Attribute Global Types for Dynamic Checking of Protocols in Logic-based Multiagent Systems

Viviana Mascardi and Davide Ancona

DIBRIS, Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
Università degli Studi di Genova, Italy

ICLP 2013,
27 of August, 2013
Multiagent systems (MASs) are an industrial-strength technology for integrating and coordinating autonomous and heterogeneous systems [JSW98]. Agents are expected to reason on what is happening in their environment and inside themselves: a logic-based approach to their specification, implementation, and verification has been often followed [KS99].
Multiagent systems (MASs) are an industrial-strength technology for integrating and coordinating autonomous and heterogeneous systems [JSW98]. Agents are expected to reason on what is happening in their environment and inside themselves: a logic-based approach to their specification, implementation, and verification has been often followed [KS99].

MASs are open, highly dynamic, and unpredictable, ensuring conformance of the agents’ actual behavior to a given interaction protocol is of paramount importance to guarantee the participants’ interoperability and security. Many approaches to this issue are based on computational logic [BBMP05,MST03,GMS07,MM06,Rob04,SC09,SOCS,TYS+09].
Contribution and background

We extended a formalism (“constrained global types”) that we recently proposed for specifying and dynamically verifying multi-party agent interaction protocols [ABM13,ADM12,AMB12]. Intuitively, a constrained global type represents the state of an interaction protocol from which several transition steps to other states (that is, to other constrained global types) are possible, with a resulting sending action.
Contribution and background

We extended a formalism ("constrained global types") that we recently proposed for specifying and dynamically verifying multi-party agent interaction protocols \[\text{ABM13,ADM12,AMB12}\]. Intuitively, a constrained global type represents the state of an interaction protocol from which several transition steps to other states (that is, to other constrained global types) are possible, with a resulting sending action.

Constrained global types are inspired by global types with synchronization constraints. Global types are behavioral types designed for specifying in a compact way multiparty interactions between distributed components, and verifying their correctness \[\text{CDP12,CHY07,HYC08}\].
Contribution and background

We extended a formalism (“constrained global types”) that we recently proposed for specifying and dynamically verifying multi-party agent interaction protocols [ABM13, ADM12, AMB12]. Intuitively, a constrained global type represents the state of an interaction protocol from which several transition steps to other states (that is, to other constrained global types) are possible, with a resulting sending action.

Constrained global types are inspired by global types with synchronization constraints. Global types are behavioral types designed for specifying in a compact way multiparty interactions between distributed components, and verifying their correctness [CDP12, CHY07, HYC08].

Their extension (this ICLP2013 work) is inspired by attribute grammar, a formal way to define attributes for the productions of a context free grammar, associating these attributes with values [Knu90].
Advantages of Attribute Global Types w.r.t. Constrained ones

Attribute global types are more expressive than constrained global types, since they allow parametric specifications of protocols.
Advantages of Attribute Global Types w.r.t. Constrained ones

Attribute global types are more expressive than constrained global types, since they allow parametric specifications of protocols.

They can be still effectively used for dynamic checking of protocols: Attribute Global Types can be easily represented as Prolog terms, and a mechanism for verifying that a sequence of messages complies to an Attribute Global Type has been designed and implemented in Prolog.
“Sending actions” and “sending action types”

Sending actions

A sending action occurs between two agents and consists of two agent identifiers (sender and receiver), the performative expressed in some ACL, and the actual content of the message. Example: msg(seller, buyer, tell, price(pasta,10)).
“Sending actions” and “sending action types”

**Sending actions**

A sending action occurs between two agents and consists of two agent identifiers (sender and receiver), the performative expressed in some ACL, and the actual content of the message. Example: `msg(seller, buyer, tell, price(pasta,10))`.

**Sending action types**

Sending actions types model which kind of message pattern is expected at a certain point of the conversation. A sending action type $\alpha$ is a monadic predicate on sending actions.
Attribute Global Types

Syntax

\[ \tau = \]

Constrained global types can be cyclic (recursive), and hence they can be represented by a finite set of syntactic equations, as happens in most modern Prolog implementations.

Example: \[ \tau = \alpha_0 : \tau \]

Semantics

The global type \( \tau \) represents a set of possibly infinite sequences of sending actions. The interpretation is based on the notion of transition, a total function \( \delta : \mathbb{N} \times T \times A \rightarrow P \text{fin}(T \times \mathbb{N}) \), where \( T \) and \( A \) denote the set of contractive and constrained global types with attributes and of sending actions, respectively.
Attribute Global Types

Syntax

\[ \tau = \lambda \mid \alpha_1 \cdot \tau_2 \mid \tau_1 + \tau_2 \mid \rho(\tau, \alpha(n)) \mid \alpha(n) : \tau \mid \alpha(n) : \tau \mid \tau \]

Constrained global types can be cyclic (recursive), and hence they can be represented by a finite set of syntactic equations, as happens in most modern Prolog implementations.

Example:

\[ \tau = \alpha_0 : \tau \]

Semantics

The global type \( \tau \) represents a set of possibly infinite sequences of sending actions. The interpretation is based on the notion of transition, a total function \( \delta : N \times T \times A \rightarrow P \text{fin}(T \times N) \), where \( T \) and \( A \) denote the set of contractive and constrained global types with attributes and of sending actions, respectively.
Attribute Global Types

Syntax

\[ \tau = \lambda \ | \ \alpha^n : \tau \ | \ ]

Constrained global types can be cyclic (recursive), and hence they can be represented by a finite set of syntactic equations, as happens in most modern Prolog implementations.

Example:

\[ \tau = \alpha_0 \]

Semantics

The global type \( \tau \) represents a set of possibly infinite sequences of sending actions. The interpretation is based on the notion of transition, a total function \( \delta : N \times T \times A \rightarrow P \) \( \text{fin}(T \times N) \), where \( T \) and \( A \) denote the set of contractive and constrained global types with attributes and of sending actions, respectively.
### Syntax

\[ \tau = \lambda \mid \alpha^n : \tau \mid \alpha : \tau \]
## Attribute Global Types

### Syntax

\[
\tau = \lambda \quad | \quad \alpha^n : \tau \quad | \quad \alpha : \tau \quad | \quad \tau_1 + \tau_2
\]
Attribute Global Types

Syntax

\[ \tau = \lambda \mid \alpha^n \cdot \tau \mid \alpha : \tau \mid \tau_1 + \tau_2 \mid \tau_1 | \tau_2 \]

Constrained global types can be cyclic (recursive), and hence they can be represented by a finite set of syntactic equations, as happens in most modern Prolog implementations.

Example:

\[ \tau = \alpha \]

Semantics

The global type \( \tau \) represents a set of possibly infinite sequences of sending actions. The interpretation is based on the notion of transition, a total function \( \delta : N \times T \times A \rightarrow \mathbb{P}(T \times N) \), where \( T \) and \( A \) denote the set of contractive and constrained global types with attributes and of sending actions, respectively.
### Syntax

\[
\tau = \lambda \mid \alpha^n : \tau \mid \alpha : \tau \mid \tau_1 + \tau_2 \mid \tau_1 \cdot \tau_2
\]

Constrained global types can be cyclic (recursive), and hence they can be represented by a finite set of syntactic equations, as happens in most modern Prolog implementations.

Example:

\[
\tau = \alpha_0 : \tau
\]

Semantics

The global type \(\tau\) represents a set of possibly infinite sequences of sending actions. The interpretation is based on the notion of transition, a total function \(\delta : N \times T \times A \rightarrow P_{\text{fin}}(T \times N)\), where \(T\) and \(A\) denote the set of contractive and constrained global types with attributes and of sending actions, respectively.
Attribute Global Types

Syntax

\[ \tau = \lambda \mid \alpha^n : \tau \mid \alpha : \tau \mid \tau_1 + \tau_2 \mid \tau_1 \cdot \tau_2 \mid fc(\tau, cn, n) \]

Constrained global types can be cyclic (recursive), and hence they can be represented by a finite set of syntactic equations, as happens in most modern Prolog implementations.

Example:

\[ \tau = \alpha_0 : \tau \]

Semantics

The global type \( \tau \) represents a set of possibly infinite sequences of sending actions. The interpretation is based on the notion of transition, a total function \( \delta : N \times T \times A \rightarrow P(\mathrm{fin}(T \times N)) \), where \( T \) and \( A \) denote the set of contractive and constrained global types with attributes and of sending actions, respectively.
Attribute Global Types

Syntax

\[ \tau = \lambda \ | \ \alpha^n : \tau \ | \ \alpha : \tau \ | \ \tau_1 + \tau_2 \ | \ \tau_1 \cdot \tau_2 \ | \ fc(\tau, cn, n) \ | \ \tau(\text{attr}_\tau)[c] \]
## Attribute Global Types

### Syntax

\[
\tau = \lambda \mid \alpha^n : \tau \mid \alpha : \tau \mid \tau_1 + \tau_2 \mid \tau_1 \cdot \tau_2 \mid fc(\tau, cn, n) \mid \tau(\text{attr}_\tau)[c]
\]

Constrained global types can be cyclic (recursive), and hence they can be represented by a finite set of syntactic equations, as happens in most modern Prolog implementations.

Example: \( \tau = \alpha^0 : \tau \)
Attribute Global Types

Syntax

\[ \tau = \lambda | \alpha^n: \tau | \alpha: \tau | \tau_1 + \tau_2 | \tau_1 \cdot \tau_2 | fc(\tau, cn, n) | \tau(\text{attr}_\tau)[c] \]

Constrained global types can be cyclic (recursive), and hence they can be represented by a finite set of syntactic equations, as happens in most modern Prolog implementations.
Example: \( \tau = \alpha^0: \tau \)

Semantics

The global type \( \tau \) represents a set of possibly infinite sequences of sending actions.
The interpretation is based on the notion of transition, a total function \( \delta: \mathbb{N} \times \mathcal{T} \times \mathcal{A} \rightarrow \mathcal{P}_{\text{fin}}(\mathcal{T} \times \mathbb{N}) \), where \( \mathcal{T} \) and \( \mathcal{A} \) denote the set of contractive and constrained global types with attributes and of sending actions, respectively.
ABP with 4 messages

\[
\begin{align*}
ABP_2 &= (MA_1 | MA_2) | MM_2 \\
MA_1 &= msg_1^1 : ack_1^0 : MA_1 \\
MA_2 &= msg_2^1 : ack_2^0 : MA_2 \\
MM_2 &= msg_1 : msg_2 : MM_2 \\
(Alice, Bob, tell, m(1)) &\in msg_1 \quad (Bob, Alice, tell, a(1)) \in ack_1 \\
(Alice, Bob, tell, m(2)) &\in msg_2 \quad (Bob, Alice, tell, a(2)) \in ack_2
\end{align*}
\]
ABP with 20 messages

\[
\begin{align*}
ABP_{10} &= (MA_1|MA_2|MA_3|MA_4|MA_5|MA_6|MA_7|MA_8|MA_9|MA_{10})|MM_{10} \\
MA_1 &= \text{msg}_1:\text{ack}_0:MA_1 \\
MA_2 &= \text{msg}_2:\text{ack}_0:MA_2 \\
MA_3 &= \text{msg}_3:\text{ack}_0:MA_3 \\
MA_4 &= \text{msg}_4:\text{ack}_0:MA_4 \\
MA_5 &= \text{msg}_5:\text{ack}_0:MA_5 \\
MA_6 &= \text{msg}_6:\text{ack}_0:MA_6 \\
MA_7 &= \text{msg}_7:\text{ack}_0:MA_7 \\
MA_8 &= \text{msg}_8:\text{ack}_0:MA_8 \\
MA_9 &= \text{msg}_9:\text{ack}_0:MA_9 \\
MA_{10} &= \text{msg}_{10}:\text{ack}_{10}:MA_{10} \\
MM_{10} &= \text{msg}_1:\text{msg}_2:\text{msg}_3:\text{msg}_4:\text{msg}_5:\text{msg}_6:\text{msg}_7:\text{msg}_8:\text{msg}_9:\text{msg}_{10}:MM_{10} \\
(Alice, Bob, tell, m(1)) &\in \text{msg}_1 \\
(Alice, Bob, tell, m(2)) &\in \text{msg}_2 \\
(Alice, Bob, tell, m(3)) &\in \text{msg}_3 \\
(Alice, Bob, tell, m(4)) &\in \text{msg}_4 \\
(Alice, Bob, tell, m(5)) &\in \text{msg}_5 \\
(Alice, Bob, tell, m(6)) &\in \text{msg}_6 \\
(Alice, Bob, tell, m(7)) &\in \text{msg}_7 \\
(Alice, Bob, tell, m(8)) &\in \text{msg}_8 \\
(Alice, Bob, tell, m(9)) &\in \text{msg}_9 \\
(Alice, Bob, tell, m(10)) &\in \text{msg}_{10} \\
(Bob, Alice, tell, a(1)) &\in \text{ack}_1 \\
(Bob, Alice, tell, a(2)) &\in \text{ack}_2 \\
(Bob, Alice, tell, a(3)) &\in \text{ack}_3 \\
(Bob, Alice, tell, a(4)) &\in \text{ack}_4 \\
(Bob, Alice, tell, a(5)) &\in \text{ack}_5 \\
(Bob, Alice, tell, a(6)) &\in \text{ack}_6 \\
(Bob, Alice, tell, a(7)) &\in \text{ack}_7 \\
(Bob, Alice, tell, a(8)) &\in \text{ack}_8 \\
(Bob, Alice, tell, a(9)) &\in \text{ack}_9 \\
(Bob, Alice, tell, a(10)) &\in \text{ack}_{10}
\end{align*}
\]
ABP with k messages

\[ ABP'(k) = f_c(MA(\_), \vert, k) \mid MM'(k, 0, \_)[mod(k, p, c)] = msg(c):MM'(k, c, \_)[mod(k, p, c)] \]

\[ MA(i) = msg^1(i)\!:ack^0(\_):MA(i) \]

\[ (Alice, Bob, tell, m(I)) \in msg(I) \quad (Bob, Alice, tell, a(I)) \in ack(I) \]

The constraint \([mod(k, p, c)]\) is a shortcut for the boolean condition

\[ k : Nat \land p : Nat \land c : Nat \land p \in [0..k] \land c \in [1..k] \land (p < k \Rightarrow c = p + 1) \land (p = k \Rightarrow c = 1) \]
Dynamic checking of conformance to protocols

Prolog clauses implementing the transition function

```prolog
fc(T, T1, C, N) :- N>1, copy_term(T1, Fresh), N1 is N-1,
fc(T2, T1, C, N1), T =.. [C, Fresh, T2].
fc(Fresh, T, _, 1) :- copy_term(T, Fresh).

/* seq-prod */ next(0, (AType,N):T,AMsg,T,N) :- has_type(AMsg, AType),!.

/* seq-cons */ next(N, AType:T1,AMsg,T1,M) :-
    has_type(AMsg, AType), !, N > 0, M is N - 1.
/* fork-both-l */ next(N,T1|T2,A,T3|T4,P) :-
    next(N,T1,A,T3,M), M > 0, next(M,T2,A,T4,P).
/* fork-both-r */ next(N,T1|T2,A,T4|T3,P) :-
    !, next(N,T2,A,T3,M), M > 0, next(M,T1,A,T4,P).
/* fork-l */ next(N,T1|T2,A,T3|T2,M) :- next(N,T1,A,T3,M).
/* fork-r */ next(N,T1|T2,A,T1|T3,M) :- next(N,T2,A,T3,M).
/* choice-l */ next(N,T1+,A,T2,M) :- next(N,T1,A,T2,M).
/* choice-r */ next(N,+,T1,A,T2,M) :- !, next(N,T1,A,T2,M).
/* cat-l */ next(N,T1*T2,A,T3*T2,M) :- next(N,T1,A,T3,M).
/* cat-r */ next(N,T1*T2,A,T3,M) :- !, empty(T1), next(N,T2,A,T3,M).
/* attr */ next(M, attrType(Name, Constr, T1), A, T2, N) :-
    !, attrType(Name, Constr, T1), next(M, T1, A, T2, N), call(Constr).
```

/* Goal to prove:
next(0, LastState, msg(S, R, P, C), NewState, 0) */
Monitoring conversations in logic-based MASs

The logic-based representation and implementation allow us to integrate a monitor agent implementing a run-time verification mechanism of protocol compliance into any logic-based agent oriented programming language that supports the basic Prolog built-ins. We experimented our approach with Jason.
Monitoring conversations in logic-based MASs

The logic-based representation and implementation allow us to integrate a monitor agent implementing a run-time verification mechanism of protocol compliance into any logic-based agent oriented programming language that supports the basic Prolog built-ins. We experimented our approach with Jason.

Monitoring a Jason MAS compliant (or not...) to the FIPA Iterated Contract Net Protocol consisting of more than 50 participants exchanging 22 messages in each conversation with the initiator required less than one minute on an Acer TravelMate 6293 with Intel Core 2 Duo P8400/2.26 GHz (Dual-Core) processor and 4 GB RAM.
Monitor agent in Jason (1)

V. Mascardi, D. Ancona (DIBRIS, Genova)

Attribute Global Types for MASs

ICLP 2013, 27-08-2013
Monitor agent in Jason (2)

V. Mascardi, D. Ancona (DIBRIS, Genova)
Questions???

Thank you for your attention!
Logic-based MASs and Interaction Verification (1)


Logic-based MASs and Interaction Verification (2)


Global Types and Attribute Grammars


Constrained Global Types for MASs (our previous work)


The global type $\tau$ represents a set of possibly infinite sequences of sending actions. The basic constructors are:
Constrained Global Types for MASs

The global type \( \tau \) represents a set of possibly infinite sequences of sending actions. The basic constructors are:

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

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

- \( \alpha \cdot \tau \cdot \text{seq-cons} \): a consumer of sending action \( a \) matching type \( \alpha \) \((a \in \alpha)\), and followed by any sequence in the set represented by \( \tau \).

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

- \( \tau_1 | \tau_2 \) (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 \( \tau_2 \).
The global type $\tau$ represents a set of possibly infinite sequences of sending actions. The basic constructors are:

- $\lambda$ (empty sequence), representing the singleton set $\{\epsilon\}$ containing the empty sequence $\epsilon$.
- $\alpha^n: \tau$ (seq-prod): the set of all sequences whose first element is a sending action $a$ matching type $\alpha$ ($a \in \alpha$), and the remaining part is a sequence in the set represented by $\tau$. 
Constrained Global Types for MASs

The global type $\tau$ represents a set of possibly infinite sequences of sending actions. The basic constructors are:

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

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

- $\alpha:\tau$ (*seq-cons*): a consumer of sending action $a$ matching type $\alpha$ ($a \in \alpha$), and followed by any sequence in the set represented by $\tau$. 

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

- $\tau_1 | \tau_2$ (*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 $\tau_2$. 


V. Mascardi, D. Ancona (DIBRIS, Genova)

Attribute Global Types for MASs

ICLP 2013, 27-08-2013
Constrained Global Types for MASs

The global type $\tau$ represents a set of possibly infinite sequences of sending actions. The basic constructors are:

- $\lambda$ (empty sequence), representing the singleton set $\{\epsilon\}$ containing the empty sequence $\epsilon$.
- $\alpha^n:\tau$ (seq-prod): the set of all sequences whose first element is a sending action $a$ matching type $\alpha$ ($a \in \alpha$), and the remaining part is a sequence in the set represented by $\tau$.
- $\alpha:\tau$ (seq-cons): a consumer of sending action $a$ matching type $\alpha$ ($a \in \alpha$), and followed by any sequence in the set represented by $\tau$.
- $\tau_1 + \tau_2$ (choice), representing the union of the sequences of $\tau_1$ and $\tau_2$. 
Constrained Global Types for MASs

The global type $\tau$ represents a set of possibly infinite sequences of sending actions. The basic constructors are:

- $\lambda$ (empty sequence), representing the singleton set $\{e\}$ containing the empty sequence $e$.
- $\alpha^n : \tau$ (seq-prod): the set of all sequences whose first element is a sending action $a$ matching type $\alpha$ ($a \in \alpha$), and the remaining part is a sequence in the set represented by $\tau$.
- $\alpha : \tau$ (seq-cons): a consumer of sending action $a$ matching type $\alpha$ ($a \in \alpha$), and followed by any 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 $\tau_2$. 
Constrained Global Types for MASs

The global type $\tau$ represents a set of possibly infinite sequences of sending actions. The basic constructors are:

- $\lambda$ (empty sequence), representing the singleton set $\{\epsilon\}$ containing the empty sequence $\epsilon$.
- $\alpha^n: \tau$ (seq-prod): the set of all sequences whose first element is a sending action $a$ matching type $\alpha$ ($a \in \alpha$), and the remaining part is a sequence in the set represented by $\tau$.
- $\alpha: \tau$ (seq-cons): a consumer of sending action $a$ matching type $\alpha$ ($a \in \alpha$), and followed by any sequence in the set represented by $\tau$.
- $\tau_1 + \tau_2$ (choice), representing the union of the sequences of $\tau_1$ and $\tau_2$.
- $\tau_1 | \tau_2$ (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 $\tau_2$. 
The “meta-construct” $fc$ (for \textit{finite composition}) that takes $\tau$, a constructor $cn$, and a positive natural number $n$ as inputs and generates the “normal” constrained global type $(\tau \; cn \; \tau \; cn \; \ldots \; cn \; \tau \; cn \; \lambda)$ ($n$ times).
Composition constructor and attributes

- The “meta-construct” $fc$ (for finite composition) that takes $\tau$, a constructor $cn$, and a positive natural number $n$ as inputs and generates the “normal” constrained global type $(\tau cn \tau cn \ldots cn \tau cn \lambda)$ ($n$ times).

- Attribute global types are constrained global types whose sending action types may have attributes, included within curly brackets, and that have been extended to provide contextual information by means of further attributes and conditions, included in square brackets.