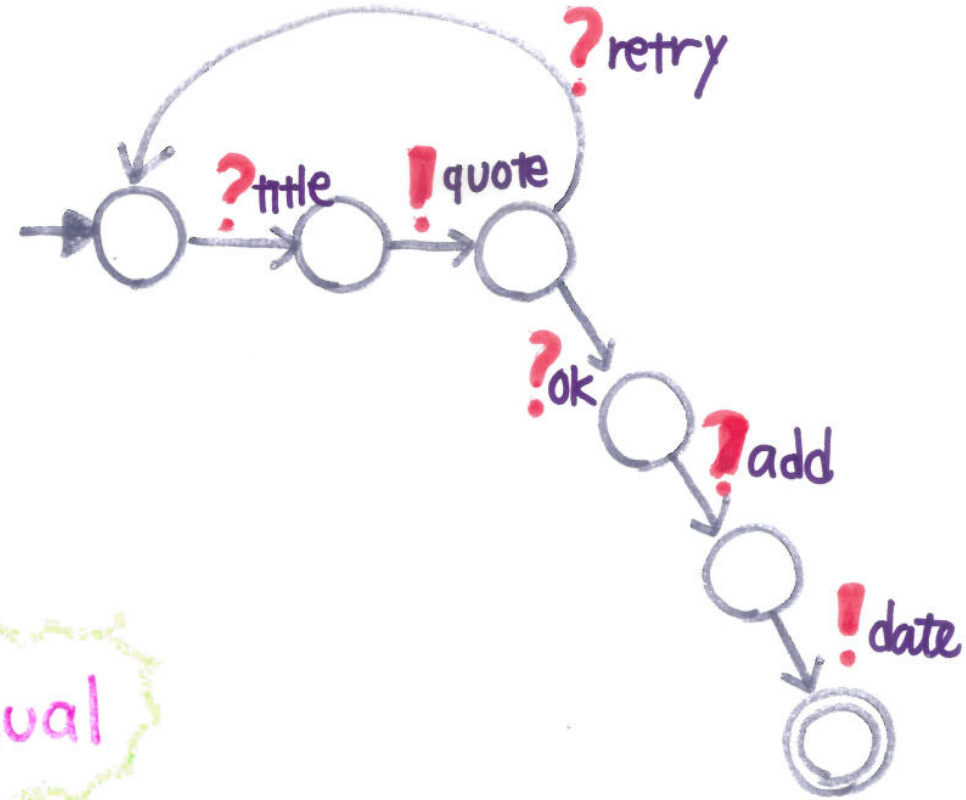
Communicating Automata [1980s]





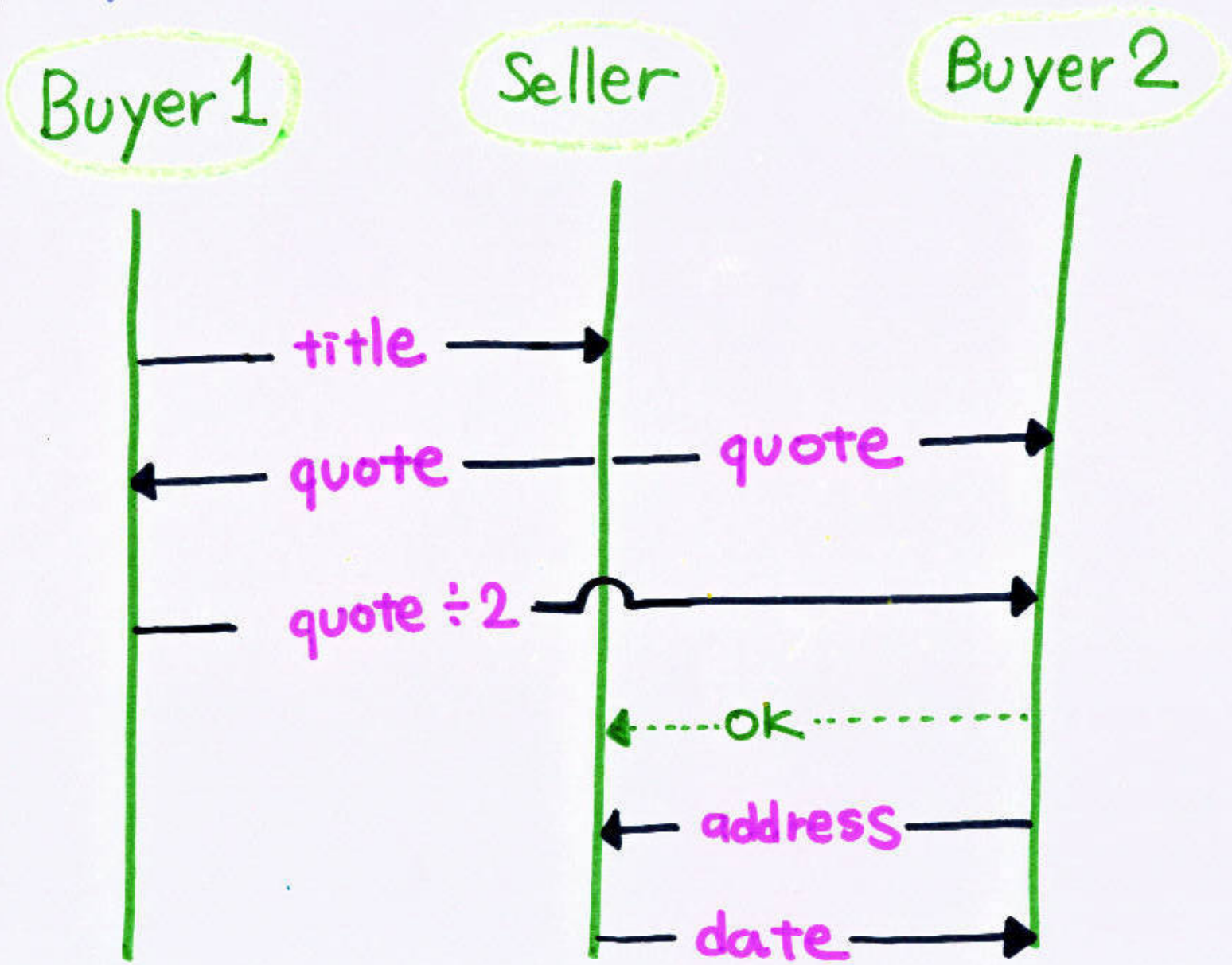
1. Deterministic
2. No-Mixed State
3. Compatible

dual

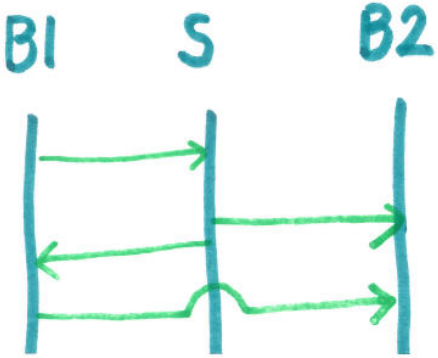


[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

Multiparty Session Types



Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

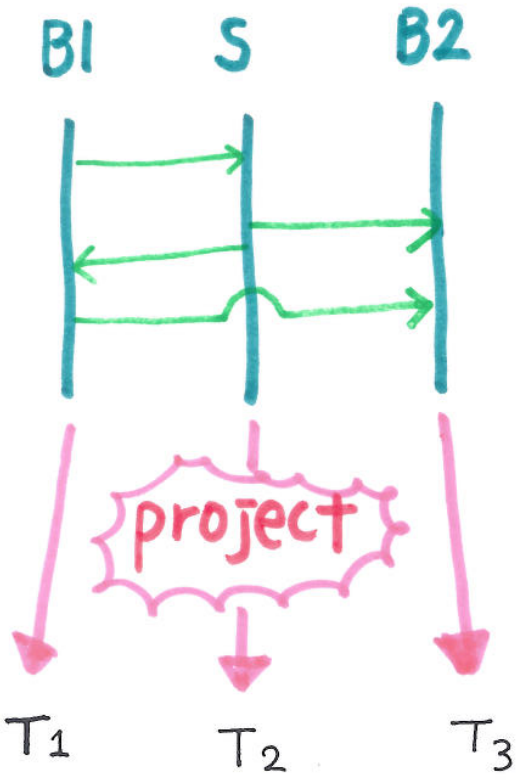
B1 → S Int.

S → B2 Char

STEP 1

Write Global Type

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

$B1 \rightarrow S$ Int.

$S \rightarrow B2$ Char

T

$B_1?$ Int. $B_2!$ Char

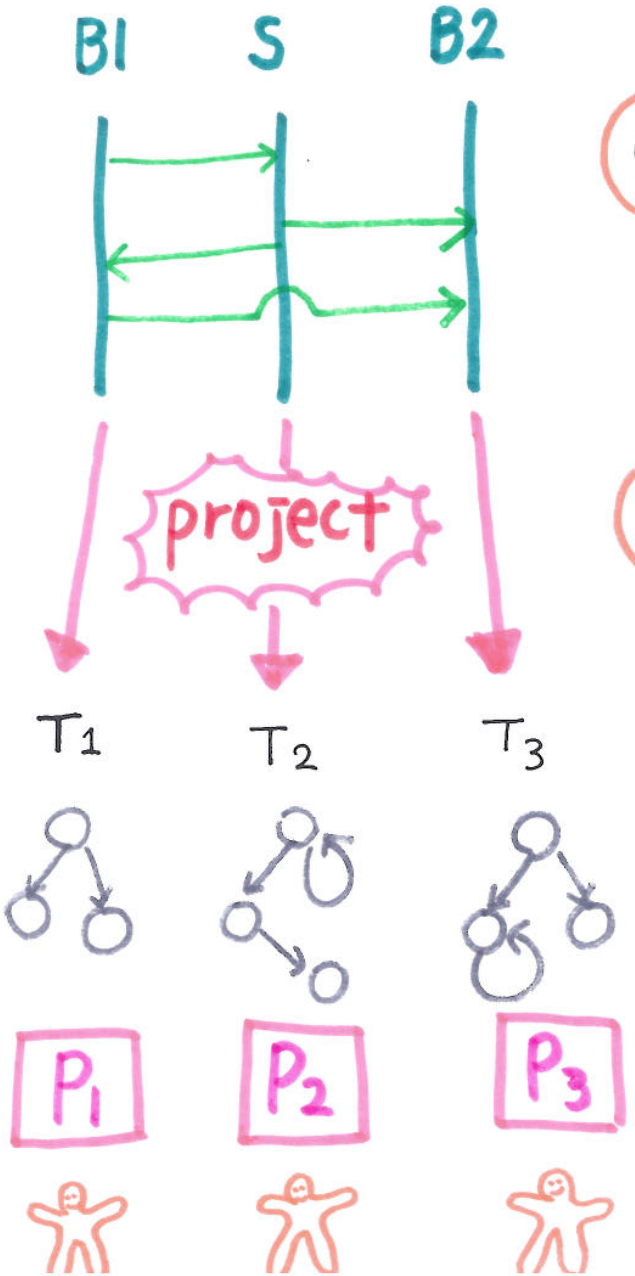
STEP 1

Write Global Type

STEP 2

Project to Local Types

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G $B1 \rightarrow S$ Int.
$S \rightarrow B2$ Char

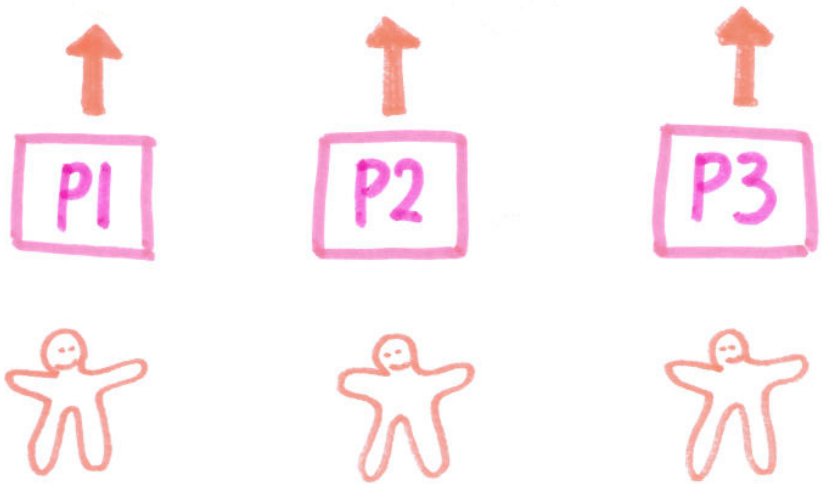
T $B_1?\text{Int}.\ B_2!\text{Char}$

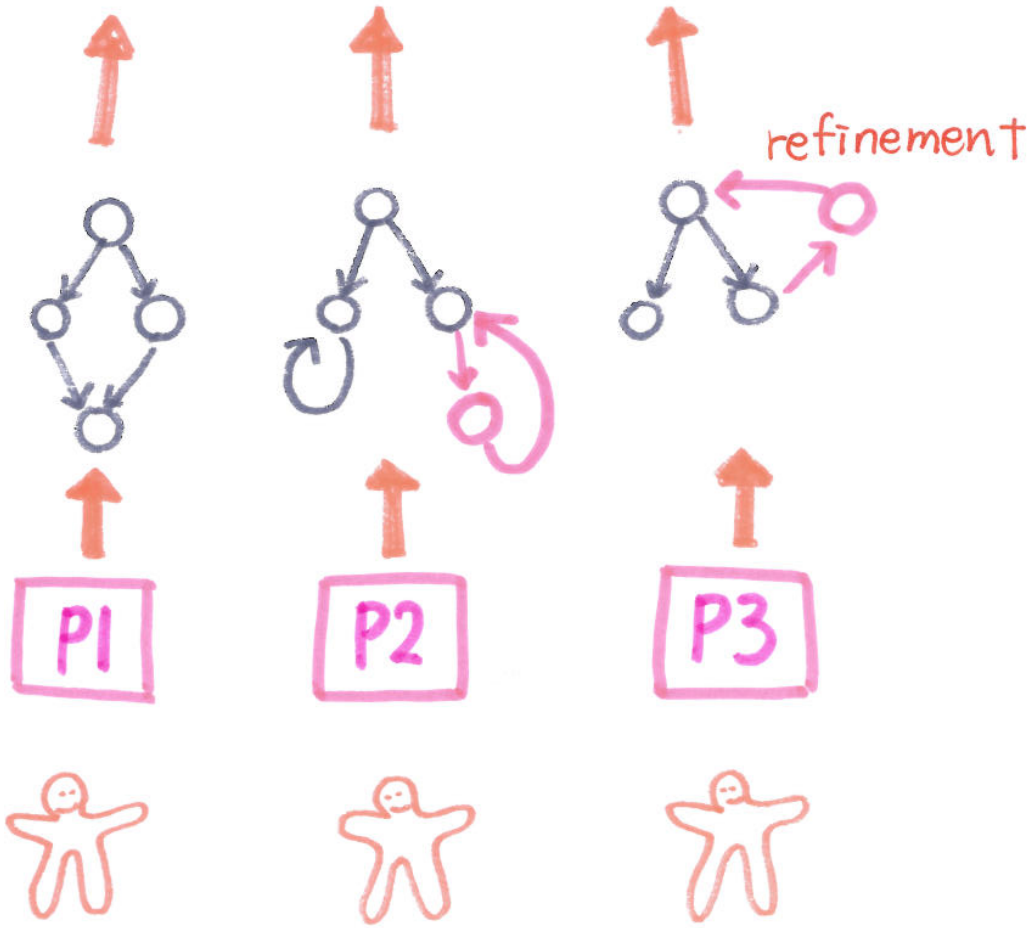
P $B_1?(x).\ B_2!\langle\text{"apple"}\rangle$

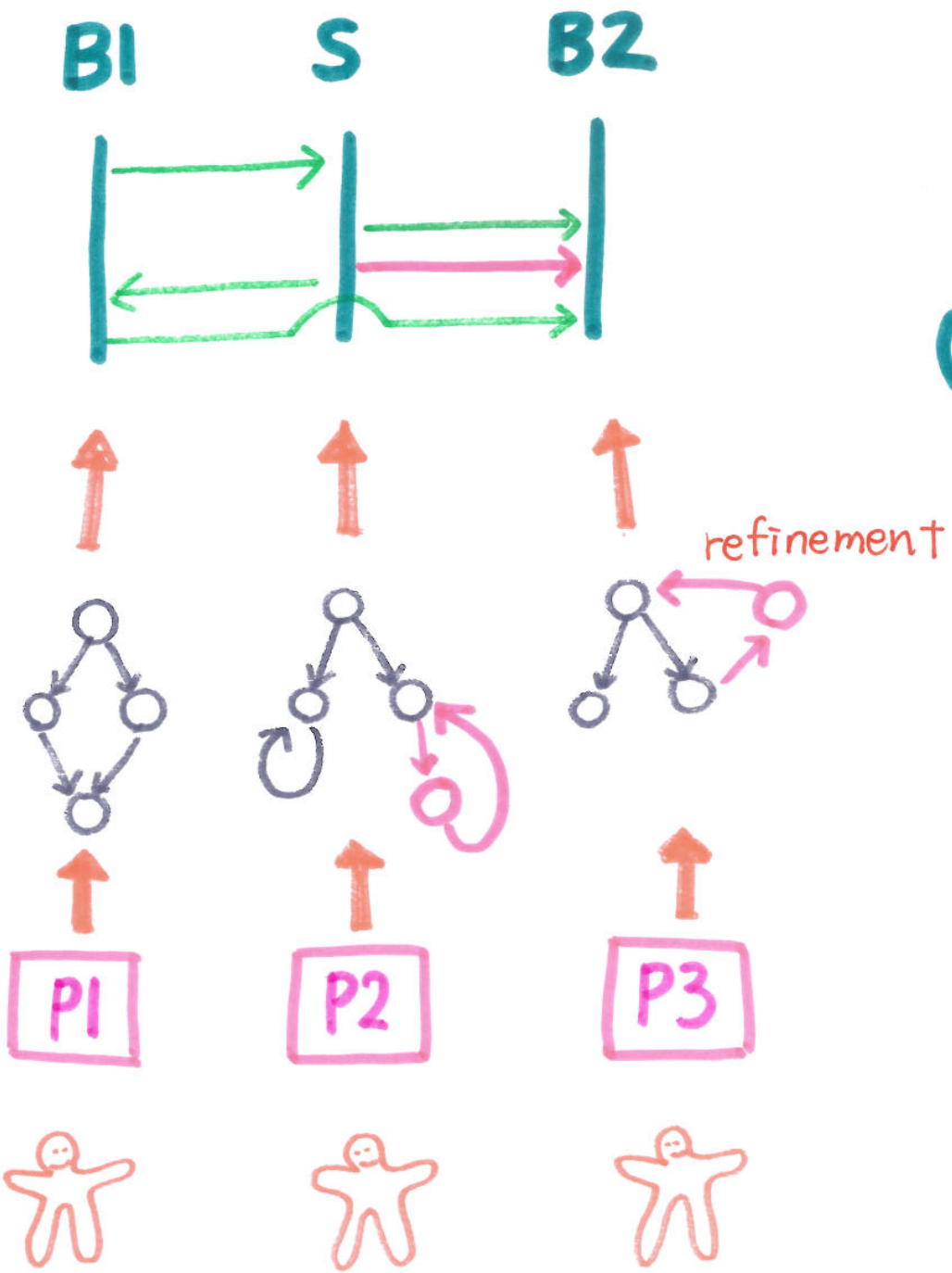
STEP 1
Write Global Type

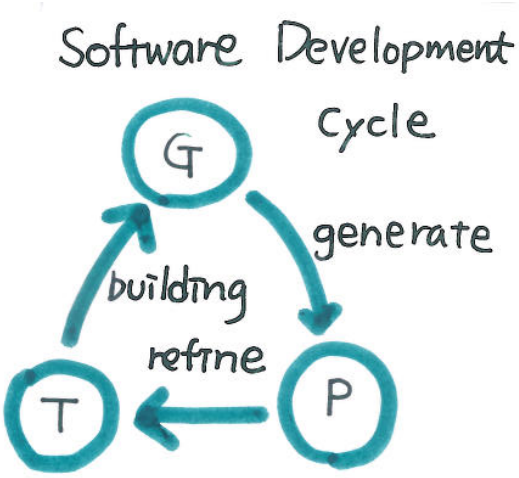
STEP 2
Project to Local Type

STEP 3
- Static Check
- Generate Code
- Run-time check


P1
P2
P3


refinement
P1
P2
P3





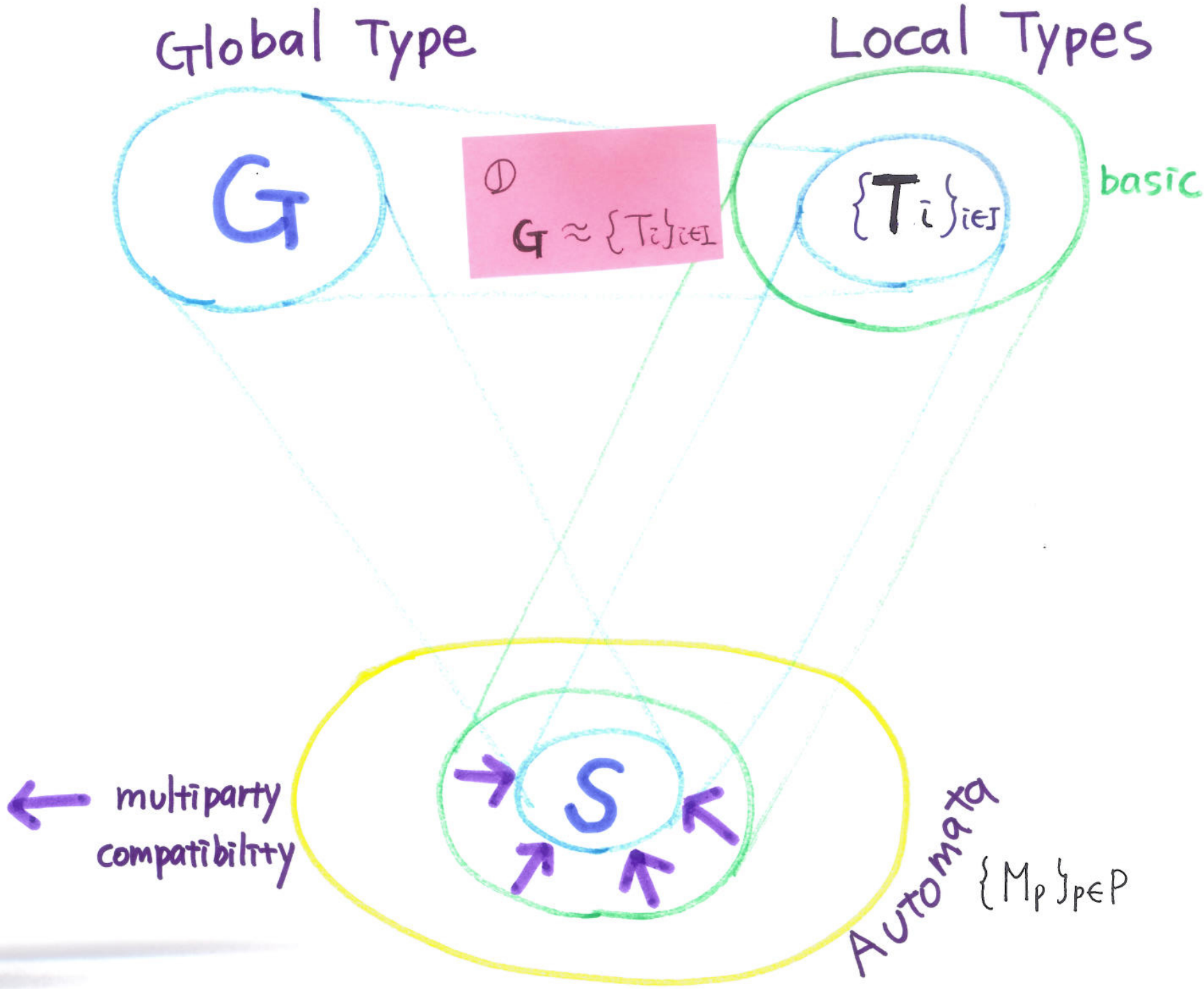
- Optimisation
- refinement
- Inference
- Testing

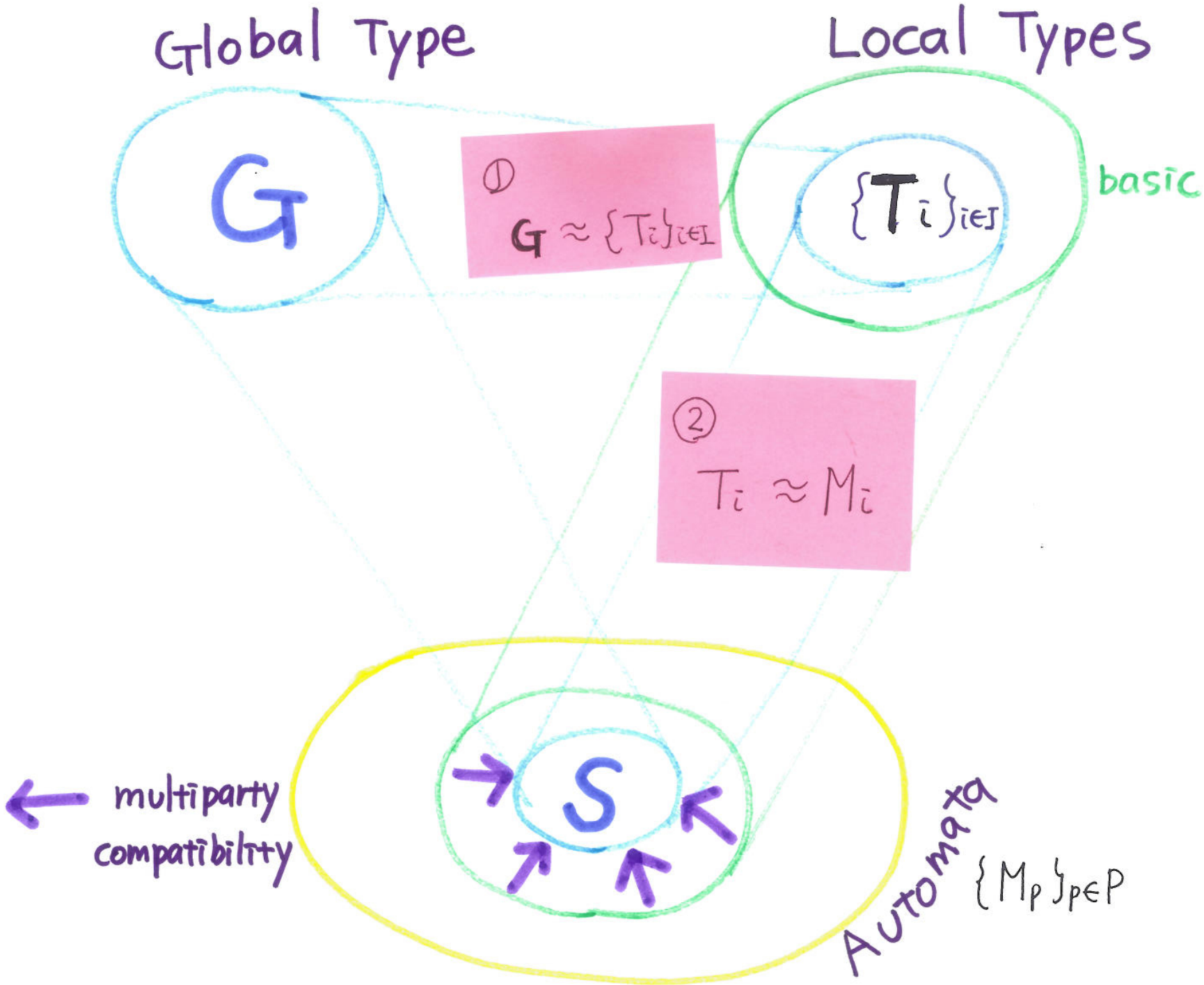
Aims

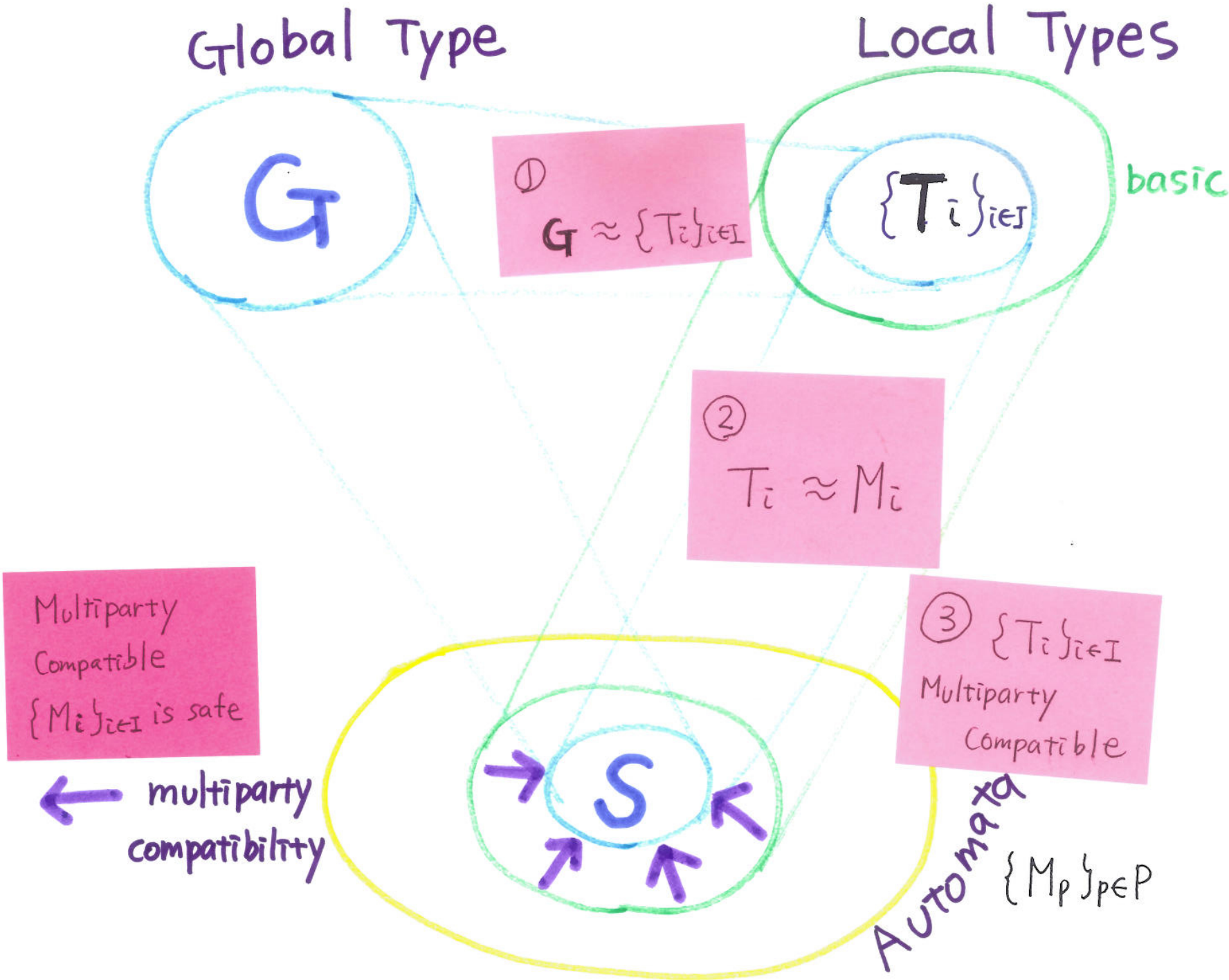
Building and Understanding

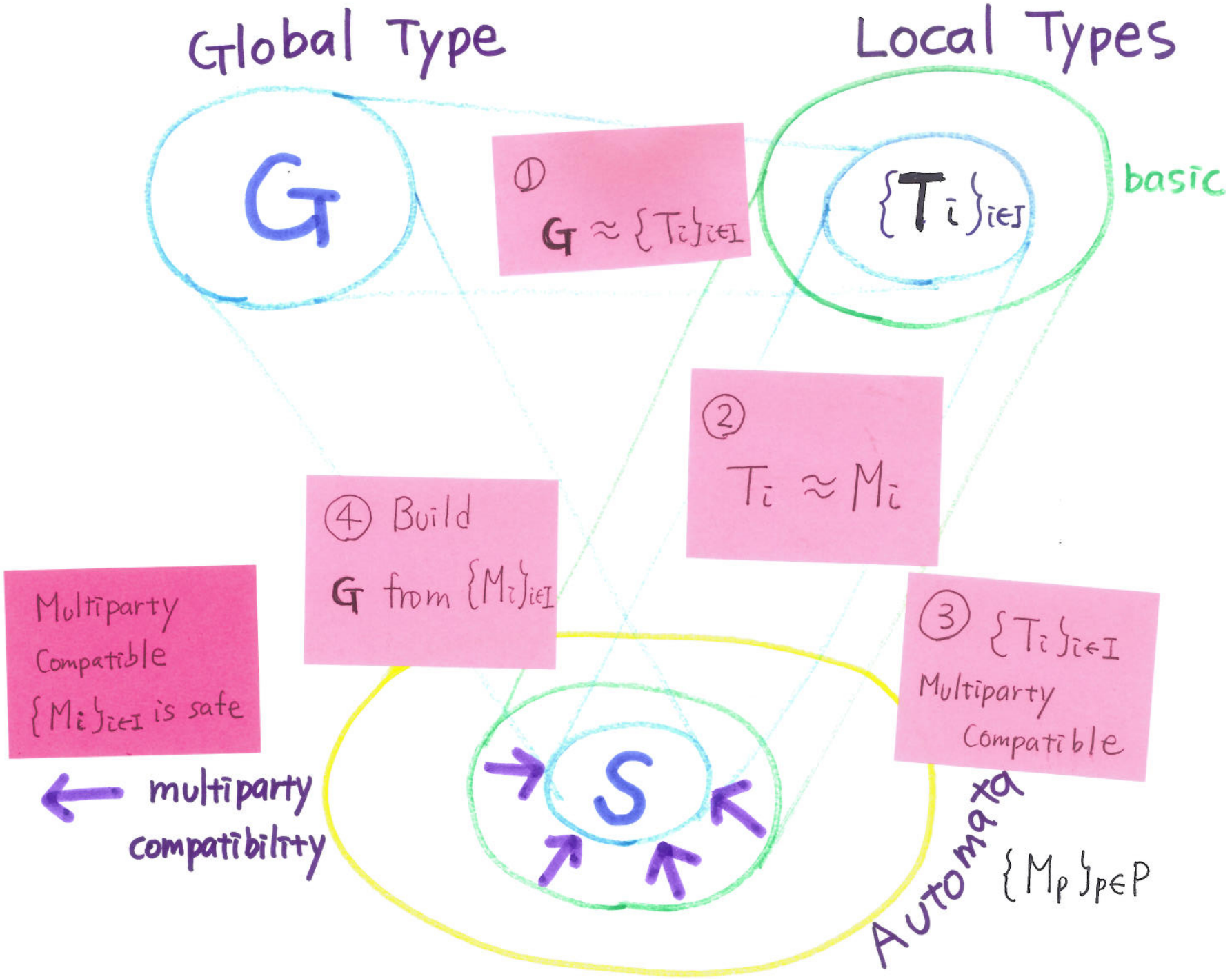
- Building Global Types from Communicating Automata (CA)

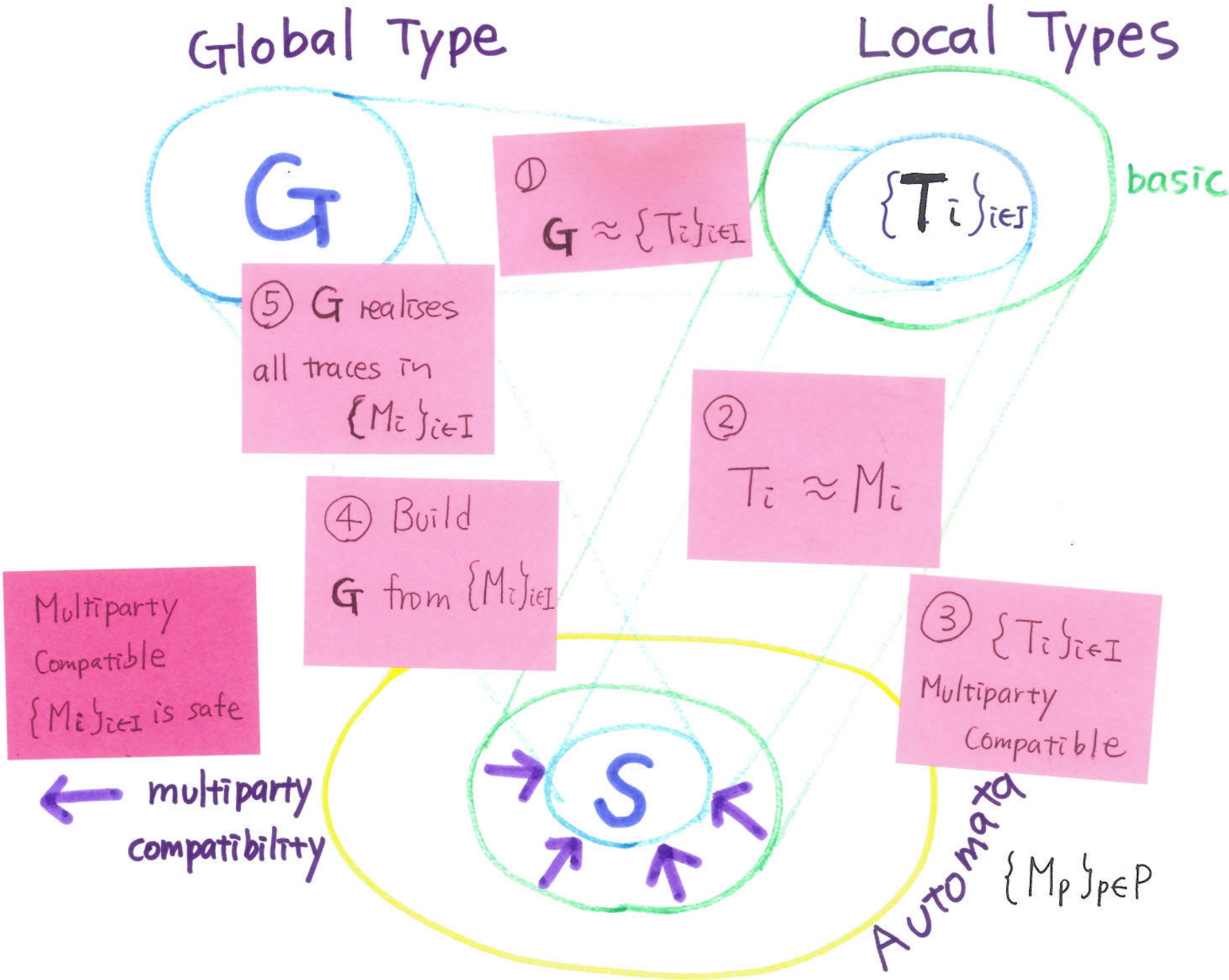
- Sound and Complete Characterisation of CA which can Build Global Types ⇨ Multiparty Compatibility


Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
S
multiparty compatibility
Automata
{Mp}p∈P


Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
② Ti ≈ Mi
S
multiparty compatibility
Automata
{Mp}p∈P


Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
② Ti ≈ Mi
Multiparty Compatible {Mi}i∈I is safe
multiparty compatibility
S
③ {Ti}i∈I Multiparty Compatible
Automata {Mp}p∈P


Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
② Ti ≈ Mi
④ Build G from {Mi}i∈I
Multiparty Compatible {Mi}i∈I is safe
③ {Ti}i∈I Multiparty Compatible
S
multiparty compatibility
Automata {Mp}p∈P


Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
⑤ G realises all traces in {Mi}i∈I
② Ti ≈ Mi
④ Build G from {Mi}i∈I
Multiparty Compatible {Mi}i∈I is safe
multiparty compatibility
S
③ {Ti}i∈I Multiparty Compatible
Automata
{Mp}p∈P

Global Types

$$G ::= p \to p' : \{ a_j . G_j \}_{j \in J}$$

$$\mid \quad \mu t . G$$

$$\mid \quad t$$

$$\mid \quad \text{end}$$

Local Types

$$T ::= p ! \{ a_j . T_j \}_{j \in J}$$

$$\mid \; p ? \{ a_j . T_j \}_{j \in J}$$

$$\mid \; \mu t . T$$

$$\mid \; t$$

$$\mid \; \text{end}$$

$p, p', \ldots$ Participant

a, b, c Σ Alphabet

Projection from G onto q $G\upharpoonright q$

$$\left(p \to p' : \{a_j . G_j\}_{j \in J}\right)\upharpoonright q$$

$$= \begin{cases} p\,!\{a_j . G_j\upharpoonright q\}_{j \in J} & q = p \\ p'\,?\{a_j . G_j\upharpoonright q\}_{j \in J} & q = p' \\ G_1\upharpoonright q = G_j\upharpoonright q & \forall j \in J \quad \text{otherwise} \end{cases}$$

Projection from G onto q $\quad G\upharpoonright q$

$$(p \to p' : \{a_j . G_j\}_{j \in J}) \upharpoonright q$$

$$= \begin{cases} p\,!\,\{a_j . G_j \upharpoonright q\}_{j \in J} & q = p \\ p\,?\,\{a_j . G_j \upharpoonright q\}_{j \in J} & q = p' \\ G_1 \upharpoonright q = G_j \upharpoonright q & \forall j \in J \quad \text{otherwise} \end{cases}$$

Bad $\quad A \to B : \{a . C \to D : c \;,\; b . C \to D : d\}$

Good $\quad A \to B : \{a . C \to D : c \;,\; b . C \to D : c\}$

LTS Local Types $\ell ::= pq!a \mid pq?a$

$$q!\{a_i.T_i\}_{i\in I} \xrightarrow{pq!a_i} T_i \qquad q?\{a_i.T_i\}_{i\in I} \xrightarrow{qp?a_i} T_i$$

$$\frac{T[\mu t.T/t] \xrightarrow{\ell} T'}{\mu t.T \xrightarrow{\ell} T'}$$

$(\vec{T};\vec{w})$ w_{pq} queue

Send $T_p \xrightarrow{pq!\ell} T_p' \Rightarrow$

$$(\ldots T_p \ldots ; \ldots w_{pq} \ldots) \xrightarrow{pq!\ell} (\ldots T_p' \ldots ; \ldots w_{pq}\cdot\ell \ldots)$$

Receive $T_q \xrightarrow{pq?\ell} T_q' \Rightarrow$

$$(\ldots T_q \ldots ; \ldots \ell\cdot w_{pq} \ldots) \xrightarrow{pq?\ell} (\ldots T_q' \ldots ; \ldots w_{pq} \ldots)$$

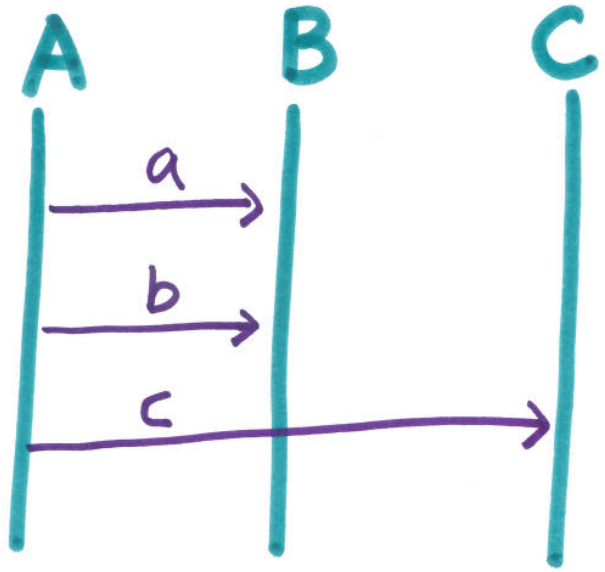
LTS Global Types

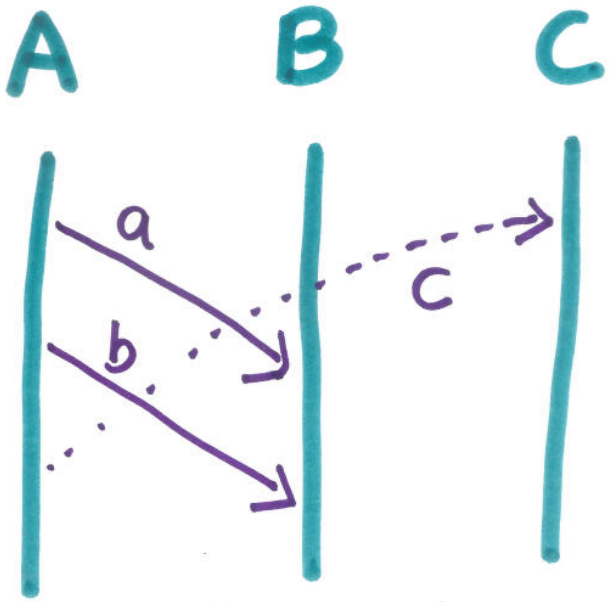
$p \rightsquigarrow p' : j \{a_i . G_i\}_{i \in I}$

put $\quad p \to p' : \{a_i . G_i\}_{i \in I} \xrightarrow{pp'!a_j} p \rightsquigarrow p' : j \{a_i . G_i\}_{i \in I}$

get $\quad p \rightsquigarrow p' : j \{a_i . G_i\}_{i \in I} \xrightarrow{pp'?a_j} G_j$

$$\frac{\forall j \in I \quad G_j \xrightarrow{\ell} G_j' \qquad p, q \notin sub(\ell)}{p \to p' : \{a_i . G_i\}_{i \in I} \xrightarrow{\ell} p \to p' : \{a_i . G_i'\}_{i \in I}} \qquad \frac{G_j \xrightarrow{\ell} G_j' \qquad q \notin sub(\ell) \quad \forall i \in I \setminus j \;\; G_i' = G_i}{p \rightsquigarrow q : j \{a_i . G_i\}_{i \in I} \xrightarrow{\ell} p \rightsquigarrow q : j \{a_i . G_i'\}_{i \in I}}$$





$A \to B:a \,.\, A \to B:b \,.\, A \to C:c$

↓ AB!a OUT

$A \rightsquigarrow B:a \,.\, A \to B:b \,.\, A \to C:c$

↓ AB?a IN

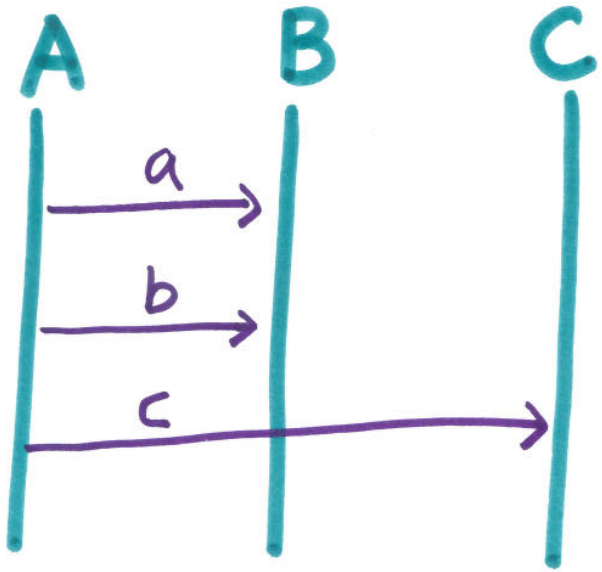
$A \to B:b \;\; A \to C:c$

↓ AB!b OUT

$A \rightsquigarrow B:b \;\; A \to C:c$

↓ AB?b IN

$A \to C:c$



$A \to B:a.\ A \to B:b,\ A \to C:c$

↓ AB!a OUT

$A \rightsquigarrow B:a.\ A \to B:b.\ A \to C:c$

↓ AB?a IN

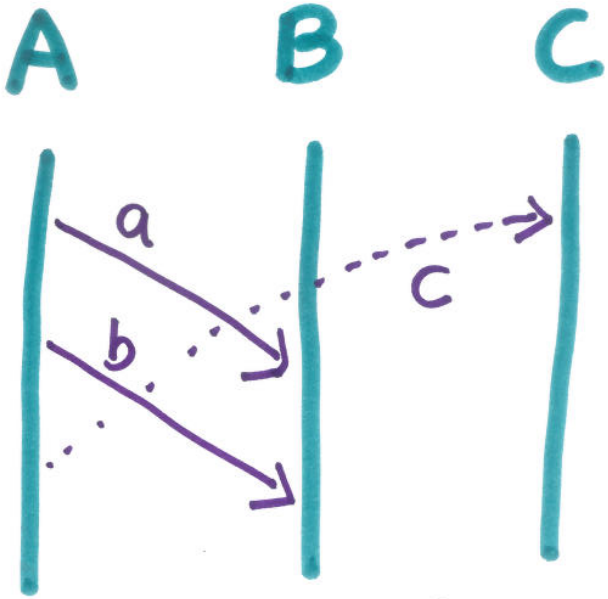
$A \to B:b\ \ A \to C:c$

↓ AB!b OUT

$A \rightsquigarrow B:b\ \ A \to C:c$

↓ AB?b IN

$A \to C:c$



$A \to B:a.\ A \to B:b.\ A \to C:c$

↓ AB!a OUT

↓ AB!b OUT

↓ AC!c OUT

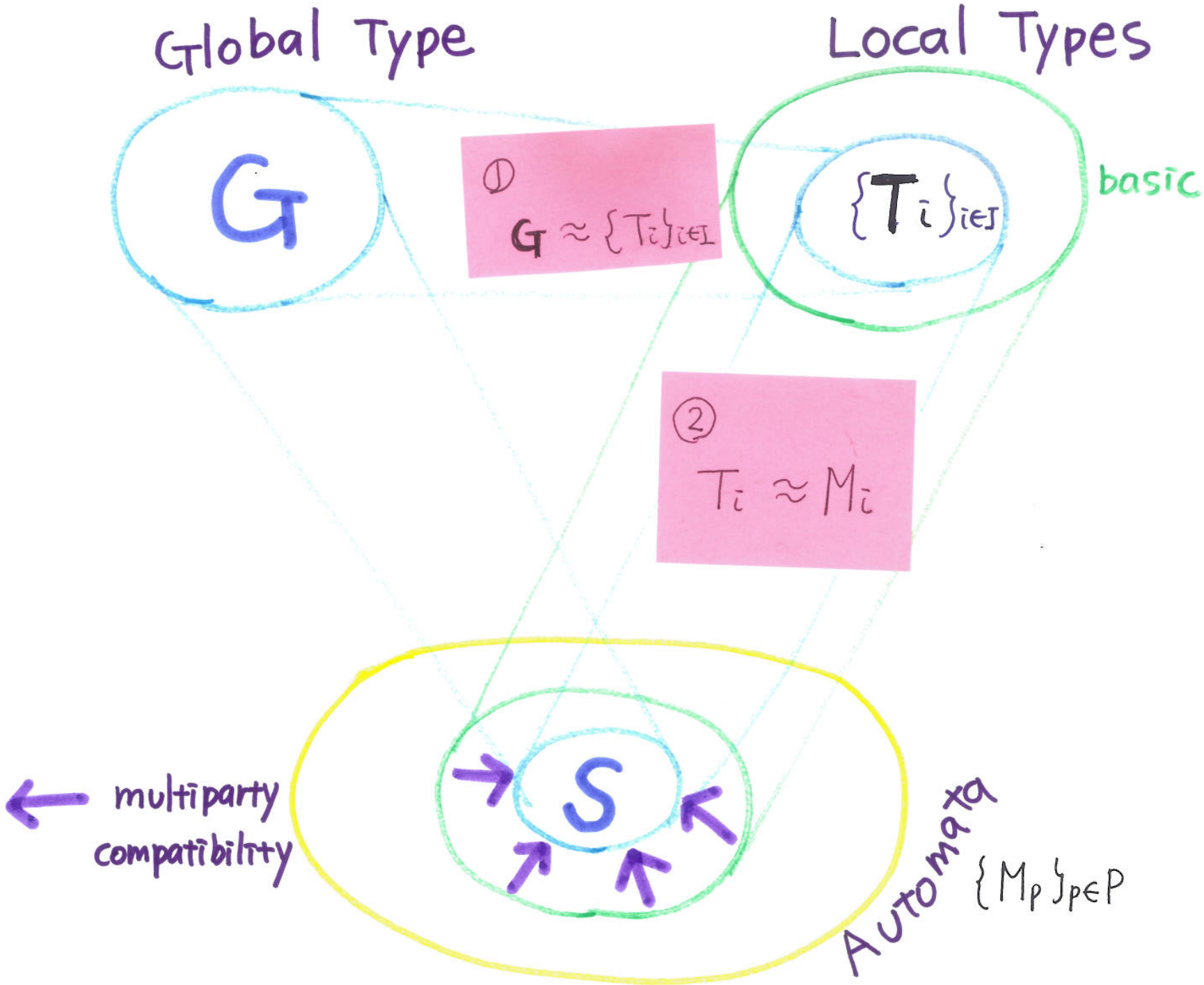
$A \rightsquigarrow B:a.\ A \rightsquigarrow B:b.\ A \rightsquigarrow C:c$

↓ AC?c IN

$A \rightsquigarrow B:a.\ A \rightsquigarrow B:b$

↓ AB?a IN

$A \rightsquigarrow B:b$


Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
② Ti ≈ Mi
S
multiparty compatibility
Automata
{Mp}p∈P

CFSMs [1980 -] ITU notation SDL · MSCS ···

Def A CFSM $M = (Q, C, q0, \Sigma, \delta)$

Q a finite set of states

$C = \{ pq \in Participant^2 \mid p \neq q \}$

$q0$ initial state

Σ a finite alphabet of messages

$\delta \subseteq Q \times (C \times \{!, ?\} \times \Sigma) \times Q$ a finite set of transitions

Def CS $S = (M_p)_{p \in Participant}$

$S \xrightarrow{t} S'$ configuration $s = (\vec{q}\,;\vec{w})$

states q_p queues w_{pq}

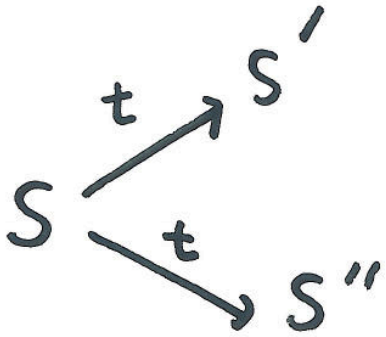
Send

$(\ldots q_p \ldots\,;\ldots w_{pq} \ldots) \xrightarrow{pq!\ell} (\ldots q_p' \ldots\,;\ldots w_{pq} \cdot \ell \ldots)$

Receive

$(\ldots q_q \ldots\,;\ldots \ell \cdot w_{pq} \ldots) \xrightarrow{pq?\ell} (\ldots q_q' \ldots\,;\ldots w_{pq} \ldots)$

Deteministic CFSM

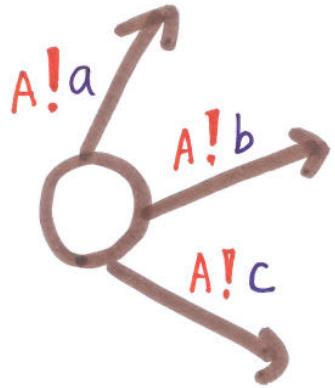


$\Rightarrow \quad S' = S''$

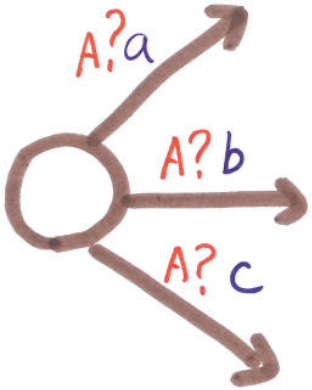
Basic CFSMs

A CFSM is **Basic** if **deterministic**

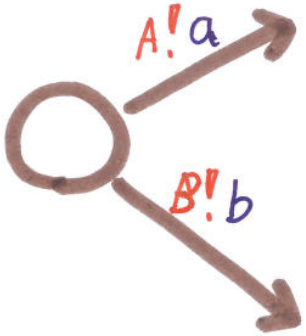
directed, has **no mixed states**



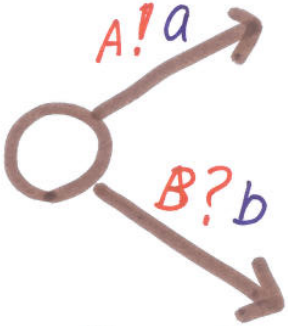
sending



receiving



non directed



mixed

✓ ✓ ✗ ✗

T = A!{a, b, c} A?{a, b, c}

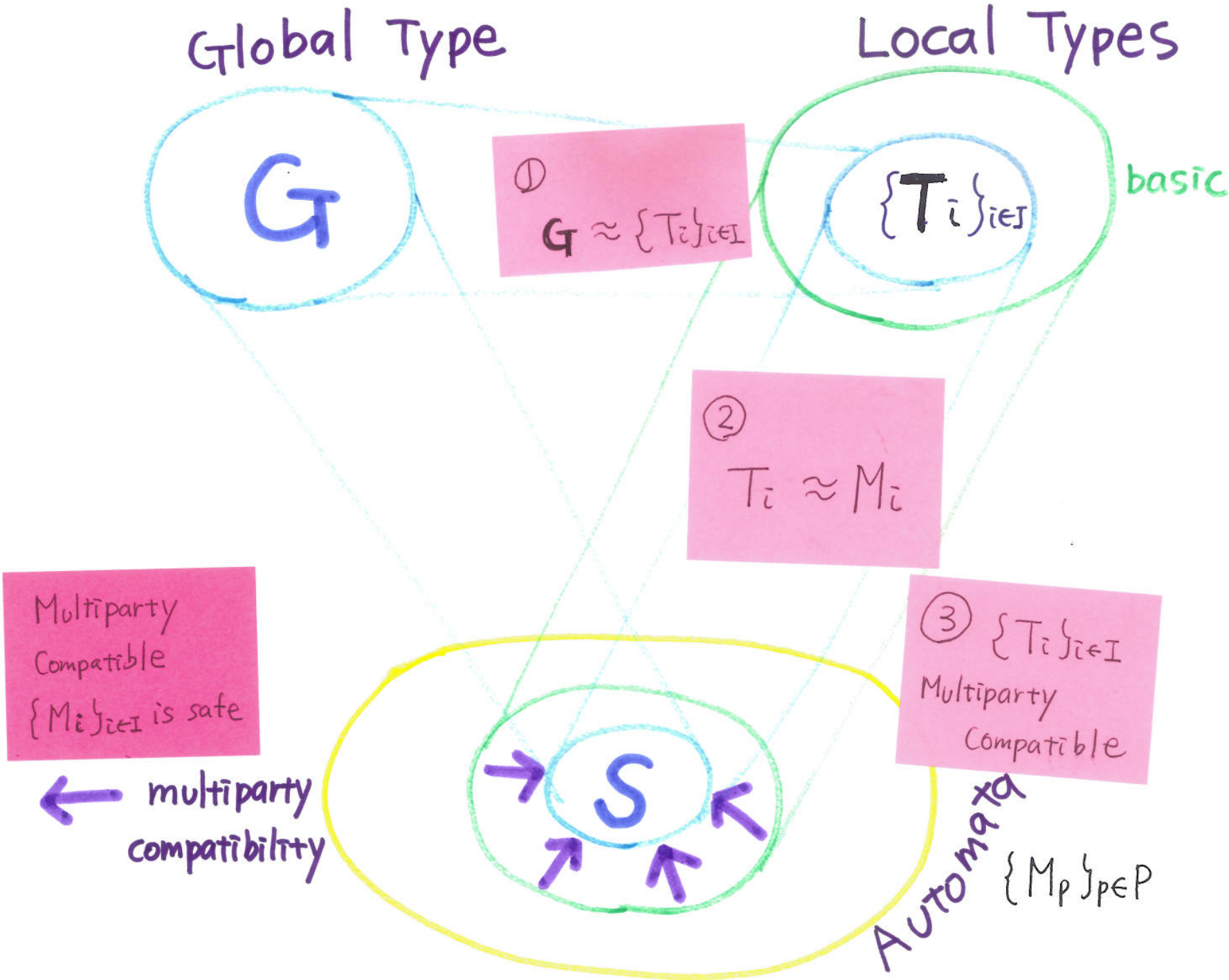
Theorem 1 $Tr(G) \approx Tr(\{G\lceil p\}_{p \in Participant})$

Soundness and Completeness between a global type and projected local types

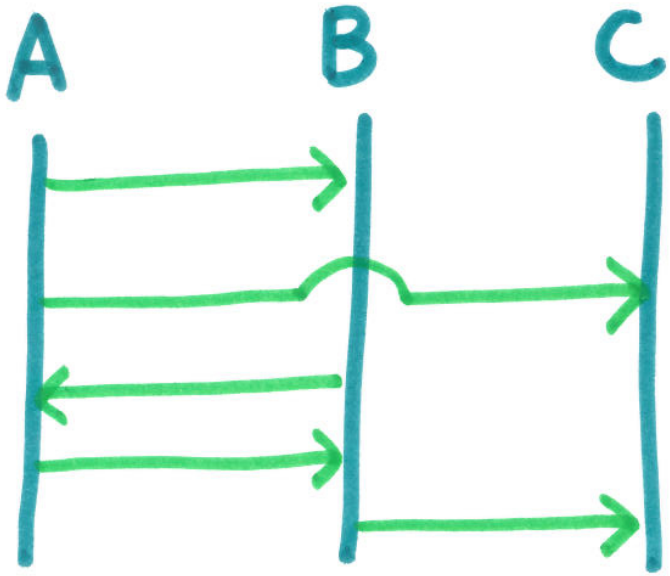
Theorem 2 $Tr(T) \approx Tr(A(T))$ T basic

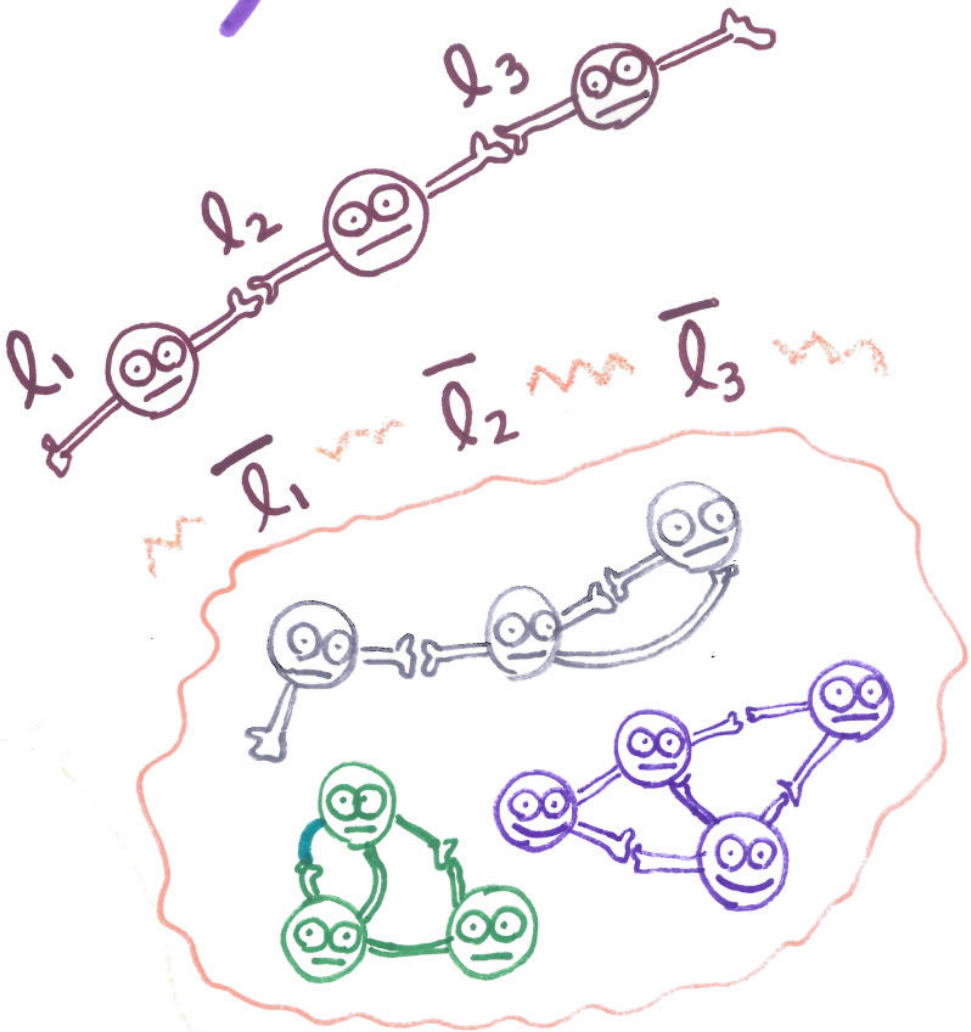
Soundness and Completeness between a local type and a basic CFSM of T

T

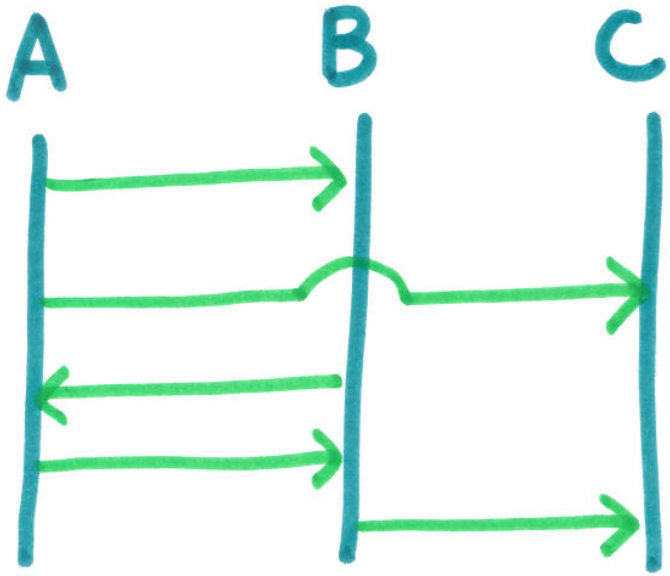

Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
② Ti ≈ Mi
Multiparty Compatible {Mi}i∈I is safe
multiparty compatibility
S
③ {Ti}i∈I Multiparty Compatible
Automata {Mp}p∈P

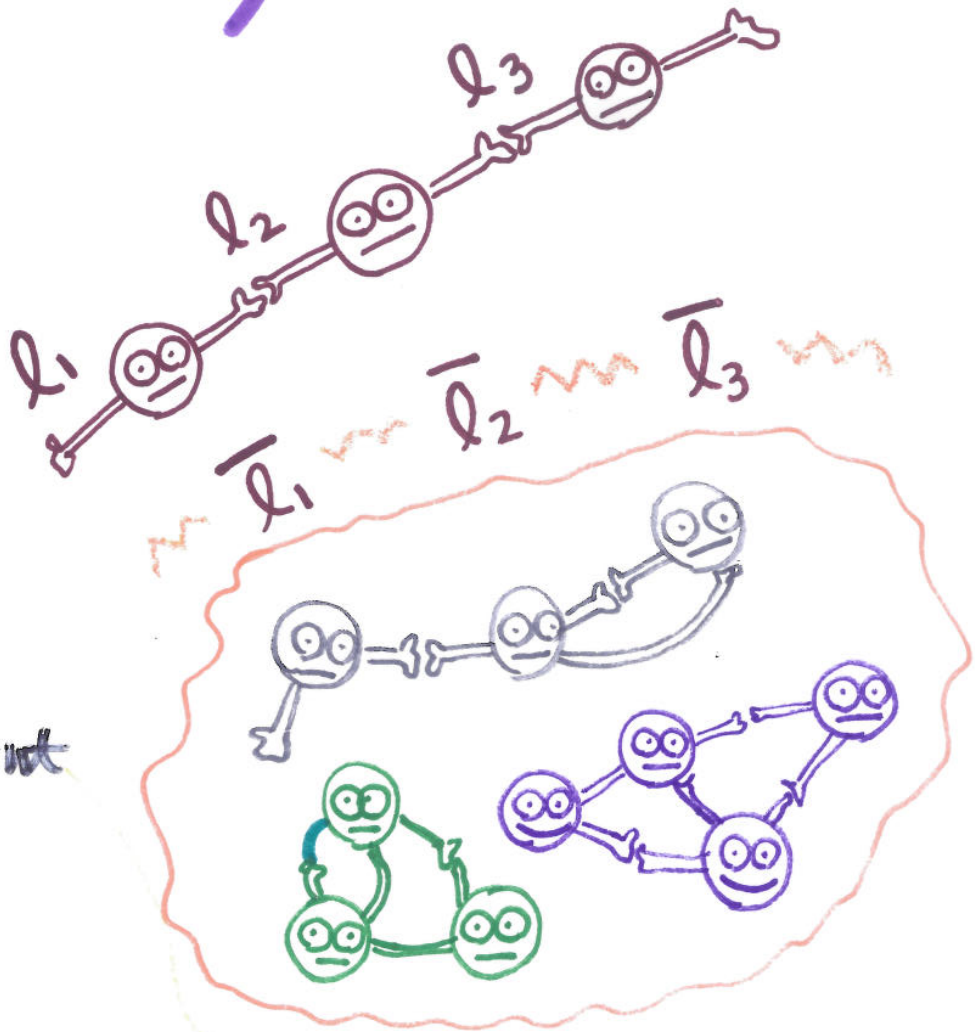
Multiparty Compatibility





Multiparty Compatibility





Def $S = (M_p)_{p \in Participant}$

$\forall s \,.\, s_0 \overset{\text{1-buffer execution}}{\rightsquigarrow} s$

if M_i does action ℓ

then $(M_j)_{j \in P \setminus i}$ do action $\bar{\ell}$

after some $\rightsquigarrow$

Theorem (Safety) Suppose S is basic and multiparty compatible. Then S is free from the following errors. Let $S = (\vec{q}; \vec{w})$

Deadlock if $\vec{w} = \vec{\varepsilon}$ and not final, each q_p is receiving

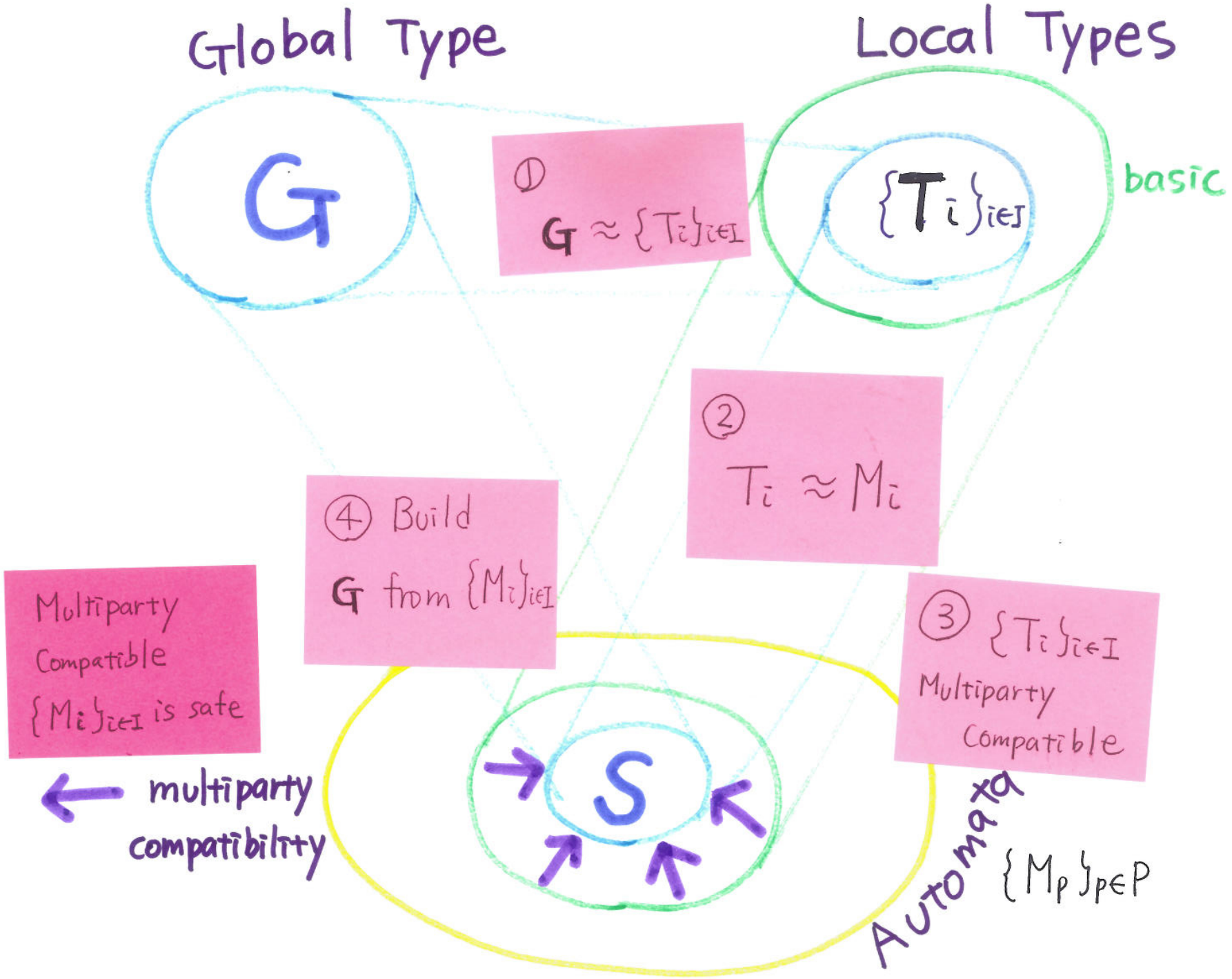
Orphan Message all q are final, but $\vec{w} \neq \vec{\varepsilon}$

Unspecified Reception $(q_p, p_q?\underline{a}, q'_p)$ implies $|w_{pq}| > 1$ and $w_{pq} \notin \underline{a} \cdot \Sigma^*$

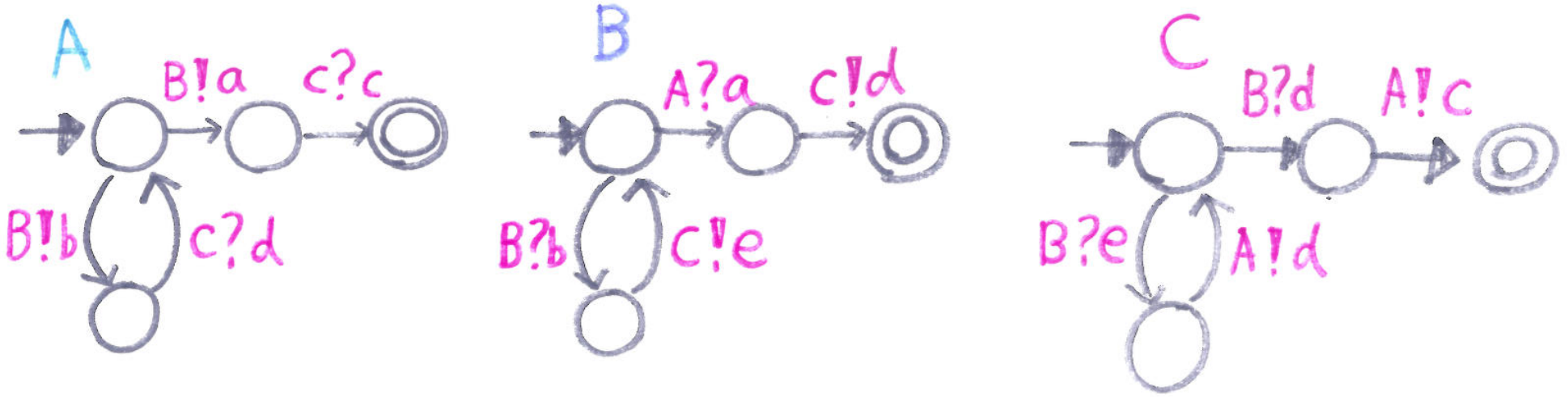
Proposition Soundness $A(\{G\upharpoonright p\}_{p \in \text{Participant}})$ is multiparty compatible.

projection

Proposition Multiparty Compatibility is decidable.


Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
② Ti ≈ Mi
④ Build G from {Mi}i∈I
Multiparty Compatible {Mi}i∈I is safe
③ {Ti}i∈I Multiparty Compatible
S
multiparty compatibility
Automata {Mp}p∈P

Synthesis

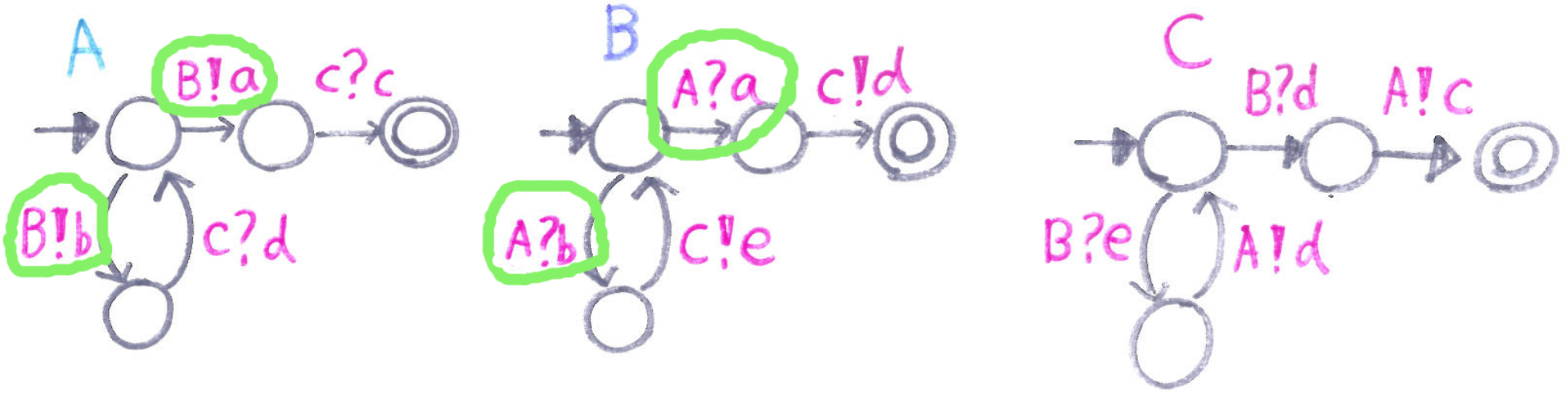


$$\mu t.\ \mathbf{A}\to\mathbf{B}:\{a.\ \mathbf{B}\to\mathbf{C}:\{d.\ \mathbf{C}\to\mathbf{A}:\{c:\mathrm{end}\}\}$$
$$b.\ \mathbf{B}\to\mathbf{C}:\{e.\ \mathbf{C}\to\mathbf{A}:\{d:t\}\}\}$$

Theorem Suppose S is basic and compatible.

Then there is an algorithm to build a well-formed G.

Synthesis

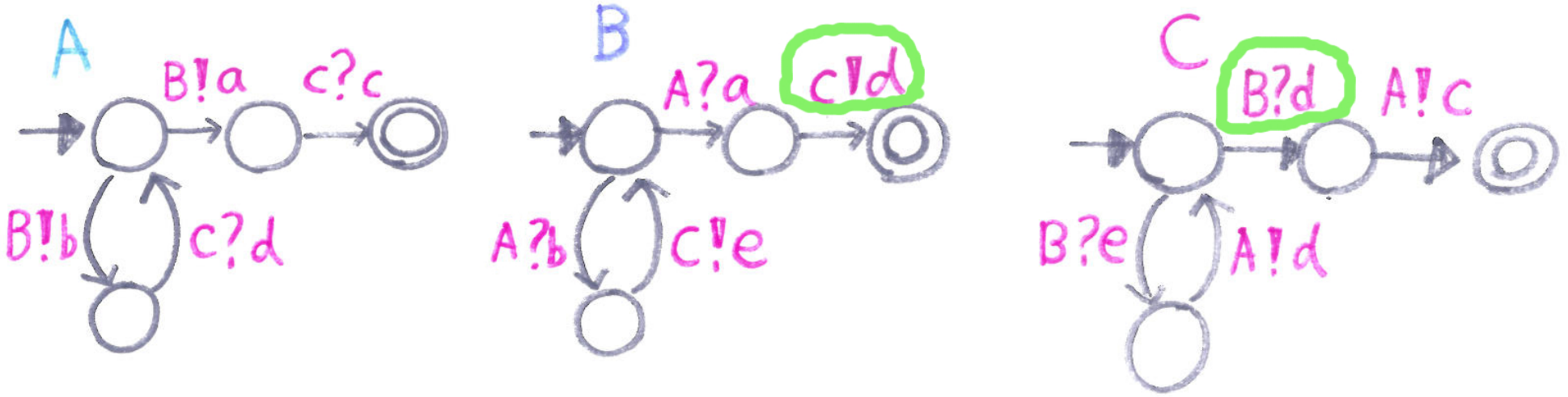


$$\mu t.\ A \to B : \{ a.\ B \to C : \{ d.\ C \to A : \{ c : \text{end} \} \},\ b.\ B \to C : \{ e.\ C \to A : \{ d : t \} \} \}$$

Theorem Suppose S is basic and compatible.

Then there is an algorithm to build a well-formed G.

Synthesis

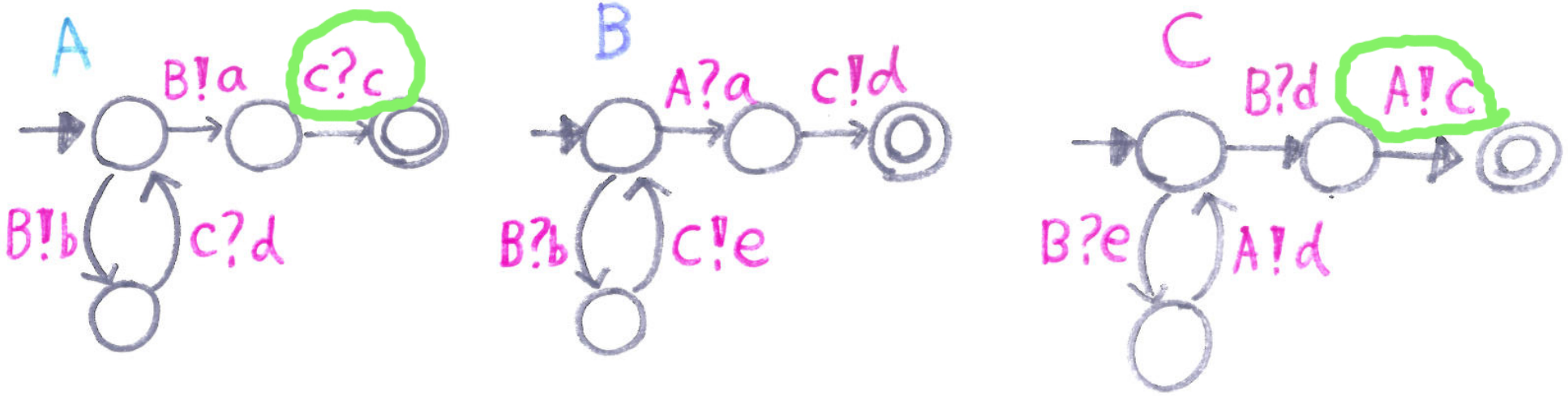


$$\mu t.\ A \to B : \{ a.\ B \to C : \{ d.\ C \to A : \{ c : \mathrm{end} \} \},\ b.\ B \to C : \{ e.\ C \to A : \{ d : t \} \} \}$$

Theorem Suppose S is basic and compatible.

Then there is an algorithm to build a well-formed G.

Synthesis

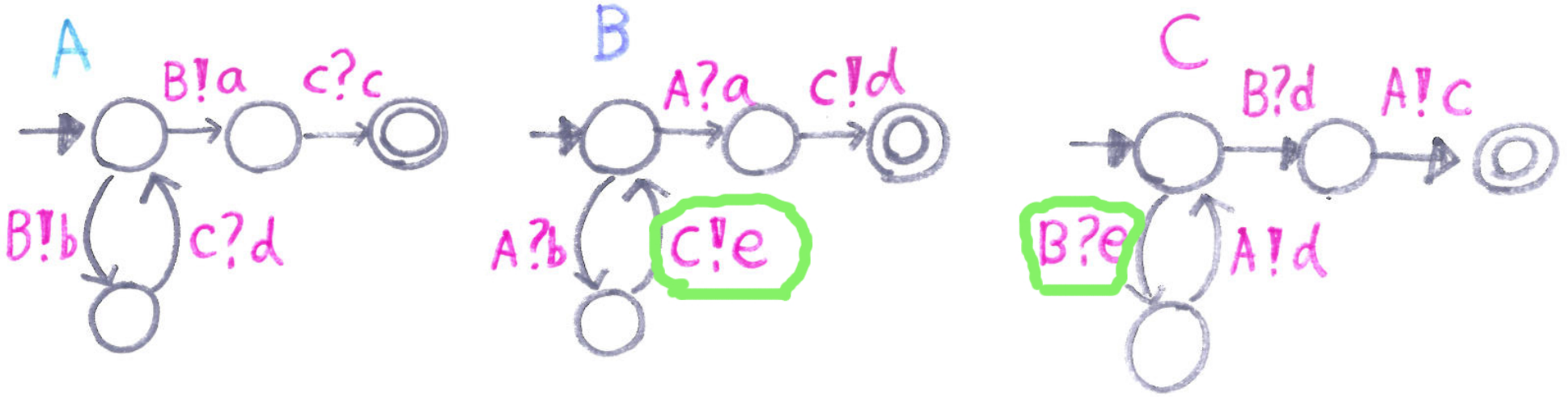


$$\mu t.\ A \to B : \{ a.\ B \to C : \{ d.\ C \to A : \{ c : \mathrm{end} \} \} \}$$
$$b.\ B \to C : \{ e.\ C \to A : \{ d : t \} \} \}$$

Theorem Suppose S is basic and compatible.

Then there is an algorithm to build a well-formed G.

Synthesis

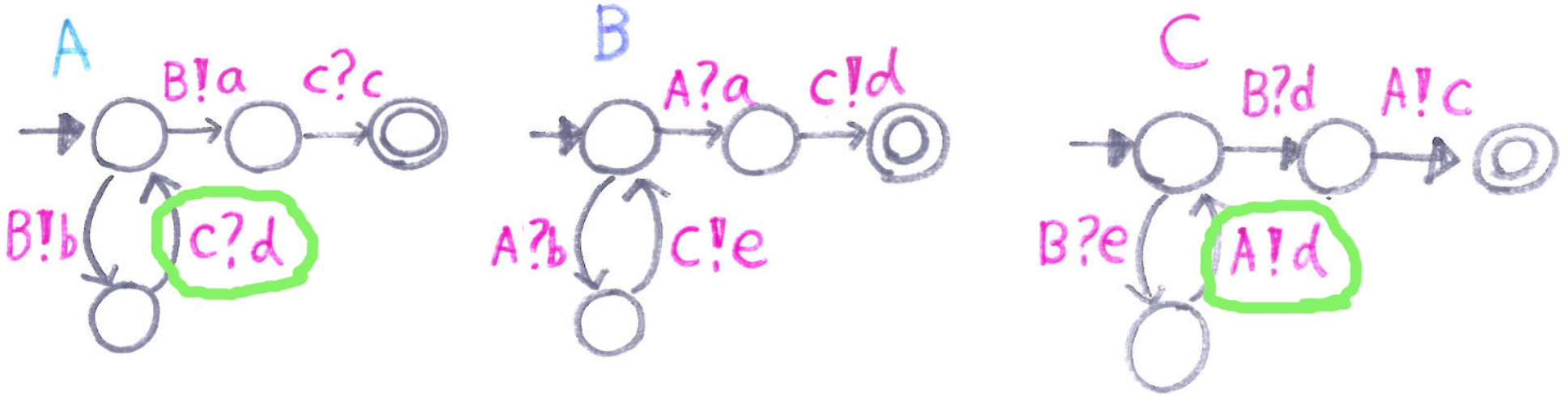


$$\mu t.\ A \to B : \{ a.\ B \to C : \{ d.\ C \to A : \{ c : \text{end} \} \} \}$$
$$b.\ B \to C : \{ e.\ C \to A : \{ d : t \} \} \}$$

Theorem Suppose S is basic and compatible.

Then there is an algorithm to build a well-formed G.

Synthesis

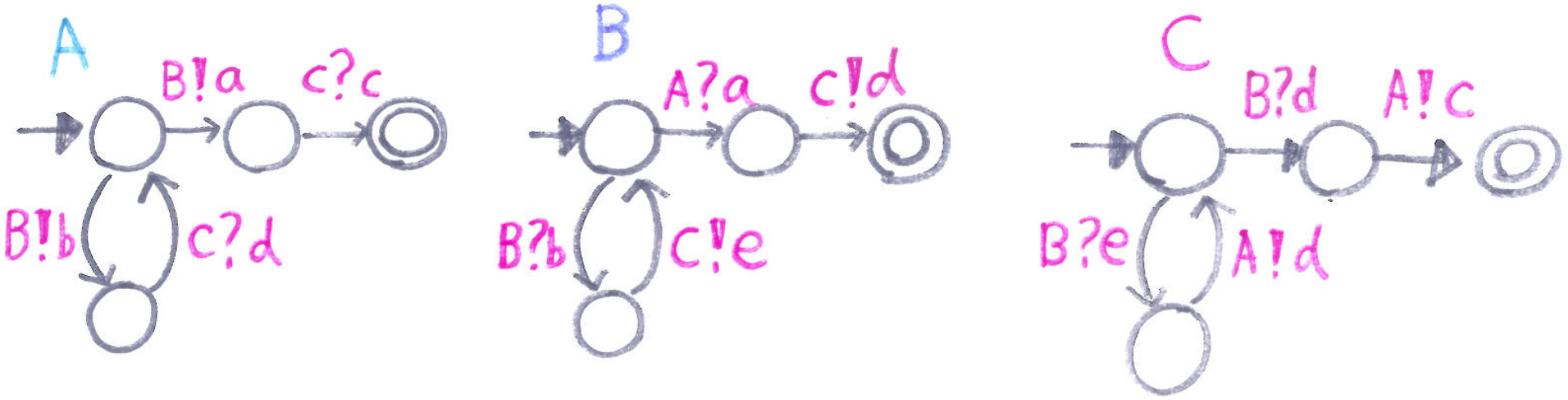


$$\mu t.\ A \to B : \{ a.\ B \to C : \{ d.\ C \to A : \{ c : \text{end} \} \} \}$$
$$b.\ B \to C : \{ e.\ C \to A : \{ d : t \} \} \}$$

Theorem Suppose S is basic and compatible.

Then there is an algorithm to build a well-formed G.

Synthesis



$$\mu t.\ A \to B : \{ a.\ B \to C : \{ d.\ C \to A : \{ c : end \} \} \}$$
$$b.\ B \to C : \{ e.\ C \to A : \{ d : t \} \} \}$$

Theorem Problem! S is bo… …npatible.

1-bound $Tr(G) \approx_1 Tr(S)$

Then there is an algorithm … well-formed G.

Lemma If S is basic and multiparty compatible, then $Tr(S_1) \approx_1 Tr(S_2)$ implies $Tr(S_1) \approx Tr(S_2)$

Theorem Completeness

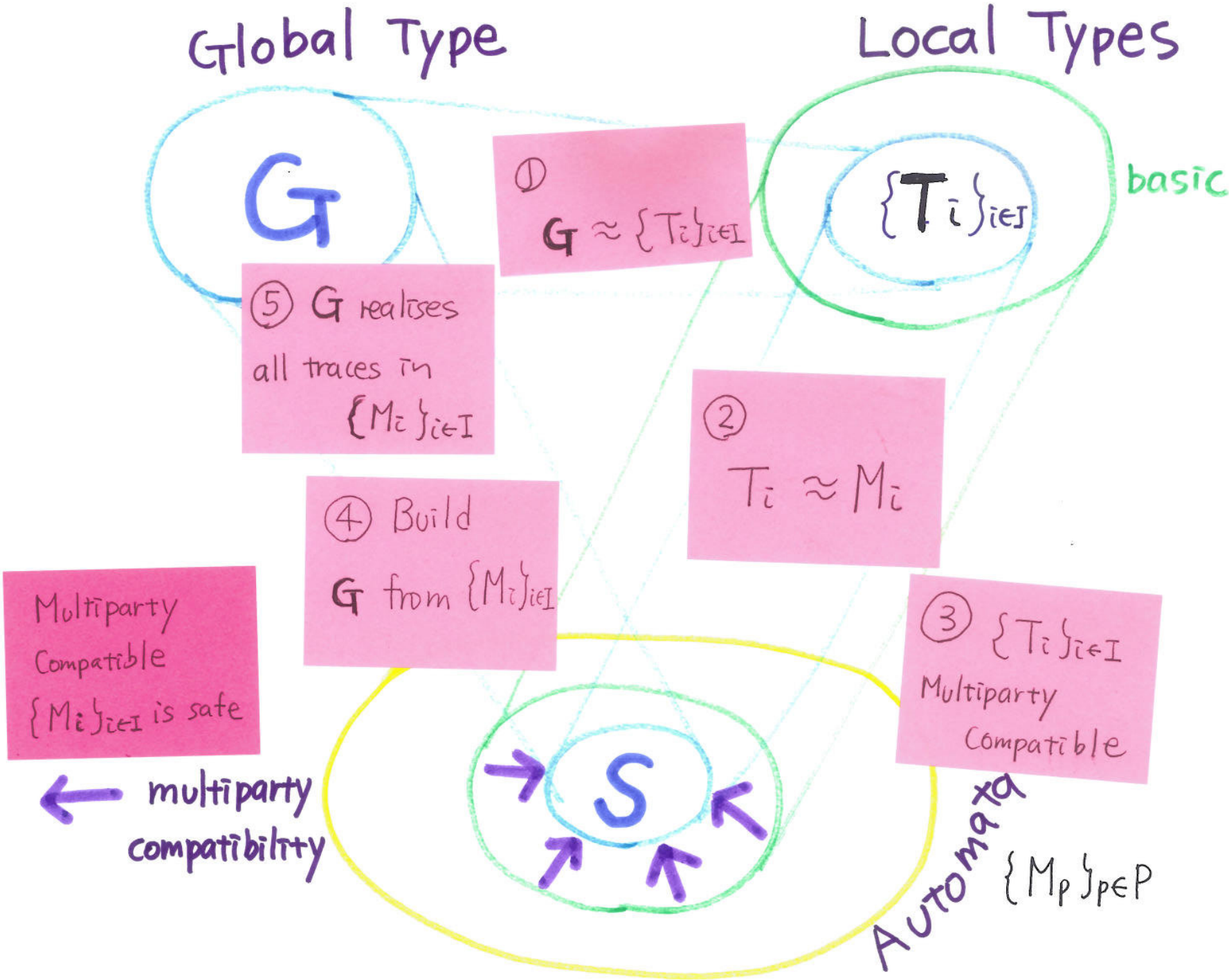
$$Tr(G) \approx Tr(\{T_i\}_{i \in I}) \qquad \text{Global} \approx \text{Local}$$

$$\approx Tr(\{A(T_i)\}_{i \in I}) \qquad \text{Local} \approx \text{CFSM}$$

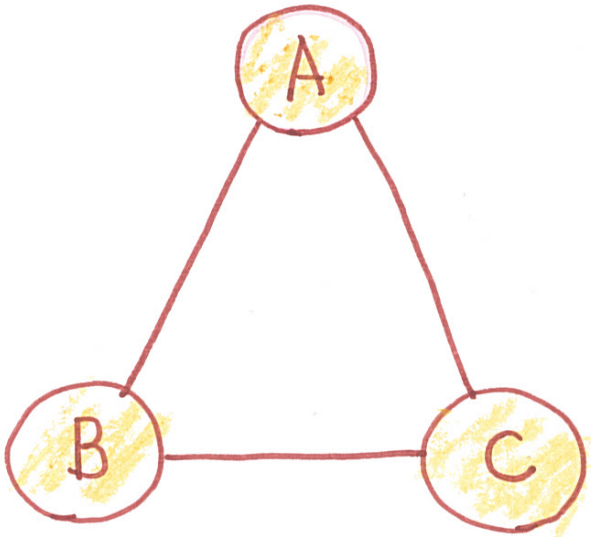
$$\approx_1 Tr(S) \qquad \text{Synthesis}$$

$$\approx Tr(S) \qquad \text{Lemma}$$

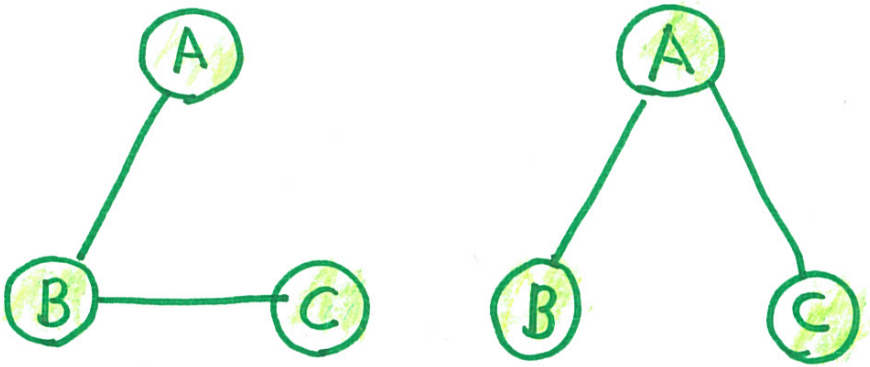
Suppose S is basic and compatible. Then there exists G s.t. $G \approx S$. If G is projectable, then there exists a multiparty compatible basic S s.t. S is safe and $G \approx S$.

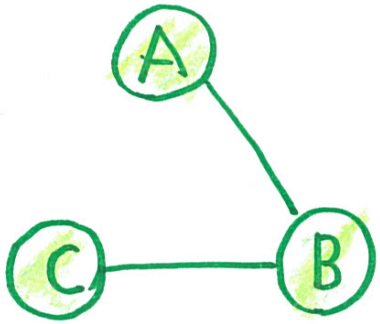

Global Type
Local Types
G
① G ≈ {Ti}i∈I
{Ti}i∈I
basic
⑤ G realises all traces in {Mi}i∈I
② Ti ≈ Mi
④ Build G from {Mi}i∈I
Multiparty Compatible {Mi}i∈I is safe
multiparty compatibility
S
③ {Ti}i∈I Multiparty Compatible
Automata
{Mp}p∈P

Interconnectability of Session Logical Processes



Multi Party





Linear Logical Session

cf. [1995] Specification Structures and Proposition as Types

Mapping and Fusing

Linear Logic

Multiparty

Thread

cut-free process

same participant

$x(z).\ \overline{y}\langle 1\rangle$

$S[p][q](z).\ S[p][r]\langle 1\rangle$

Partial Global Type

$p \rightsquigarrow q\langle Nat\rangle\ ;\ p \rightsquigarrow r\langle Int\rangle$

Fuse Compositionally

Logic — Binary

$$\frac{P \vdash \Delta, x:A \qquad Q \vdash \Delta', x:A^{\perp}}{(\nu x)(P \mid Q) \vdash \Delta, \Delta'}$$

$F(p \rightsquigarrow q\langle Int\rangle.\ p \rightsquigarrow r\langle Bool\rangle,\ r \rightsquigarrow p\langle Bool\rangle.\ r \rightsquigarrow s\langle Nat\rangle)$

$= \quad p \rightsquigarrow q\langle Int\rangle.\ p \longrightarrow r\langle Bool\rangle.\ r \rightsquigarrow s\langle Nat\rangle.$

Mapping and Fusing

Linear Logic

Multiparty

Thread

cut-free process

same participant

$x(z).\ \overline{y}\langle 1\rangle$

$S[p][q](z).\ S[p][r]\langle 1\rangle$

Partial Global Type

$p \rightsquigarrow q\langle Nat\rangle\ ;\ p \rightsquigarrow r\langle Int\rangle$

Fuse Compositionally

Logic — Binary

$$\frac{P \vdash \Delta, x:A \qquad Q \vdash \Delta', x:A^{\perp}}{(\nu x)(P \mid Q) \vdash \Delta, \Delta'}$$

$F(p \rightsquigarrow q\langle Int\rangle.\ p \rightsquigarrow r\langle Bool\rangle,\ r \rightsquigarrow p\langle Bool\rangle.\ r \rightsquigarrow s\langle Nat\rangle)$

$= \quad p \rightsquigarrow q\langle Int\rangle.\ p \longrightarrow r\langle Bool\rangle.\ r \rightsquigarrow s\langle Nat\rangle.$

Theorem (uniqueness)

There is only one unique mapping from LL to MP which preserves "cut-free process" (modulo bijective renaming)

Theorem (Interconnection Network)

ING : interconnection glaph (NODES: PARTICIPANTS, EDGES $p \leftrightarrow q$)
in LL is acyclic.

proof Fuse is acyclic. and Fused Global Types are Partical Multiparty Compatible

Results

① No name passing LL → MP with a Single Session

② Delegation → Several MPs

③ Replications → No difference

[Multicut by PMC]

$P \vdash \Delta, x_1 : A_1, \ldots, x_n : A_n \qquad Q \vdash \Delta', x_1 : A_1^{\perp}, \ldots, x_n : A_n^{\perp}$

$G \leftarrow$ partial MC fusable $\rightarrow G'$

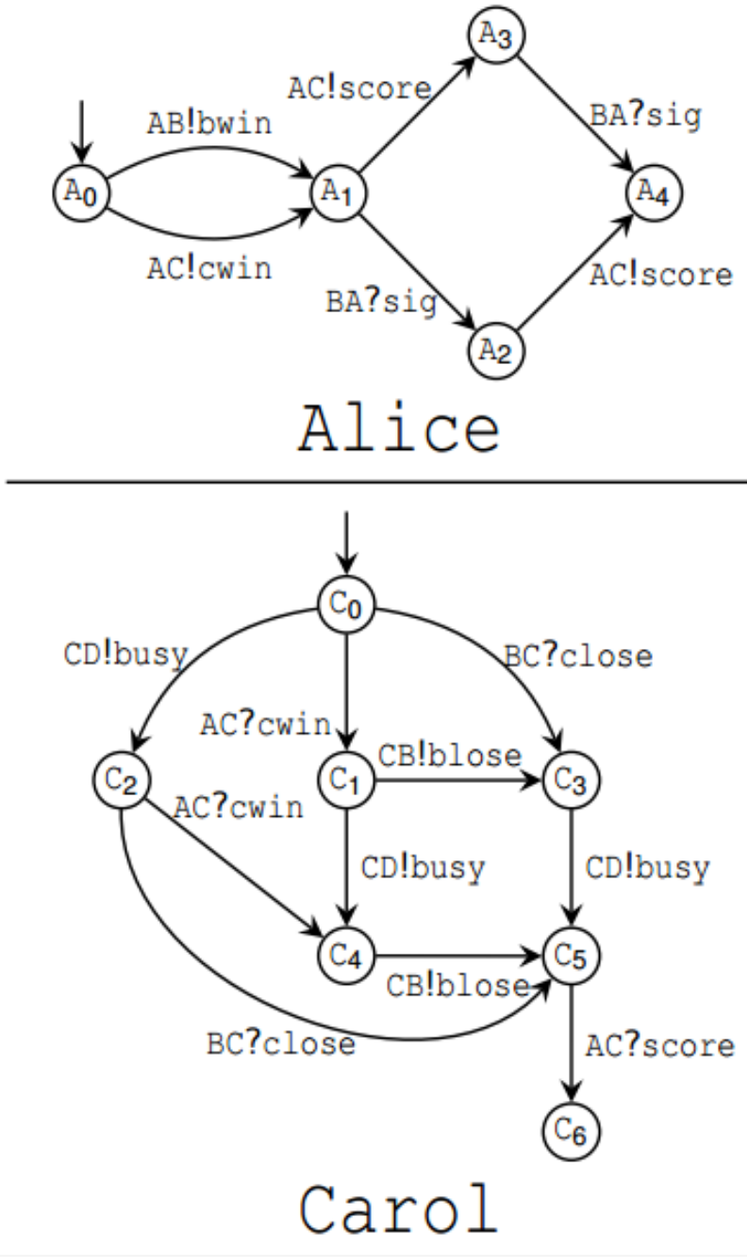
$(\nu \tilde{x})(P \mid Q) \vdash \Delta, \Delta'$

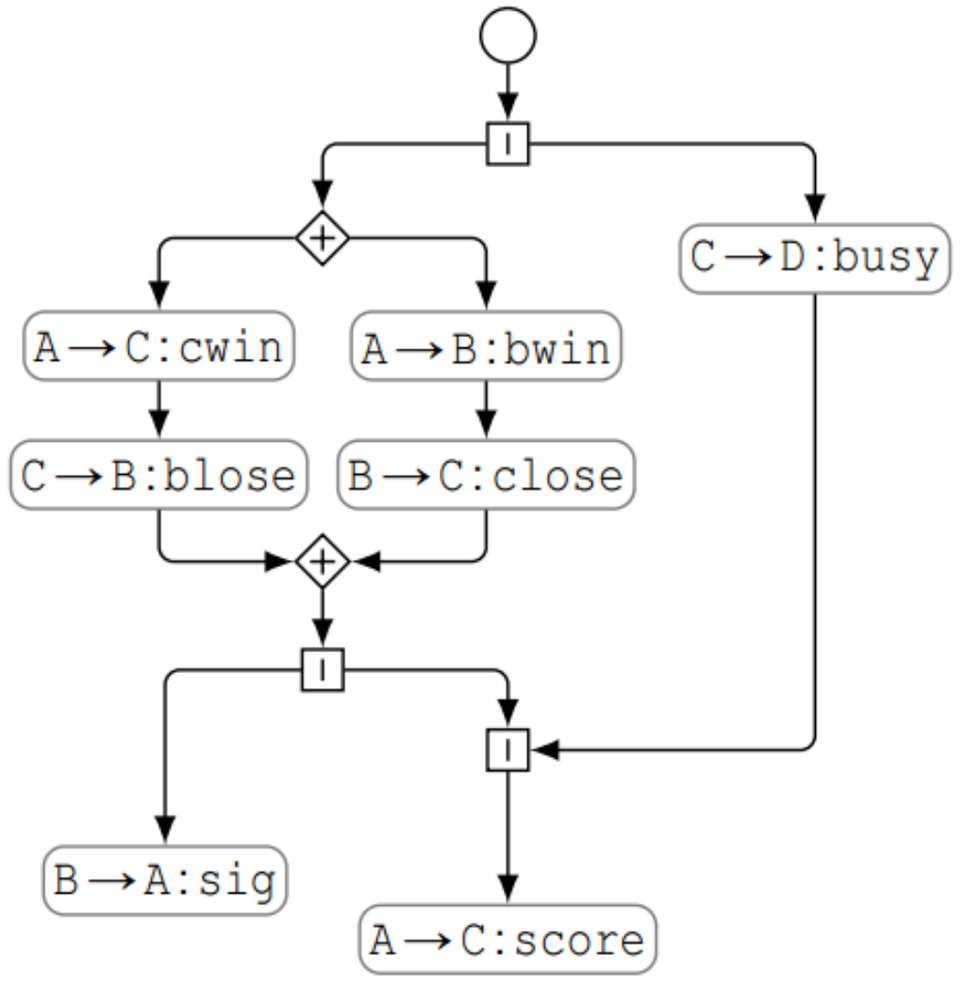
Theorem No Deadlock · Terminations

cf. • More Typable Terms than [Concur '15 / Concur '16] since PMC is more fine-grained.

• No Need of Mediator or Annotations

From Communicating Machines to Graphical Choreographies [POPL'15, CONCUR'15]

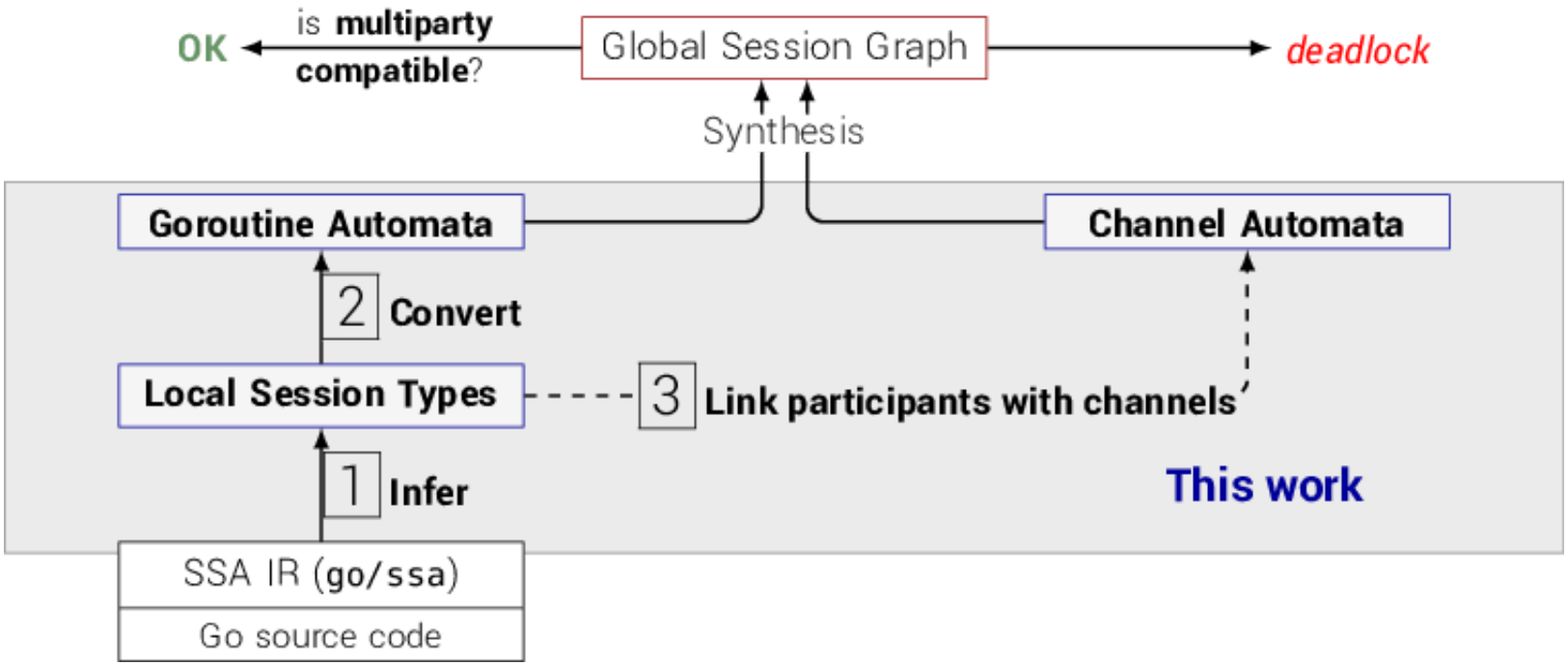




[ESOP'10,ESOP'12,CONCUR'12,CONCUR'14]

Contributions

- Static deadlock detection tool *dingo-hunter*
- Deadlock detection based on session types
- **Infer** session types as Communicating Automata
- **Synthesise** global session graphs from CA

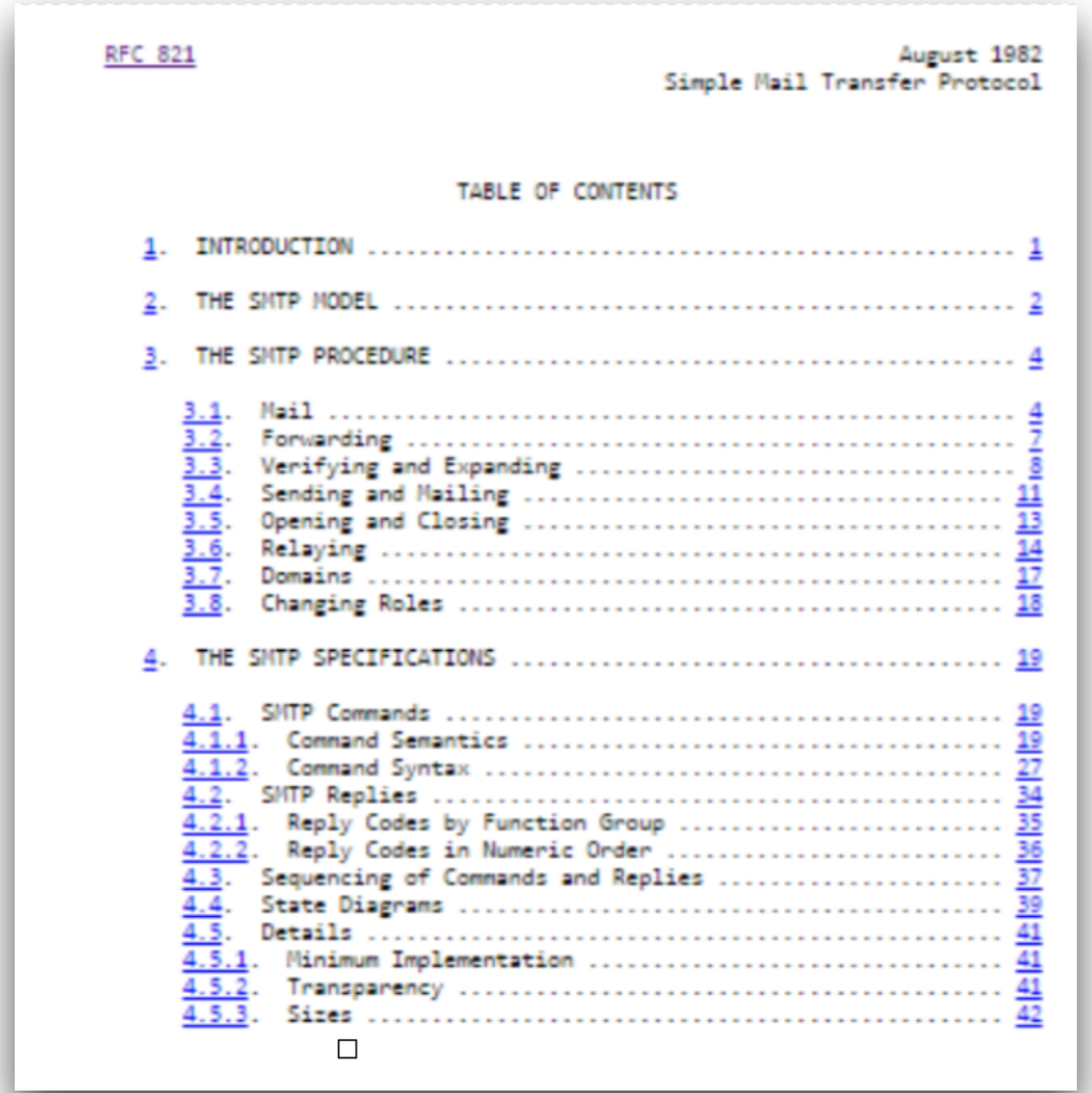


Go and Concurrency

- Developed by Google for multi-core programming
- Concurrency model built on CSP (process calculi)
- Message-passing **communication** over channels

" *Do not communicate by sharing memory*; *instead*, *share memory by communicating*. "
-- Effective Go (developer guide)

Java API Generation [FASE'16]



RFC 821 August 1982
Simple Mail Transfer Protocol

TABLE OF CONTENTS