Global Types for Dynamic Checking of Protocol Conformance of Multi-Agent Systems

Davide Ancona, Viviana Mascardi and Matteo Barbieri

Università di Genova

CINA kick-off meeting, Pisa, February 4-6, 2013

Outline

Outline

Research topics and aims

Work Package 2
Negotiation: Transactions, Reversibility and Compensations

Task 2.4: Contracts and Compliance

Session types for multi-agent systems

Aims

- Dynamic checking of protocol conformance in multi-agent systems
- Expressive formalisms for specifying complex protocols
- Protocol specifications as first-order values
- Integration with static checking, gradual typing (possible future directions)

Motivations

- Testing multi-agent systems is a difficult task

Motivations

- Testing multi-agent systems is a difficult task
- Dynamic checking
  - Verification of complex protocols
  - Verification of both control and data flow
  - Verification of non functional properties (e.g. responsiveness)

Motivations

- Testing multi-agent systems is a difficult task
- Dynamic checking
  - Verification of complex protocols
  - Verification of both control and data flow
  - Verification of non functional properties (e.g. responsiveness)
- Expressivity
  - Complex protocols should be expressible in a succinct way
  - Specifications should be human-readable

Motivations

- Testing multi-agent systems is a difficult task
- Dynamic checking
  - Verification of complex protocols
  - Verification of both control and data flow
  - Verification of non functional properties (e.g. responsiveness)
- Expressivity
  - Complex protocols should be expressible in a succinct way
  - Specifications should be human-readable
- Protocol specifications as first-order values
  - Agents can send and receive protocol specifications
  - Interesting feature for adaptivity through behavioral types (WP1, T1.3), and run-time monitoring for global correctness (WP5, T5.3)

Motivations

- Testing multi-agent systems is a difficult task
- Dynamic checking
  - Verification of complex protocols
  - Verification of both control and data flow
  - Verification of non functional properties (e.g. responsiveness)
- Expressivity
  - Complex protocols should be expressible in a succinct way
  - Specifications should be human-readable
- Protocol specifications as first-order values
  - Agents can send and receive protocol specifications
  - Interesting feature for adaptivity through behavioral types (WP1, T1.3), and run-time monitoring for global correctness (WP5, T5.3)
- Static checking
  - Stronger guarantees
  - More challenging, more conservative analysis

Used formalisms

- Behavioral types
  - Global types: specify agent interaction globally
  - Session types: specify agent interaction locally

Used formalisms

- Behavioral types
  - Global types: specify agent interaction globally
  - Session types: specify agent interaction locally
- Contractive regular types (guarded cyclic terms)
  - Natural support for recursion
  - Coinductive interpretation: infinite interactions are allowed

Used formalisms

- Behavioral types
  - Global types: specify agent interaction globally
  - Session types: specify agent interaction locally
- Contractive regular types (guarded cyclic terms)
  - Natural support for recursion
  - Coinductive interpretation: infinite interactions are allowed
- Trace-based interpretation
  - Defined in terms of a labeled transition system
  - Dynamic checking obtained through the implementation of the transition relation

Outline

Multi-agent systems (MASs)

- industrial-strength technology for integrating and coordinating heterogeneous systems
- intrinsically distributed nature, asynchronous message passing
- agent-oriented programming languages are typically dynamically typed

Jason

- AgentSpeak: a logic-based agent-oriented programming language, based on the belief-desire-intention (BDI) software model
- Jason: open source interpreter for an extended version of AgentSpeak, supporting a Prolog-like language for specifying agents behavior
- communication model: speech-act based, with performatives (a.k.a. illocutionary forces)

Sending actions in Jason

```
.send(recipient,performative,content)
```

- *recipient*: the *id* of the agent that will receive the message
- *performative*: specifies the semantics/aim of the message

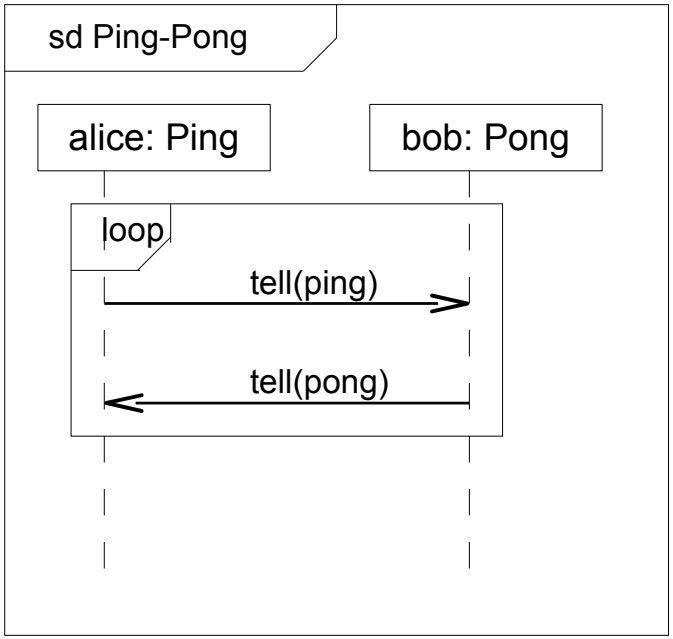
  `tell untell achieve unachieve tell-how untell-how ask-if ask-all ask-how`

- *content*: a (possibly empty) set of atoms or plans

Protocol specification for multi-agent systems

Interaction diagrams in FIPA AUML

- specify the behavior of a system from a global point of view
- suitable for humans, but not for verification
- not expressive enough in some situations



[FIPA Modeling: Interaction Diagrams,
`http://www.auml.org/auml/documents/ID-03-07-02.pdf`]

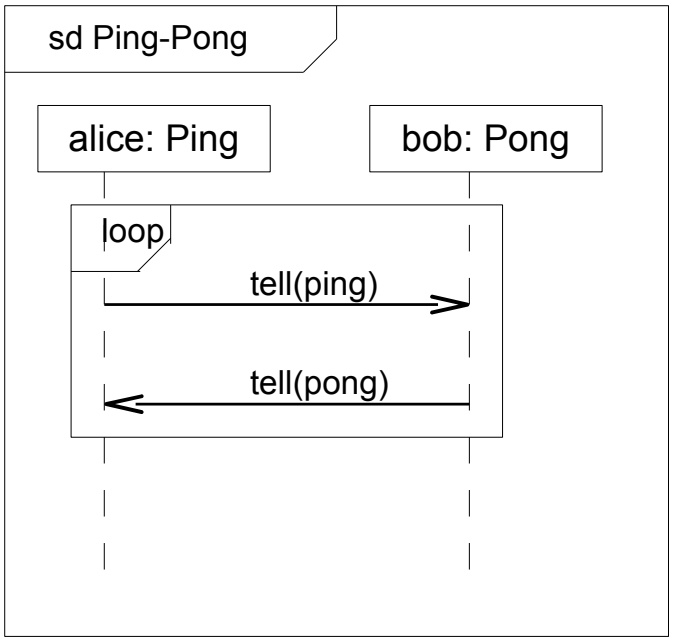
Interpretation of protocol specification

Protocol

set of (possibly infinite) sequences of sending actions

Example: ping-pong protocol

```
msg(alice,bob,tell,ping) msg(bob,alice,tell,pong)
     msg(alice,bob,tell,ping) msg(bob,alice,tell,pong) ...
```



Outline

Protocols specified by global types

Sending action types

Monadic predicates which define a set of valid sending actions
Example:
`ping` corresponds to `msg(alice,bob,tell,ping)`
`pong` corresponds to `msg(bob,alice,tell,pong)`

Protocols specified by global types

Sending action types

Monadic predicates which define a set of valid sending actions
Example:
`ping` corresponds to `msg(alice,bob,tell,ping)`
`pong` corresponds to `msg(bob,alice,tell,pong)`

A global type for the ping-pong protocol

```
PingPong = ping:pong:PingPong
```

α:*T* means

any message action specified by the sending action type α,
followed by any sequence specified by the global type *T*

Global types as Prolog cyclic terms

`PingPong` is a contractive regular type defined by a recursive syntactic equation

- Modern Prolog systems (and Jason as well) support cyclic terms (a.k.a. regular terms)
- Example: the unification problem

  ```
  PingPong = ping:pong:PingPong.
  ```

  succeeds with the answer `PingPong = ping:pong:PingPong`
- Regular terms naturally support recursive types
- Regular Prolog terms: a very compact representation of protocol specifications through global types

Global types as Prolog cyclic terms

`PingPong` is a contractive regular type defined by a recursive syntactic equation

- Modern Prolog systems (and Jason as well) support cyclic terms (a.k.a. regular terms)
- Example: the unification problem

  ```
  PingPong = ping:pong:PingPong.
  ```

  succeeds with the answer `PingPong = ping:pong:PingPong`
- Regular terms naturally support recursive types
- Regular Prolog terms: a very compact representation of protocol specifications through global types

Protocol specifications as first-class values

Protocols can be easily manipulated and exchanged by agents

Constructors for global types

- λ: *empty sequence* (denoting $\{\epsilon\}$)
- $\alpha{:}\tau$ *sequence constructor* (already seen)
- $\tau_1 + \tau_2$ *choice* (union)
- $\tau_1 | \tau_2$ *fork* (shuffle)
- $\tau_1 \cdot \tau_2$ *concat* (concatenation)

Interpretation of global types

- $[\![\tau]\!]$ = set of finite/infinite sequences of sending actions determined by the transition relation
- Finite sequences are allowed if the final type τ specifies a protocol that can terminate, that is, $\epsilon(\tau)$ holds

Interpretation of global types

- $[\![\tau]\!]$ = set of finite/infinite sequences of sending actions determined by the transition relation
- Finite sequences are allowed if the final type τ specifies a protocol that can terminate, that is, $\epsilon(\tau)$ holds

Definition of $\epsilon(\_)$

$$(\epsilon\text{-seq})\frac{}{\epsilon(\lambda)} \qquad (\epsilon\text{-lchoice})\frac{\epsilon(\tau_1)}{\epsilon(\tau_1+\tau_2)} \qquad (\epsilon\text{-rchoice})\frac{\epsilon(\tau_2)}{\epsilon(\tau_1+\tau_2)}$$

$$(\epsilon\text{-fork})\frac{\epsilon(\tau_1)\quad\epsilon(\tau_2)}{\epsilon(\tau_1|\tau_2)} \qquad (\epsilon\text{-cat})\frac{\epsilon(\tau_1)\quad\epsilon(\tau_2)}{\epsilon(\tau_1\cdot\tau_2)}$$

Example: $[\![\lambda+\texttt{ping}:\lambda]\!] = \{\epsilon, \texttt{msg(alice,bob,tell,ping)}\}$

Transition rules

Simple version, unconstrained shuffle

$$\text{(seq)}\ \frac{}{\alpha{:}\tau \xrightarrow{a} \tau}\ a{\in}\alpha \qquad \text{(choice-l)}\ \frac{\tau_1 \xrightarrow{a} \tau_1'}{\tau_1 + \tau_2 \xrightarrow{a} \tau_1'} \qquad \text{(choice-r)}\ \frac{\tau_2 \xrightarrow{a} \tau_2'}{\tau_1 + \tau_2 \xrightarrow{a} \tau_2'}$$

$$\text{(fork-l)}\ \frac{\tau_1 \xrightarrow{a} \tau_1'}{\tau_1 \,|\, \tau_2 \xrightarrow{a} \tau_1' \,|\, \tau_2} \qquad \text{(fork-r)}\ \frac{\tau_2 \xrightarrow{a} \tau_2'}{\tau_1 \,|\, \tau_2 \xrightarrow{a} \tau_1 \,|\, \tau_2'}$$

$$\text{(cat-l)}\ \frac{\tau_1 \xrightarrow{a} \tau_1'}{\tau_1 \cdot \tau_2 \xrightarrow{a} \tau_1' \cdot \tau_2} \qquad \text{(cat-r)}\ \frac{\tau_2 \xrightarrow{a} \tau_2'}{\tau_1 \cdot \tau_2 \xrightarrow{a} \tau_2'}\ \epsilon(\tau_1)$$

Outline

Alternating bit protocol (ABP(4))

- A typical benchmark for testing expressivity (see DeniélouYoshida@ESOP2012)
- Infinite sequences of the following sending actions:
  - `msg(alice,bob,tell,msg1)` (sending action type `msg1`)
  - `msg(alice,bob,tell,msg2)` (sending action type `msg2`)
  - `msg(bob,alice,tell,ack1)` (sending action type `ack1`)
  - `msg(bob,alice,tell,ack2)` (sending action type `ack2`)

Required constraints (for all n):

- $\texttt{msg1}_n < \texttt{ack1}_n < \texttt{msg1}_{n+1}$
- $\texttt{msg2}_n < \texttt{ack2}_n < \texttt{msg2}_{n+1}$
- $\texttt{msg1}_n < \texttt{msg2}_n < \texttt{msg1}_{n+1}$

α_n denotes the n-th occurrence of the sending action a of type α

A global type specifying ABP(4)

A first attempt

$$WrongAltBit = MA_1 | MA_2$$
$$MA_1 = msg_1 : ack_1 : MA_1$$
$$MA_2 = msg_2 : ack_2 : MA_2$$

- $\texttt{msg1}_n < \texttt{ack1}_n < \texttt{msg1}_{n+1}$
- $\texttt{msg2}_n < \texttt{ack2}_n < \texttt{msg2}_{n+1}$
- $\texttt{msg1}_n < \texttt{msg2}_n < \texttt{msg1}_{n+1}$

A global type specifying ABP(4)

A first attempt

$$WrongAltBit = MA_1 | MA_2$$
$$MA_1 = msg_1{:}ack_1{:}MA_1$$
$$MA_2 = msg_2{:}ack_2{:}MA_2$$

- $\mathtt{msg1}_n < \mathtt{ack1}_n < \mathtt{msg1}_{n+1}$ **OK**
- $\mathtt{msg2}_n < \mathtt{ack2}_n < \mathtt{msg2}_{n+1}$
- $\mathtt{msg1}_n < \mathtt{msg2}_n < \mathtt{msg1}_{n+1}$

A global type specifying ABP(4)

A first attempt

$$WrongAltBit = MA_1 | MA_2$$
$$MA_1 = msg_1 : ack_1 : MA_1$$
$$MA_2 = msg_2 : ack_2 : MA_2$$

- $\texttt{msg1}_n < \texttt{ack1}_n < \texttt{msg1}_{n+1}$ **OK**
- $\texttt{msg2}_n < \texttt{ack2}_n < \texttt{msg2}_{n+1}$ **OK**
- $\texttt{msg1}_n < \texttt{msg2}_n < \texttt{msg1}_{n+1}$

A global type specifying ABP(4)

A first attempt

$$WrongAltBit = MA_1 | MA_2$$
$$MA_1 = msg_1 : ack_1 : MA_1$$
$$MA_2 = msg_2 : ack_2 : MA_2$$

- $\mathtt{msg1}_n < \mathtt{ack1}_n < \mathtt{msg1}_{n+1}$ **OK**
- $\mathtt{msg2}_n < \mathtt{ack2}_n < \mathtt{msg2}_{n+1}$ **OK**
- $\mathtt{msg1}_n < \mathtt{msg2}_n < \mathtt{msg1}_{n+1}$ **NO**

A global type specifying ABP(4)

Correct type

$$
\begin{aligned}
AltBit_1 &= msg_1{:}M_2 \\
AltBit_2 &= msg_2 : M_1 \\
M_1 &= (msg_1 : A_2) + (ack_2 : AltBit_1) \\
A_1 &= (ack_1 : M_1) + (ack_2 : ack_1 : AltBit_1) \\
M_2 &= (msg_2 : A_1) + (ack_1 : AltBit_2) \\
A_2 &= (ack_2 : M_2) + (ack_1 : ack_2 : AltBit_2)
\end{aligned}
$$

Problems:

- quite complex type, not very readable
- the complexity of the type grows exponentially with the number of messages

A global type specifying ABP(6)

$$
\begin{aligned}
AltBit_3 &= msg_1 : S_1 \\
S_1 &= (msg_2 : S_2) + (ack_1 : msg_2 : S_6) \\
S_2 &= (ack_1 : S_6) + ((ack_2 : S_4) + (msg_3 : S_3)) \\
S_3 &= (ack_1 : S_7) + ((ack_2 : S_8) + (ack_3 : S_5)) \\
S_4 &= (ack_1 : msg_3 : ack_3 : AltBit_3) + (msg_3 : S_8) \\
S_5 &= (ack_1 : ack_2 : AltBit_3) + (ack_2 : ack_1 : AltBit_3) \\
S_6 &= (msg_3 : S_7) + (ack_2 : msg_3 : ack_3 : AltBit_3) \\
S_7 &= (ack_3 : ack_2 : AltBit_3) + (ack_2 : ack_3 : AltBit_3) \\
S_8 &= (ack_1 : ack_3 : AltBit_3) + (ack_3 : ack_1 : AltBit_3)
\end{aligned}
$$

Outline

Constrained shuffle

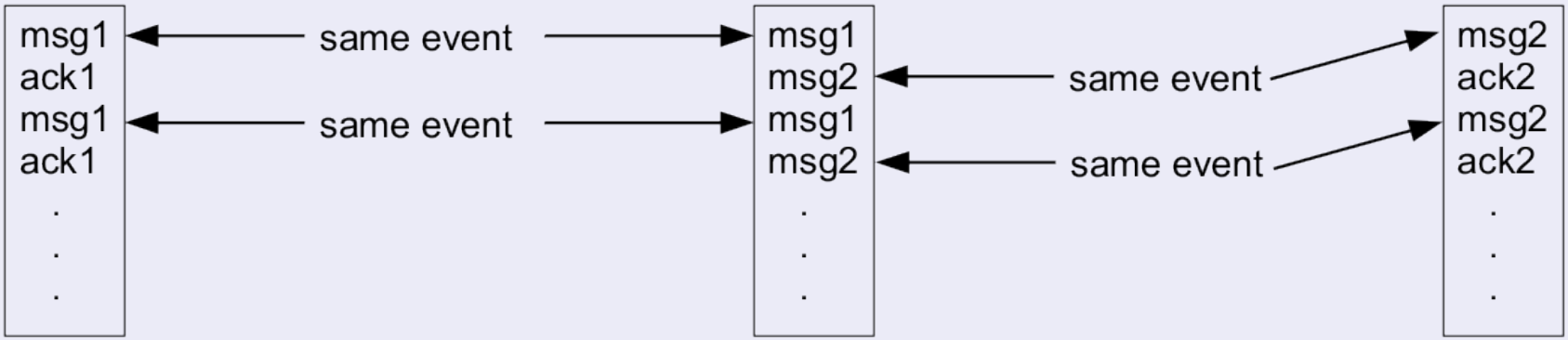
Intuition

For every correct sequence s of ABP(4)

- s restricted to `msg1` and `ack1` is in $MA_1 = msg_1{:}ack_1{:}MA_1$
- s restricted to `msg2` and `ack2` is in $MA_2 = msg_2{:}ack_2{:}MA_2$
- s restricted to `msg1` and `msg2` is in $MM = msg_1{:}msg_2{:}MM$

But $MA_1 | MA_2 | MM$ is not correct!

Shuffle needs to be constrained



Extended global types

- Constrained shuffle
- *Producer* sending action type α^n: must be "consumed" by n consumer types ($n \geq 0$)
- *Consumer* sending action type α

An unconstrained shuffle contains just producer sending action types of shape α^0

Transition rules (revisited)

Extended version, constrained shuffle

$$\text{(seq-prod)}\ \frac{}{0, \alpha^n{:}\tau \xrightarrow{a} \tau, n}\ a{\in}\alpha \qquad \text{(seq-cons)}\ \frac{}{n, \alpha{:}\tau \xrightarrow{a} \tau, n-1}\ \begin{array}{l} n{>}0 \\ a{\in}\alpha \end{array}$$

$$\text{(fork-both-l)}\ \frac{n_1, \tau_1 \xrightarrow{a} \tau_1', n_2 \quad n_2, \tau_2 \xrightarrow{a} \tau_2', n_3}{n_1, \tau_1 | \tau_2 \xrightarrow{a} \tau_1' | \tau_2', n_3}\ n_2{>}0$$

$$\text{(fork-both-r)}\ \frac{n_1, \tau_2 \xrightarrow{a} \tau_2', n_2 \quad n_2, \tau_1 \xrightarrow{a} \tau_1', n_3}{n_1, \tau_1 | \tau_2 \xrightarrow{a} \tau_1' | \tau_2', n_3}\ n_2{>}0$$

$$\text{(fork-l)}\ \frac{n_1, \tau_1 \xrightarrow{a} \tau_1', n_2}{n_1, \tau_1 | \tau_2 \xrightarrow{a} \tau_1' | \tau_2, n_2} \qquad \text{(fork-r)}\ \frac{n_1, \tau_2 \xrightarrow{a} \tau_2', n_2}{n_1, \tau_1 | \tau_2 \xrightarrow{a} \tau_1 | \tau_2', n_2}$$

$$\text{(choice-l)}\ \frac{n_1, \tau_1 \xrightarrow{a} \tau_1', n_2}{n_1, \tau_1 + \tau_2 \xrightarrow{a} \tau_1', n_2} \qquad \text{(choice-r)}\ \frac{n_1, \tau_2 \xrightarrow{a} \tau_2'}{n_1, \tau_1 + \tau_2 \xrightarrow{a} \tau_2', n_2}$$

$$\text{(cat-l)}\ \frac{n_1, \tau_1 \xrightarrow{a} \tau_1', n_2}{n_1, \tau_1 \cdot \tau_2 \xrightarrow{a} \tau_1' \cdot \tau_2, n_2} \qquad \text{(cat-r)}\ \frac{n_1, \tau_2 \xrightarrow{a} \tau_2', n_2}{n_1, \tau_1 \cdot \tau_2 \xrightarrow{a} \tau_2', n_2}\ \epsilon(\tau_1)$$

Alternating bit protocol (revisited)

ABP(4)

$$AB = MA'_1 | MA'_2 | MM$$
$$MA'_1 = msg_1^1 : ack_1^0 : MA'_1$$
$$MA'_2 = msg_2^1 : ack_2^0 : MA'_2$$
$$MM = msg_1 : msg_2 : MM$$

ABP(6)

$$AB_3 = MA_1 | MA_2 | MA_3 | MM_3$$
$$MA_1 = msg_1^1 : ack_1^0 : MA_1$$
$$MA_2 = msg_2^1 : ack_2^0 : MA_2$$
$$MA_3 = msg_3^1 : ack_3^0 : MA_3$$
$$MM_3 = msg_1 : msg_2 : msg_3 : MM_3$$

A comparison

ABP(4): proposed solution in DeniélouYoshida@ESOP2012

$$
\begin{aligned}
\mathbf{G}_{ABP} = \text{def } \mathbf{x_0} &= \mathbf{x}_1 \mid \mathbf{x}_2 \\
\mathbf{x}_1 + \mathbf{x}_3 &= \mathbf{x}_4 \\
\mathbf{x}_2 + \mathbf{x}_5 &= \mathbf{x}_6 \\
\mathbf{x}_4 &= \texttt{Alice} \rightarrow \texttt{Bob} : \mathit{Msg}_1\langle\, \texttt{string} \,\rangle; \mathbf{x}_7 \\
\mathbf{x}_7 &= \mathbf{x}_8 \mid \mathbf{x}_9 \\
\mathbf{x}_8 &= \texttt{Bob} \rightarrow \texttt{Alice} : \mathit{Ack}_1\langle\, \texttt{unit} \,\rangle; \mathbf{x}_{10} \\
\mathbf{x}_6 \mid \mathbf{x}_9 &= \mathbf{x}_{11} \\
\mathbf{x}_{11} &= \texttt{Alice} \rightarrow \texttt{Bob} : \mathit{Msg}_2\langle\, \texttt{string} \,\rangle; \mathbf{x}_{12} \\
\mathbf{x}_{12} &= \mathbf{x}_{13} \mid \mathbf{x}_{14} \\
\mathbf{x}_{13} &= \texttt{Bob} \rightarrow \texttt{Alice} : \mathit{Ack}_2\langle\, \texttt{unit} \,\rangle; \mathbf{x}_5 \\
\mathbf{x}_{10} \mid \mathbf{x}_{14} &= \mathbf{x}_3 \text{ in } \mathbf{x}_0
\end{aligned}
$$

Outline


D. Ancona, M. Barbieri, and V. Mascardi.
Global types for dynamic checking of protocol conformance of multi-agent systems (extended abstract).
In *ICTS 2012*, pages 39–43, 2012.


D. Ancona, M. Barbieri, and V. Mascardi.
Constrained global types for dynamic checking of protocol conformance in multi-agent systems.
SAC 2013. ACM. To appear, 2013.


D. Ancona, S. Drossopoulou, and V. Mascardi.
Automatic Generation of Self-Monitoring MASs from Multiparty Global Session Types in Jason.
In *DALT X. Revised, Selected and Invited Papers*, volume 7784 of *LNAI*. Springer, 2012.


G. Castagna, M. Dezani-Ciancaglini, and L. Padovani.
On global types and multi-party session.
Logical Methods in Computer Science, 8(1), 2012.


P.-M. Deniélou and N. Yoshida.
Multiparty session types meet communicating automata.
In *ESOP'12*, LNCS. Springer, 2012.

Thank you for your attention...

...questions?