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
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

\[ \text{send}(\text{recipient}, \text{performative}, \text{content}) \]

- **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

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
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

Protocol specification: a formal approach

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

Example 1: ping-pong protocol

\[
\text{msg}(\text{alice}, \text{bob}, \text{tell}, \text{ping}) \quad \text{msg}(\text{bob}, \text{alice}, \text{tell}, \text{pong}) \\
\text{msg}(\text{alice}, \text{bob}, \text{tell}, \text{ping}) \quad \text{msg}(\text{bob}, \text{alice}, \text{tell}, \text{pong}) \ldots
\]
Protocols as global types

Example 1: ping-pong protocol

\[ \text{PingPong} = \alpha_1 : \alpha_2 : \text{PingPong} \]

- where \( \alpha_1 \) sending action type corresponding to
  \[ \text{msg}(\text{alice}, \text{bob}, \text{tell}, \text{ping}) \]
- where \( \alpha_2 \) sending action type corresponding to
  \[ \text{msg}(\text{bob}, \text{alice}, \text{tell}, \text{pong}) \]
- 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
  \[ \text{PingPong} = \text{ping:pong:PingPong}. \]
  succeeds with the answer \[ \text{PingPong} = \text{ping:pong:PingPong} \]
- 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

Jason implementation of a MAS $S$

Protocol specification with a global type

Sending action types as predicates

Generator

Jason extended self-monitoring MAS $S'$
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

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
Global types

The set of regular terms defined on the following constructors:

- \( \lambda \) (empty sequence), representing the singleton set \( \{\epsilon\} \) containing the empty sequence \( \epsilon \).

- \( \alpha: \tau \) (seq), representing the set of all sequences whose first element is a sending action matching type \( \alpha \), and the remaining part is a sequence in the set represented by \( \tau \).

- \( \tau_1 + \tau_2 \) (choice), representing the union of the sequences of \( \tau_1 \) and \( \tau_2 \).

- \( \tau_1 \mid \tau_2 \) (fork), representing the set obtained by shuffling the sequences in \( \tau_1 \) with the sequences in \( \tau_2 \).

- \( \tau_1 \cdot \tau_2 \) (concat), representing the set of sequences obtained by concatenating the sequences of \( \tau_1 \) with those of \( \tau_2 \).
A global type $\tau$ 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: $T_1 = (\lambda + \alpha : T_1)$
- a non contractive type: $T_2 = \lambda + (T_2 | T_2) + (T_2 \cdot T_2)$
Transition rules

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

\[
\begin{align*}
\text{(seq)} & : \quad a \in \alpha \quad \quad \alpha : \mathcal{T} \rightarrow \tau \\
\text{(choice-l)} & : \quad \tau_1 \xrightarrow{a} \tau_1' \quad \tau_1 + \tau_2 \xrightarrow{a} \tau_1' \\
\text{(choice-r)} & : \quad \tau_2 \xrightarrow{a} \tau_2' \quad \tau_1 + \tau_2 \xrightarrow{a} \tau_2' \\
\text{(fork-l)} & : \quad \tau_1 \xrightarrow{a} \tau_1' \quad \tau_1 \mid \tau_2 \xrightarrow{a} \tau_1' \mid \tau_2 \\
\text{(fork-r)} & : \quad \tau_2 \xrightarrow{a} \tau_2' \quad \tau_1 \mid \tau_2 \xrightarrow{a} \tau_1' \mid \tau_2' \\
\text{(cat-l)} & : \quad \tau_1 \xrightarrow{a} \tau_1' \quad \tau_1 \cdot \tau_2 \xrightarrow{a} \tau_1' \cdot \tau_2 \\
\text{(cat-r)} & : \quad \tau_2 \xrightarrow{a} \tau_2' \quad \tau_1 \cdot \tau_2 \xrightarrow{a} \tau_1' \cdot \tau_2 \\
\end{align*}
\]
Definition of $\epsilon(\_)$

$\epsilon(\tau)$ holds if and only if $\tau$ contains $\lambda$

\[
\begin{align*}
(\epsilon\text{-seq}) \quad & \quad \frac{\epsilon(\lambda)}{\epsilon(\lambda)} \\
(\epsilon\text{-lchoice}) \quad & \quad \frac{\epsilon(\tau_1)}{\epsilon(\tau_1 + \tau_2)} \\
(\epsilon\text{-rchoice}) \quad & \quad \frac{\epsilon(\tau_2)}{\epsilon(\tau_1 + \tau_2)} \\
(\epsilon\text{-fork}) \quad & \quad \frac{\epsilon(\tau_1)}{\epsilon(\tau_1 | \tau_2)} \\
(\epsilon\text{-cat}) \quad & \quad \frac{\epsilon(\tau_1)}{\epsilon(\tau_1 \cdot \tau_2)}
\end{align*}
\]
Interpretation of global types

**Run**

A *run* $\rho$ for $\tau_0$ 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 $\tau_k$ (with $k \geq 0$) s.t. $\epsilon(\tau_k)$.

$A(\rho) = \text{sequence of sending actions } a_0a_1\ldots a_n\ldots \text{ contained in } \rho.$

**Interpretation**

$[\tau_0] = \{A(\rho) \mid \rho \text{ is a run for } \tau_0 \}$
Proposition 1

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

Proposition 2

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

Corollary

If $\tau$ is contractive, then $\llbracket \tau \rrbracket \neq \emptyset$
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
Example 2: brokering protocol

Brokering = item:(Negotiation + End)
Negotiation = offer:counter:(Negotiation + End)
End = final:λ | result:λ
Example 3: extended ping-pong protocol

```
sd ExtPing-Pong

alice: master
bob: slave

loop
loop [n] tell(ping)
loop [n] tell(pong)
```

"n" can change at any external loop, but inside one loop the number of "b" sent by bob must be the same as the number of "a" sent by alice

```
Loop = PingPong \cdot Loop
PingPong = ping:(pong:\lambda + (PingPong \cdot pong:\lambda))
```
Example 4: alternating bit protocol

Proposed by Deniélou and Yoshida (ESOP 2012)

Infinite sequences of the following sending action types:

- Alice sends $\text{msg1}$ to Bob
- Alice sends $\text{msg2}$ to Bob
- Bob sends $\text{ack1}$ to Alice
- Bob sends $\text{ack2}$ to Alice

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

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

Where $\alpha_n$ denotes the $n$-th occurrence of $\alpha$ 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

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
Global types and constraints

Intuition: for every correct sequence $s$

- $s$ restricted to $\text{msg1}$ and $\text{ack1}$ is $M_{1A1} = \text{msg1:ack1:M1A1}$
- $s$ restricted to $\text{msg2}$ and $\text{ack2}$ is $M_{2A2} = \text{msg2:ack2:M2A2}$
- $s$ restricted to $\text{msg1}$ and $\text{msg2}$ is $M_{1M2} = \text{msg1:msg2:M1M2}$

But neither $M_{1A1} | M_{2A2}$ nor $M_{1A1} | M_{2A2} | M_{1M2}$ are correct, since the shuffle is unconstrained
Idea

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

An unconstrained global type is a particular case of constrained global type where all sending action types have shape $\alpha^0$
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:

(seq-prod) $0, \alpha^{n:\tau} \xrightarrow{a} n, \tau$

(seq-cons1) $n, \alpha:\tau \xrightarrow{a} n - 1, \tau$

(seq-cons2) $n, \alpha:\tau \xrightarrow{a} n, \alpha:\tau$

(empty) $n, \lambda \xrightarrow{a} n, \lambda$

(fork-sync-l) $n_1, \tau_1 \xrightarrow{a} n_2, \tau'_1$

(fork-sync-r) $n_1, \tau_2 \xrightarrow{a} n_2, \tau'_2$
Extended transition rules

Generalization of the previous rules:

(choice-l) \[ n_1, \tau_1 \xrightarrow{a} n_2, \tau'_1 \]
\[ n_1, \tau_1 + \tau_2 \xrightarrow{a} n_2, \tau'_1 \]

(fork-l) \[ n_1, \tau_1 \xrightarrow{a} 0, \tau'_1 \]
\[ n_1, \tau_1 \| \tau_2 \xrightarrow{a} 0, \tau'_1 \| \tau_2 \]

(cat-l) \[ n_1, \tau_1 \xrightarrow{a} n_2, \tau'_1 \]
\[ n_1, \tau_1 \cdot \tau_2 \xrightarrow{a} n_2, \tau'_1 \cdot \tau_2 \]

(choice-r) \[ n_1, \tau_2 \xrightarrow{a} n_2, \tau'_2 \]
\[ n_1, \tau_1 + \tau_2 \xrightarrow{a} n_2, \tau'_2 \]

(fork-r) \[ n_1, \tau_2 \xrightarrow{a} 0, \tau'_2 \]
\[ n_1, \tau_1 \| \tau_2 \xrightarrow{a} 0, \tau_1 \| \tau'_2 \]

(cat-r) \[ n_1, \tau_2 \xrightarrow{a} n_2, \tau'_2 \]
\[ n_1, \tau_1 \cdot \tau_2 \xrightarrow{a} n_2, \tau'_2 \]
\[ \epsilon(\tau_1) \]
Alternating bit protocol (revisited)

Dimension 2

\( \text{AltBit2} = \text{M1A1} \mid \text{M2A2} \mid \text{M1M2} \)

\( \text{M1A1} = \text{msg1}^1 : \text{ack1}^0 : \text{M1A1} \)

\( \text{M2A2} = \text{msg2}^1 : \text{ack2}^0 : \text{M2A2} \)

\( \text{M1M2} = \text{msg1} : \text{msg2} : \text{M1M2} \)

Dimension 3

\( \text{AltBit3} = \text{M1A1} \mid \text{M2A2} \mid \text{M3A3} \mid \text{M1M2M3} \)

\( \text{M1A1} = \text{msg1}^1 : \text{ack1}^0 : \text{M1A1} \)

\( \text{M2A2} = \text{msg2}^1 : \text{ack2}^0 : \text{M2A2} \)

\( \text{M3A3} = \text{msg3}^1 : \text{ack3}^0 : \text{M3A3} \)

\( \text{M1M2M3} = \text{msg1} : \text{msg2} : \text{msg3} : \text{M1M2M3} \)
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
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 $\omega$-automata
- projecting global types
- from dynamic to static checking of protocol conformance
Thank you for your attention...

...questions?