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

Davide Ancona, Matteo Barbieri and Viviana Mascardi

Università di Genova

Italian Conference on Theoretical Computer Science, Varese, September 19-21, 2012

Outline

1. Background on multi-agent systems
2. Previous work (Declarative Agent Languages and Technologies - DALT 2012, Ancona, Drossopoulou, Mascardi)
3. Global types: formalization
4. Expressive power of global types (by examples)
5. An extension to enhance the expressive power (not in the paper)
6. Conclusion and future work

Outline

1. Background on multi-agent systems
2. Previous work (Declarative Agent Languages and Technologies - DALT 2012, Ancona, Drossopoulou, Mascardi)
3. Global types: formalization
4. Expressive power of global types (by examples)
5. An extension to enhance the expressive power (not in the paper)
6. Conclusion and future work

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

Outline

Protocols and multi-agent systems

A protocol represents an agreement on how participating agents [systems] interact with each other. Without a protocol, it is hard to do a meaningful interaction: participants simply cannot communicate effectively.

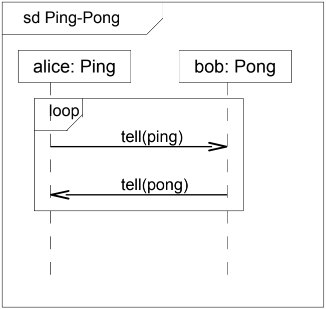
[From the manifesto of Scribble, a language to describe application-level protocols among communicating systems initially designed by Kohei Honda and Gary Brown, `http://www.jboss.org/scribble/`]

Protocol specification

Interaction diagrams in FIPA AUML

- specify the behavior of a system from a global point of view
- suitable for humans, but not for verification

A first example: ping-pong protocol



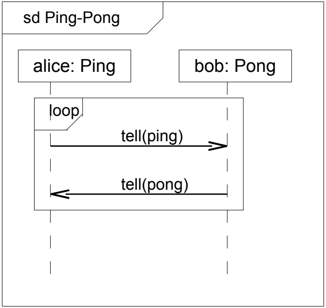
[FIPA Modeling: Interaction Diagrams,
`http://www.auml.org/auml/documents/ID-03-07-02.pdf`]

Protocol specification: a formal approach

protocol =
(possibly infinite) set of (possibly infinite) sequences of sending actions

Example 1: ping-pong protocol

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



Protocols as global types

Example 1: ping-pong protocol

`PingPong` = α_1:α_2:`PingPong`

- where α_1 sending action type corresponding to `msg(alice,bob,`**`tell`**`,ping)`
- where α_2 sending action type corresponding to `msg(bob,alice,`**`tell`**`,pong)`
- sending action types = monadic predicates

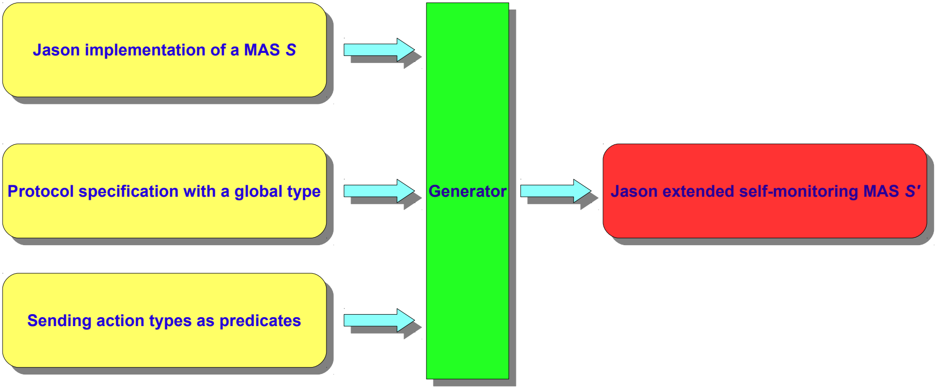
Global types as Prolog cyclic terms

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

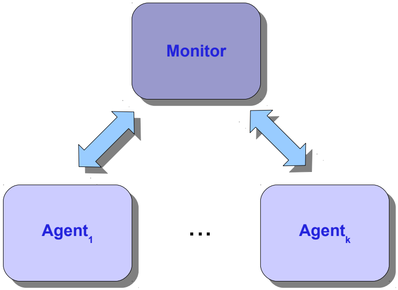
  ```
  PingPong = ping:pong:PingPong.
  ```

  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
- Protocols can be easily manipulated and exchanged by agents

Automatic generation of a self-monitoring MAS



Centralized monitor agent



- protocol conformance dynamically checked by a monitor agent *M*
- other agents ask *M* permission to send their messages
- the monitor notifies all failures
- the monitor checks responsiveness of the agents

Outline

Global types

The set of regular terms defined on the following constructors:

- λ (empty sequence), representing the singleton set $\{\epsilon\}$ containing the empty sequence ϵ.
- $\alpha{:}\tau$ (*seq*), representing the set of all sequences whose first element is a sending action matching type α, and the remaining part is a sequence in the set represented by τ.
- $\tau_1 + \tau_2$ (*choice*), representing the union of the sequences of τ_1 and τ_2.
- $\tau_1 | \tau_2$ (*fork*), representing the set obtained by shuffling the sequences in τ_1 with the sequences in τ_2 .
- $\tau_1 \cdot \tau_2$ (*concat*), representing the set of sequences obtained by concatenating the sequences of τ_1 with those of τ_2.

Contractive global types

A global type τ is *contractive* if it does not contain paths whose nodes can only be constructors in $\{+, |, \cdot\}$ (such paths are necessarily infinite).

Examples:

- a contractive type: $\texttt{T1} = (\lambda + \alpha\texttt{:T1})$
- a non contractive type: $\texttt{T2} = \lambda + (\texttt{T2} \mid \texttt{T2}) + (\texttt{T2} \cdot \texttt{T2})$

Transition rules

- $\mathcal{T}$ contractive global types, $\mathcal{A}$ sending actions
- total function $\delta{:}\mathcal{T} \times \mathcal{A} \rightarrow \mathcal{P}_{fin}(\mathcal{T})$
- $\tau_1 \xrightarrow{a} \tau_2$ means $\tau_2 \in \delta(\tau_1, a)$

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

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

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

Definition of $\epsilon(\_)$

$\epsilon(\tau)$ holds if and only if τ contains λ

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

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

Interpretation of global types

Run

A *run* ρ for τ_0 is a sequence $\tau_0 \xrightarrow{a_0} \tau_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} \tau_n \xrightarrow{a_n} \tau_{n+1} \xrightarrow{a_{n+1}} \ldots$ of valid transitions such that

- either the sequence is infinite,
- or it terminates with the type τ_k (with $k \geq 0$) s.t. $\epsilon(\tau_k)$.

$A(\rho)$ = sequence of sending actions $a_0 a_1 \ldots a_n \ldots$ contained in ρ.

Interpretation

$[\![\tau_0]\!] = \{A(\rho) \mid \rho \text{ is a run for } \tau_0 \}$

Results

Proposition 1

Let τ be a contractive type. Either $\epsilon(\tau)$ holds or there exist *a* and τ' s.t. $\tau \xrightarrow{a} \tau'$.

Proposition 2

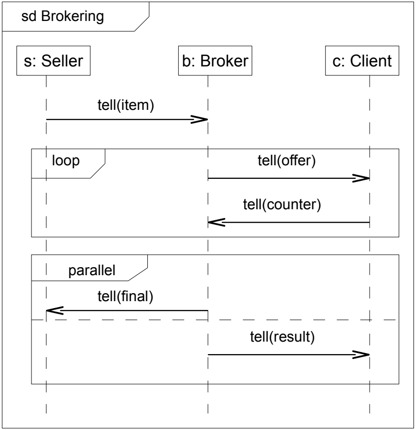
If τ is contractive and $\tau \xrightarrow{a} \tau'$ for some *a*, then τ' is contractive as well.

Corollary

If τ is contractive, then $[\![\tau]\!] \neq \emptyset$

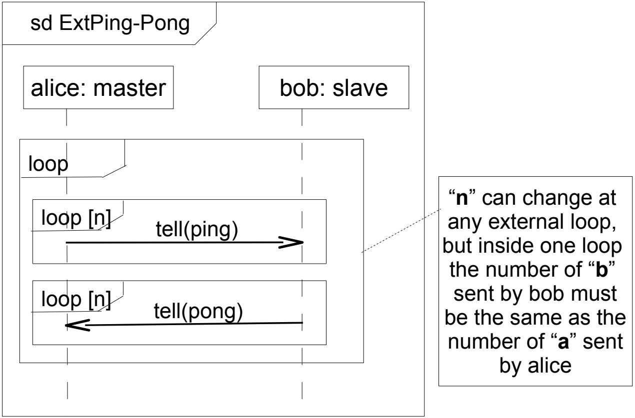
Outline

Example 2: brokering protocol




```
Brokering   = item:(Negotiation + End)
Negotiation = offer:counter:(Negotiation + End)
End         = final:λ | result:λ
```

Example 3: extended ping-pong protocol




```
Loop     = PingPong · Loop
PingPong = ping:(pong:λ + (PingPong · pong:λ))
```

Example 4: alternating bit protocol

Proposed by Deniélou and Yoshida (ESOP 2012)
Infinite sequences of the following sending action types:

- Alice sends `msg1` to Bob
- Alice sends `msg2` to Bob
- Bob sends `ack1` to Alice
- Bob sends `ack2` to Alice

Constraints (for all $n \geq 0$):

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

Where α_n denotes the n-th occurrence of α in the sequence

A global type for the alternating bit protocol

```
AltBitOne = msg1:M2
AltBitTwo = msg2:M1
M2 = (((msg2:λ) | (ack1:λ)) · M1) +
     (((msg2:ack2:λ) | (ack1:λ)) · AltBitOne)
M1 = (((msg1:λ) | (ack2:λ)) · M2) +
     (((msg1:ack1:λ) | (ack2:λ)) · AltBitTwo)
```

Problems:

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

Outline

Global types and constraints

Intuition: for every correct sequence *s*

- *s* restricted to `msg1` and `ack1` is `M1A1 = msg1:ack1:M1A1`
- *s* restricted to `msg2` and `ack2` is `M2A2 = msg2:ack2:M2A2`
- *s* restricted to `msg1` and `msg2` is `M1M2 = msg1:msg2:M1M2`

But neither

`M1A1 | M2A2`

nor

`M1A1 | M2A2 | M1M2`

are correct, since the shuffle is unconstrained

Idea

- Shuffle with a synchronization mechanism
- Producer sending action type: α^n must be synchronized with n consumer types ($n \geq 0$)
- Consumer sending action type: α

An unconstrained global type is a particular case of constrained global type where all sending action types have shape α^0

Extended transition rules

- $n_1, \tau_1 \xrightarrow{a} n_2, \tau_2$
- input n_1: sending action types to be consumed
- output n_2: sending action types left to be consumed
- top-level transition: $0, \tau_1 \xrightarrow{a} 0, \tau_2$

New rules:

$$\text{(seq-prod)}\ \frac{}{0, \alpha^n{:}\tau \xrightarrow{a} n, \tau}\ a \in \alpha \qquad \text{(seq-cons1)}\ \frac{}{n, \alpha{:}\tau \xrightarrow{a} n-1, \tau}\ \begin{array}{l} n>0 \\ a\in\alpha \end{array}$$

$$\text{(seq-cons2)}\ \frac{}{n, \alpha{:}\tau \xrightarrow{a} n, \alpha{:}\tau}\ \begin{array}{l} n>0 \\ a\notin\alpha \end{array} \qquad \text{(empty)}\ \frac{}{n, \lambda \xrightarrow{a} n, \lambda}\ n>0$$

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

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

Extended transition rules

Generalization of the previous rules:

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

$$\text{(fork-l)}\ \frac{n_1, \tau_1 \xrightarrow{a} 0, \tau_1'}{n_1, \tau_1 | \tau_2 \xrightarrow{a} 0, \tau_1' | \tau_2} \qquad \text{(fork-r)}\ \frac{n_1, \tau_2 \xrightarrow{a} 0, \tau_2'}{n_1, \tau_1 | \tau_2 \xrightarrow{a} 0, \tau_1 | \tau_2'}$$

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

Alternating bit protocol (revisited)

Dimension 2

```
AltBit2 = M1A1 | M2A2 | M1M2
M1A1 = msg1^1:ack1^0:M1A1
M2A2 = msg2^1:ack2^0:M2A2
M1M2 = msg1:msg2:M1M2
```

Dimension 3

```
AltBit3 = M1A1 | M2A2 | M3A3 | M1M2M3
M1A1 = msg1^1:ack1^0:M1A1
M2A2 = msg2^1:ack2^0:M2A2
M3A3 = msg3^1:ack3^0:M3A3
M1M2M3 = msg1:msg2:msg3:M1M2M3
```

Outline

Conclusion

- global types as Prolog regular terms
- dynamic checking of protocol conformance
- formalization
- extension to “constrained shuffle”

Future work

- in depth comparison with other formalisms for protocol specification
- relations with ω-automata
- projecting global types
- from dynamic to static checking of protocol conformance

Thank you for your attention...

...questions?