

# Verifying Agent Conformance with Protocols: an Automata Based Approach <sup>\*</sup>

Laura Giordano<sup>1</sup>, Alberto Martelli<sup>2</sup>

<sup>1</sup>Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria

<sup>2</sup>Dipartimento di Informatica, Università di Torino, Torino

**Abstract.** The paper addresses the problem of agents compatibility and their conformance to protocols. We assume that the specification of protocols is given in an action theory by means of temporal constraints and, in particular, communicative actions are defined in terms of their effects and preconditions on the social state of the protocol. In this framework, we can deal both with terminating and infinite protocols (reactive services). We show that the problem of verifying the conformance of an agent with a protocol can be solved by making use of an automata based approach, and that the conformance of a set of agents with a protocol guarantees that their interaction cannot produce deadlock situations and it only gives rise to runs of the protocol.

## 1 Introduction

One of the central problems in the area of multi-agent systems, as well as in the area of web services, concerns the interoperability of software agents in an open environment. Agents are usually loosely coupled, as they are written by different organizations, and their interoperability with other agents cannot be guaranteed a-priori. This has raised the problem of introducing conditions which enforce that a set of agents can interact properly, thus leading to the introduction of different notions of *compatibility* among agents [5] as well as to the definition of notions of *conformance* of an agent with a protocol [13, 3, 6]. The fact that an agent conforms with a protocol must guarantee that all the interactions of the agent with other conformant agents are correct: they produce executions of the protocol and do not lead to deadlock situations.

In our proposal, the interaction protocol which rules the communications among agents is specified in an action theory based on a temporal logic, namely dynamic linear time temporal logic (DLTL) [11]. In this framework, as described in the next section, protocols are given a *declarative specification* consisting of: (i) specification of communicative actions by means of their effects and preconditions on the social state which, in particular, includes commitments; (ii) a set of temporal constraints, which specify the wanted interactions (under this respect, our approach to protocol specification is similar to the one proposed in

---

<sup>\*</sup> This research has been partially supported by the project PRIN 2005 “Specification and verification of agent interaction protocols”

*DecSerFlow* [1]). Protocols with nonterminating computations, modeling reactive services [7], can also be captured in this framework. Communication among agents is assumed to be synchronous and, in this concern, we diverge from [7] and [13], where asynchronous message passing is considered.

In [9, 10], we have shown that several verification problems can be modelled as satisfiability and validity problems in the logic, by making use of an automata based approach and, in particular, by working on the Büchi automaton which can be extracted from the logical specification of the protocol. In this paper we make use of the protocol automaton in the verification of the conformance of an agent with a protocol.

In Section 3, we introduce a notion of conformance of an agent with a protocol by comparing the runs of the agent and the runs of protocol. Then, in Section 4, we define an algorithm which verifies the conformance of an agent with a protocol, assuming that both the agent and the protocol are represented as Büchi automata. The protocol automaton obtained from the temporal specification is a general (non deterministic) Büchi automaton, while agents are assumed to be modelled by deterministic Büchi automata. The non deterministic behaviors of agents can however be modelled through the “non deterministic choice” among different actions.

In the general case, the verification algorithm works in exponential time in the size of the automata. However, when the protocol automaton is deterministic, conformance can be checked in polynomial time.

For lack of space, in the paper we only address in detail the case of protocols with two participants. We shortly discuss the general problem in the conclusions.

## 2 Protocol Specification

The specification of interaction protocols [9, 10] is based on Dynamic Linear Time Temporal Logic (DLTL) [11], a linear time temporal logic which extends LTL by allowing the until operator to be indexed by programs in Propositional Dynamic Logic (PDL) as follows:  $\alpha \mathcal{U}^\pi \beta$ , where  $\pi$  is a program (a regular expression), built from a set  $\Sigma$  of atomic actions.

As for LTL, DLTL models are infinite linear sequences of worlds (propositional interpretations), each one reachable from the initial one by a finite sequence  $\tau$  of actions in  $\Sigma$ . A valuation function  $V$ , defines the interpretation of propositions at each world  $\tau$ .

A formula  $\alpha \mathcal{U}^\pi \beta$  is true at a world  $\tau$  if “ $\alpha$  until  $\beta$ ” is true on a finite stretch of behavior which is in the linear time behavior of the program  $\pi$ . The derived modalities  $\langle \pi \rangle \alpha$  and  $[\pi] \alpha$  can be defined as follows:  $\langle \pi \rangle \alpha \equiv \top \mathcal{U}^\pi \alpha$  and  $[\pi] \alpha \equiv \neg \langle \pi \rangle \neg \alpha$ . When  $\pi$  is  $\Sigma^*$ , we replace  $\langle \pi \rangle$  with  $\diamond$  and  $[\pi]$  with  $\square$ . As shown in [11], DLTL( $\Sigma$ ) is strictly more expressive than LTL( $\Sigma$ ). The satisfiability and validity problems for DLTL are PSPACE complete problems [11].

We illustrate how a protocol can be specified in this framework through the specification of a *Purchase protocol*. We have two roles: the merchant (mr) and the customer (ct). The customer sends a request to the merchant, the merchant

replies with an offer or by saying that the requested good is not available, and, if the customer receives the offer, it may accept or refuse it. If the customer accepts the offer, then the merchant delivers the goods. After receiving the goods, the customer sends the payment.

The two agents share all the communicative actions, which are: *sendOffer*, *sendNotAvail*, *sendGoods* whose sender is the merchant; *sendRequest*, *sendAccept*, *sendRefuse*, *sendPayment*, whose sender is the customer. Communication is synchronous: the agents communicate by synchronizing on the execution of communicative actions.

The Purchase protocol  $Pu$  is specified by a *domain description*  $D_{Pu}$ , which is a pair  $(\Pi, \mathcal{C})$ , where  $\Pi$  is a set of formulas describing the action theory, and  $\mathcal{C}$  is a set of *constraints*.

We adopt a social approach where an interaction protocol is specified by describing the effects of communicative actions on the social state. The social state contains the domain specific fluents describing observable facts concerning the execution of the protocol. Examples of fluents are: *requested* (the customer has requested a quote), *accepted* (the customer has accepted the quote), *goods* (the merchant has sent the goods). Also special fluents are introduced to model *commitments* among the agents [15]:  $C(i, j, \alpha)$ , means that agent  $i$  is committed to agent  $j$  to bring about  $\alpha$ . Furthermore, a *conditional commitments*  $CC(i, j, \beta, \alpha)$  means that agent  $i$  is committed to agent  $j$  to bring about  $\alpha$ , if the condition  $\beta$  is brought about.

The action theory  $\Pi$  consists of *action laws*, *causal laws*, *precondition laws*, and an *initial state*.

*Action laws*  $\mathcal{AL}$  in  $\Pi$  have the form:  $\Box(\alpha \rightarrow [a]l)$ , with  $a \in \Sigma$ ,  $l$  a fluent literal<sup>1</sup> and  $\alpha$  a conjunction of literals, meaning that executing action  $a$  in a state where precondition  $\alpha$  holds causes the effect  $l$  to hold.

Although the framework has been extended in [12] to model incomplete information by including epistemic operators, for simplicity here we do not consider the epistemic extension of the formalism. Also, actions are assumed to be *deterministic*, i.e. executing an action in a state gives a unique successor state.

Some of the effects of communicative actions in protocol  $Pu$  are the following:

$$\Box([sendOffer]CC(mr, ct, accepted, goods))$$

when the merchant sends the quote for the good, then he commits to send the goods if the customer accepts the request,

$$\Box(requested \rightarrow [sendOffer]\neg requested)$$

when the merchant sends the quote for the good, if there is a request, the request is cancelled.

*Causal laws*  $\mathcal{CL}$  in  $\Pi$  have the form:  $\Box((\alpha \wedge \bigcirc\beta) \rightarrow \bigcirc l)$  (where  $l$  a fluent literal and  $\alpha, \beta$  conjunctions of literals), meaning that if  $\alpha$  holds in a state and  $\beta$  holds in the next state, then  $l$  also holds in the next state. Such laws are intended to express “causal” dependencies among fluents. For instance, the *causal law*:

$$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg C(i, j, \alpha))$$

---

<sup>1</sup> A *fluent literal*  $l$  stands for a fluent name  $f$  or its negation  $\neg f$ .

says that a commitment to bring about  $\alpha$  is cancelled when  $\alpha$  holds. Other causal laws are needed for dealing with conditional commitments.

*Precondition laws*  $\mathcal{PL}$  have the form:  $\Box(\alpha \rightarrow [a]\perp)$ , meaning that the execution of an action  $a$  is not possible if  $\alpha$  holds (i.e. there is no resulting state following the execution of  $a$  if  $\alpha$  holds). The *precondition laws* for the actions of the customer are the following ones:

$$\begin{aligned} &\Box(\neg offer \rightarrow [sendAccept]\perp) \\ &\Box(\neg offer \rightarrow [sendRefuse]\perp) \\ &\Box(\neg goods \rightarrow [sendPayment]\perp). \end{aligned}$$

meaning that: the customer may send an accept or refuse only if an offer has been done. The customer may send a payment for the goods only if he has received the goods (all other actions are always executable for the customer).

The *initial state*  $\mathcal{IS}$  of the protocol defines the initial value of all the fluents. Here, differently from [12], we assume that the initial state is complete.

Action laws and causal laws describe the changes to the state. All other fluents which are not changed by the actions are assumed to persist unaltered to the next state. To cope with the *frame problem* [14] we use a completion construction *Comp*, which is applied to the action laws and to the causal laws [10]. Thus  $\Pi$  is defined as:

$$\Pi = Comp(\mathcal{AL} \wedge \mathcal{CL}) \wedge \mathcal{PL} \wedge \mathcal{IS}$$

The second component  $\mathcal{C}$  of the domain description  $D_{Pu}$  defines constraints as arbitrary temporal formulas of DLTL. For instance, to model the fact that, for each request, the customer answers only once sending an Offer or NotAvail, we introduce the following constraint:

$$\neg \diamond \langle sendOffer + sendNotAvail \rangle \diamond \langle sendOffer + sendNotAvail \rangle \top$$

where  $+$  is the nondeterministic choice among actions. It is not possible a *sendOffer* or *sendNotAvail* followed, eventually, by another *sendOffer* or *sendNotAvail*.

We are interested in those execution of the purchase protocol in which all commitments have been fulfilled. Hence, we add, for each commitment  $C(i, j, \alpha)$  the constraint:

$$\Box(C(i, j, \alpha) \rightarrow \diamond \alpha).$$

Given the domain description  $D_{Pu} = (\Pi, \mathcal{C})$  of Purchase protocol, the *runs* of the protocol are the linear models of  $\Pi \wedge \mathcal{C}$ .

Note that protocol “runs” are always infinite, as logic DLTL is characterized by infinite models. When we want to model terminating protocols, as the one above, we assume the domain description of the protocol to be suitably extended with an action *noop* which does nothing and which can be executed forever after termination of the protocol.

Once the specification of a protocol has been given, several kinds of verification can be performed on it, including the verification of properties of the protocol per-se, as well as the verification that a set agents are compliant with a given interaction protocol at runtime. We refer to [9, 10] for a description of these

verification problems which can be modelled as satisfiability or validity problems in the temporal logic.

### 3 Conformance

Given a protocol  $P$  with two roles  $i$  and  $j$ , and an agent  $S_i$  (playing the role of  $i$ ), we want to define a notion of conformance of  $S_i$  with the protocol  $P$  which guarantees that the interactions of  $S_i$  with any other conformant agent  $S_j$  gives rise to legal runs of the protocol and it does not produce deadlock situations.

Consider for instance a customer agent  $S_{ct}$  whose behavior differs from that of the role “customer” of protocol  $P_u$  as follows: whenever it receives an offer from the merchant, it always accepts it; after accepting the offer it expects to receive from the merchant either the goods or a warning that the delivery has been cancelled.

Although the behavior of the customer agent and that of the corresponding role of the protocol are different, we can consider however the agent to be compliant with the protocol, according to the following observations. The customer, is not forced to send all the messages that could be sent according to the protocol. For instance, it can always accept an offer, and never send the message *sendRefuse*. On the other hand, an agent can receive more messages than those it should actually receive according to the protocol (an agent can serve more requests than expected from the protocol). The customer  $S_{ct}$  can also receive the message that the delivery has been cancelled, even if no merchant conformant with the protocol will ever send it (for further comments on this notion of conformance see [5, 3]). Informally, a protocol  $S_i$  conforms with a protocol  $P$  if the following conditions hold:

- (i) The messages sent from  $S_i$  are *correct*: that is, if  $S_i$  sends a message  $m$  at some stage, then, the role  $i$  of the protocol can send message  $m$  at that stage.
- (ii)  $S_i$  must receive all the messages which it could receive according to the protocol. This is a *completeness* requirement for  $S_i$ .
- (iii) If, in a state of the protocol, role  $i$  is expected to send a message, in the corresponding state of agent  $S_i$ , it must send at least a message. This condition is required to avoid deadlock situations when the two agents  $S_i$  and  $S_j$  interact: they cannot be both waiting to receive a message.

The third condition is needed because our notion of conformance includes also interoperability of interacting agents.

Notice again that we are considering in two different ways the nondeterministic choices concerning emissions (the customer can accept or refuse an offer) and those concerning receptions (the customer receives the messages *sendOffer* or *sendNotAvail*). As usual in agent applications, we assume that, in the first case, the choice is internal to the agent (internal non determinism), while, in the second case, the choice is of the partner agents, here, the merchant (external non

determinism). We refer to [13] for a distinction between internal and external non-determinism.

In the following, we define a notion of conformance of an agent with respect to a protocol by comparing the runs of the agents and the runs of the protocol. In particular, in this definition, we will not consider the value of fluents at the different worlds in the runs, but only the sequences of actions that can be executed according to the protocol and to the agent.

Let us consider a protocol  $P$  involving two roles,  $i$  and  $j$ .

**Definition 1.** *An agent  $S_i$  is conformant with a protocol  $P$  if, whenever there are two runs,  $\sigma_S$  of  $S_i$  and  $\sigma_P$  of  $P$ , with a common prefix  $\pi$ , the following conditions are satisfied:*

- (1) *if the action  $send_{i,j}$  is executed after the prefix  $\pi$  in  $\sigma_S$ , then there exist a run  $\sigma$  common to  $P$  and  $S_i$  with prefix  $\pi send_{i,j}$ ;*
- (2) *if the action  $send_{j,i}$  is executed after the prefix  $\pi$  in  $\sigma_P$ , then there is a run  $\sigma$  common to  $P$  and  $S_i$  with prefix  $\pi send_{j,i}$ ;*
- (3) *if the action  $send_{i,j}$  is executed after the prefix  $\pi$  in  $\sigma_P$ , then there is a run  $\sigma$  common to  $P$  and  $S_i$  with prefix  $\pi send'_{i,j}$ , with  $send'_{i,j}$  possibly different from  $send_{i,j}$ .*

Item (1) says that the messages sent by agent  $S_i$  are correct. It corresponds to condition (i). Item (2) says that  $S_i$  receives all the messages he can get according to its role, and corresponds to condition (ii). Finally, item (3) corresponds to condition (iii).

We can prove that if two agents  $S_i$  and  $S_j$  are conformant with respect to a protocol  $P$ , then the interaction of  $S_i$  and  $S_j$  cannot produce deadlock situations and it only gives rise to runs of the protocol  $P$ .

**Theorem 1.** *Let  $P$  be a protocol with a nonempty set of runs. Let  $S_i$  and  $S_j$  be two agents that are conformant with  $P$ . If there are two runs  $\sigma_i$  of  $S_i$  and  $\sigma_j$  of  $S_j$  that have a common prefix  $\pi$ , then there are two runs  $\sigma'_i$  of  $S_i$  and  $\sigma'_j$  of  $S_j$ , that have a common prefix  $\pi a$ , for some action  $a$ . Moreover,  $\pi a$  is a prefix of a run of  $P$ .*

*Proof.* Let  $\pi$  be the prefix common to  $\sigma_i$  and  $\sigma_j$ . First, we show that there is a run of  $P$  with prefix  $\pi$ . We prove it by induction on the length  $l$  of the prefix  $\pi$ . For  $l = 1$ , let  $\pi = a$ . Let us assume that action  $a$  is  $send_{i,j}$ . As there is a run of  $P$  with a common prefix  $\epsilon$  with  $\sigma_i$ , by the conformance of  $S_i$  with  $P$ , case (1), there is a run  $\sigma$  common to  $P$  and  $S_i$  with prefix  $send_{i,j}$ . In case action  $a$  is  $send_{j,i}$ , we proceed similarly, using conformance of  $S_j$  with  $P$ . For the inductive case  $l + 1$ , let  $\pi = \pi' a$ . By inductive hypothesis we know that there is a run of  $P$  starting with  $\pi'$ . From the hypothesis, the runs  $\sigma_i$  of  $S_i$  and  $\sigma_j$  of  $S_j$  have a common prefix  $\pi$ , and hence  $\pi'$ . Let us assume that  $a = send_{i,j}$ . Then, by the conformance of  $S_i$  with  $P$ , case (1), there is a run  $\sigma$  common to  $P$  and  $S_i$  with prefix  $\pi' send_{i,j}$  of length  $l + 1$ . Hence, there is a run of  $P$  with prefix  $\pi$ . For  $a = send_{j,i}$ , we proceed similarly.

We can now prove the thesis. Let us now consider, for a given  $\pi$ , the different actions that can be executed in  $\sigma_i$  and  $\sigma_j$  after  $\pi$ .

*Case 1.* Assume that in  $\sigma_i$  action  $send_{i,j}$  is executed after  $\pi$ . As there is a run of  $P$  starting with  $\pi$ , by the conformance of  $S_i$  with  $P$ , case (1), there is a run  $\sigma'_i$  common to  $P$  and  $S_i$  with prefix  $\pi send_{i,j}$ . By the conformance of  $S_j$  to  $P$ , case (2), there is a run  $\sigma'_j$  common to  $P$  and  $S_j$  with prefix  $\pi send_{i,j}$ . And the thesis follows.

*Case 2.* If in  $\sigma_j$  action  $send_{j,i}$  is executed after  $\pi$ , the proof is as in case 1.

*Case 3.* Assume that in  $\sigma_i$  action  $send_{j,i}$  is executed after  $\pi$  and in  $\sigma_i$  action  $send_{i,j}$  is executed after  $\pi$  (both  $S_i$  and  $S_j$  execute a receive after  $\pi$ ). As there is a run  $\sigma_P$  of  $P$  with prefix  $\pi$ , let  $send_{i,j}$  be the action executed on  $\sigma_P$  after  $\pi$  (in case the action is  $send_{j,i}$ , we proceed similarly). Then, by the conformance of  $S_i$ , case (3), there is a run  $\sigma'_i$  common to  $P$  and  $S_i$  with prefix  $\pi send'_{i,j}$ . By the conformance of  $S_j$ , case (2), there is a run  $\sigma'_j$  common to  $P$  and  $S_j$  with prefix  $\pi send'_{i,j}$ . And the thesis follows.

**Corollary 1.** *Let  $S$  and  $J$  be two agents that are conformant with  $P$ . The interaction of  $S$  and  $P$  does not produce deadlock situations and it only produces executions of the protocol  $P$ .*

The proof is omitted for lack of space.

## 4 Verifying the Conformance of an Agent with a Protocol

### 4.1 Reasoning about protocols using automata

Verification and satisfiability problems can be solved by extending the standard approach for verification of Linear Time Temporal Logic, based on the use of Büchi automata. We recall that a *Büchi automaton* has the same structure as a traditional finite state automaton, with the difference that it accepts infinite words. More precisely a Büchi automaton over an alphabet  $\Sigma$  is a tuple  $\mathcal{B} = (Q, \rightarrow, Q_{in}, F)$  where:

- $Q$  is a finite nonempty set of states;
- $\rightarrow \subseteq Q \times \Sigma \times Q$  is a transition relation;
- $Q_{in} \subseteq Q$  is the set of initial states;
- $F \subseteq Q$  is a set of accepting states.

Let  $\sigma \in \Sigma^\omega$ . Then a run of  $\mathcal{B}$  over  $\sigma$  is a map  $\rho : prf(\sigma) \rightarrow Q$  such that:

- $\rho(\varepsilon) \in Q_{in}$
- $\rho(\tau) \xrightarrow{a} \rho(\tau a)$  for each  $\tau a \in prf(\sigma)$

The run  $\rho$  is *accepting* iff  $inf(\rho) \cap F \neq \emptyset$ , where  $inf(\rho) \subseteq Q$  is given by  $q \in inf(\rho)$  iff  $\rho(\tau) = q$  for infinitely many  $\tau \in prf(\sigma)$ .

As described in [11], the satisfiability problem for DLTL can be solved in deterministic exponential time, as for LTL, by constructing for each formula  $\alpha \in$

$DLTL(\Sigma)$  a Büchi automaton  $\mathcal{B}_\alpha$  such that the language of  $\omega$ -words accepted by  $\mathcal{B}_\alpha$  is non-empty if and only if  $\alpha$  is satisfiable.

The construction given in [11] is highly inefficient since it requires to build an automaton with an exponential number of states, most of which will not be reachable from the initial state. A more efficient approach for constructing on-the-fly a Büchi automaton from a DLTL formula has been proposed in [8], by generalizing the tableau-based algorithm for LTL. Given a formula  $\varphi$ , the algorithm builds a *labelled* Büchi automaton, i.e. a Büchi automaton extended with a *labeling function*  $\mathcal{L} : S \rightarrow 2^F$ , which associates a set of fluents with each state. Given an accepting run of the automaton, a model of the given formula  $\varphi$  can be obtained by completing the label of each state of the run in a consistent way.

For a given a domain description  $\Pi \wedge \mathcal{C}$  specifying a protocol, the above algorithm can be used to construct the corresponding labelled Büchi automaton, such that all runs accepted by the automaton represent runs of the protocol.

Here, we adopt a technique similar to the one adopted for *model checking*, i.e. by building separately the two Büchi automata corresponding to  $\Pi$  and  $\mathcal{C}$  and by making their synchronous product. The Büchi automaton for  $\mathcal{C}$  can be constructed with the general algorithm mentioned above. In the following we will call this non-deterministic automaton  $\mathcal{M}_\mathcal{C}$ .

Instead, the Büchi automaton corresponding to  $\Pi$  can be easily obtained by means of a more efficient technique, exploiting the fact that in our action theory we assume to have *complete states* and *deterministic actions*. We can obtain from the domain description a function  $next\_state_a(S)$ , for each action  $a$ , for transforming a state to the next one, and then build the automaton by repeatedly applying these functions to all states where the preconditions of the action hold, starting from the initial state. In the following we will call this deterministic automaton  $\mathcal{M}_{det}^P$ .

The runs of the deterministic automaton  $\mathcal{M}_{det}^P$  are all possible executions of the protocol according to the action theory, while the runs of  $\mathcal{M}_\mathcal{C}$  describe all possible executions satisfying the constraints in  $\mathcal{C}$ .

The Büchi automaton  $\mathcal{M}_P$  describing all runs of the protocol can thus be obtained as follows:

1. Build a labelled Büchi automaton  $\mathcal{M}_{det}^P$  obtained from the action and causal laws, precondition laws and the initial state, as described above. This automaton is deterministic, all states can be considered as accepting states, and the labels are complete.
2. Build a labelled Büchi automaton  $\mathcal{M}_\mathcal{C}$  obtained from the set of DLTL formulas expressing constraints [8]. This automaton will, in general, be nondeterministic. It is well-known that not every Büchi automaton has an equivalent deterministic Büchi automaton.
3. Build the product of the two automata (see Appendix)  $\mathcal{M}_P = \mathcal{M}_{det}^P \otimes \mathcal{M}_\mathcal{C}$ .  $\mathcal{M}_P$  will be a labelled nondeterministic Büchi automaton. Since all states of  $\mathcal{M}_{det}^P$  are accepting,  $\mathcal{M}_P$  is a standard Büchi automaton (not a generalized one).

## 4.2 An automata-based verification algorithm

In this section we define an algorithm to verify the conformance of an agent  $S$  with a protocol  $P$ . The specification of the protocol is given by the non deterministic automaton  $\mathcal{M}_P$ , defined in the previous section. In the following, we disregard state labels of  $\mathcal{M}_P$ , and consider it as a standard non-deterministic Büchi automaton. The behavior of the agent  $S$  is given by a *deterministic* Büchi automaton  $\mathcal{M}_S$ , whose accepted runs provide all the possible executions of the agent. Observe that, although  $\mathcal{M}_S$  is deterministic, the non deterministic behaviors of the agent can be modelled through the “nondeterministic choice” among different actions.

We assume that the automata  $\mathcal{M}_P$  and  $\mathcal{M}_S$  have been pruned by eliminating all the states which do not occur on any accepted run. This can be achieved by starting from the accepting states, and by propagating backwards the information on the states for which a path to an accepting state exists.

In order to verify the conformance of agent  $S$  with a protocol  $P$ , we define the synchronous product between  $\mathcal{M}_P$  and  $\mathcal{M}_S$

$$\mathcal{M} = \mathcal{M}_P \otimes \mathcal{M}_S$$

whose runs are all the runs of  $S$  which are also runs of  $P$ .  $\mathcal{M}$  is a non-deterministic generalized Büchi automaton.

The states of  $\mathcal{M}$  are triples  $\langle q_D, q_C, q_S \rangle$ , where  $q_D \in \mathcal{M}_{det}^P$ ,  $q_C \in \mathcal{M}_C$  and  $q_S \in \mathcal{M}_S$ . We assume that all the states of  $\mathcal{M}$  which are on an accepting run are marked as *alive*.

In order to verify the conformance we must be able to consider all states of  $\mathcal{M}$  which are reachable with the same prefix. Unfortunately we know that it is not possible to transform  $\mathcal{M}$  into an equivalent deterministic Büchi automaton. Therefore, we proceed as follows.

Let  $\mathcal{M}' = (Q, \Delta, Q^0)$  be a non-deterministic finite state automaton obtained by deleting the accepting states from  $\mathcal{M}$ . We can now apply to  $\mathcal{M}'$  the classical powerset construction for obtaining a deterministic automaton  $\mathcal{M}_{PS} = (Q_{PS}, \Delta_{PS}, q_{PS}^0)$ , where

- $Q_{PS} = 2^Q$
- $(q_{PS}, a, q'_{PS}) \in \Delta_{PS}$  iff  $q'_{PS} = \{q' \in Q : \exists q \in q_{PS} \text{ and } (q, a, q') \in \Delta\}$
- $q_{PS}^0 = Q^0$
- $F_{PS} = Q_{PS}$ .

Let  $Q_{PS}^R \subset Q_{PS}$  be the subset of states of  $\mathcal{M}_{PS}$  reachable from the initial state  $q_{PS}^0$ . Let  $q_{PS}^R = \{q_1, \dots, q_n\}$  be a state of  $Q_{PS}^R$ , and let  $\sigma$  be a prefix with which this state can be reached from the initial state. By construction of  $\mathcal{M}_{PS}$ , the states in  $q_{PS}^R$  are all the states which are reachable in  $\mathcal{M}$  from an initial state through the prefix  $\sigma$ . As pointed out before, every state  $q_i \in q_{PS}^R$  has the form  $\langle q_D^i, q_C^i, q_S^i \rangle$ . Since the first and third component of  $q_i$  are states of a deterministic automaton, all  $q_D^i$  of all states in  $q_i$  will be equal, and the same for  $q_S^i$ .

For verifying the conformance of an agent  $S$  with a protocol  $P$  ( $P$  involving two roles  $i$  and  $j$  and  $S$  playing role  $i$ ), we will refer to the automaton  $\mathcal{M}$ , but we will also make use of the states of the automaton  $\mathcal{M}_{PS}$  to reason on the set of states of  $\mathcal{M}$  reachable with the same prefix. We give the following algorithm.

**Algorithm (for verifying the conformance of  $S$  with  $P$ )**

For each state  $q_{PS}^R = \{ \langle q_D, q_C^1, q_S \rangle, \dots, \langle q_D, q_C^n, q_S \rangle \}$  of  $Q_{PS}^R$ , verify the following conditions:

- If in  $\mathcal{M}_S$  there is an outgoing action  $send_{i,j}$  from  $q_S$ , then there must be a state  $(q_D, q_C^k, q_S)$  of  $\mathcal{M}$  with an outgoing edge labelled with action  $send_{i,j}$ , leading to an alive state.
- For all the states  $(q_D, q_C^k)$  of  $\mathcal{M}_P$ , if there is an outgoing action  $send_{j,i}$  from  $(q_D, q_C^k)$ , then there must be a state  $\langle q_D, q_C^l, q_S \rangle$  of  $q_{PS}^R$ , so that in  $\mathcal{M}$  there is an outgoing edge from  $\langle q_D, q_C^l, q_S \rangle$  labelled with action  $send_{j,i}$ , leading to an alive state.
- For all the states  $(q_D, q_C^k)$  of  $\mathcal{M}_P$ , if there is an outgoing action  $send_{i,j}$  from  $(q_D, q_C^k)$ , then there must be a state  $\langle q_D, q_C^l, q_S \rangle$  of  $q_{PS}^R$ , so that in  $\mathcal{M}$  there is an outgoing edge from  $\langle q_D, q_C^l, q_S \rangle$  labelled with action  $send'_{i,j}$ , leading to an alive state.

We want to evaluate the complexity of the algorithm with respect to the size  $n$  of the protocol automaton. We assume that the size of the agent automaton  $\mathcal{M}_S$  is  $O(n)$ . Although the size of the product automaton is polynomial in  $n$  (namely,  $O(n^2)$ ), the size of the automaton  $\mathcal{M}_{PS}$  is exponential in the size of  $\mathcal{M}$ . Hence, the algorithm requires exponential time in  $n$ . It has to be observed, however, that, when the protocol automaton is deterministic, the automaton  $\mathcal{M}_{PS}$  is useless (each state,  $q_{PS}^R$  of  $Q_{PS}^R$  contains a single triple) and the complexity of the algorithm becomes polynomial in  $n$ .

## 5 Conclusions and Related Work

In this paper we have addressed the problem of conformance between an agent and a protocol, assuming that the specification of the protocol is given in a temporal action logic. We have addressed the case when the protocol involves two agents and we have defined an algorithm which verifies the conformance of an agent with a protocol, by making use of automata-based techniques.

In [3] a similar approach is used for conformance verification, by taking into account the asymmetry between messages that are sent and messages that are received. Agents and protocols are represented as deterministic finite automata, and protocols are limited to protocols with only two roles. The results of that paper have been extended in [4], where conformance of web services is considered. First of all, protocols can contain an arbitrary number of roles. Furthermore, by referring to nondeterministic automata, the proposed approach accounts also for the case of agents and roles which produce the same interactions but have different branching structures. This case cannot be handled in the framework in

[3] as well as in our framework, due to the fact that it is exclusively based on a trace semantics.

A similar approach is also used in [2], where an abductive framework is used to verify the conformance of agents to a choreography with any number of roles.

The notions of conformance, coverage and interoperability are defined in a different way in [6]. A distinctive feature of that formalization is that the three are orthogonal to each other. Conformance and coverage are based on the semantics of runs (a run being a sequence of states), whereas interoperability among agents is based upon the idea of blocking.

In [5], several notions of *compatibility* among agents have been analyzed, in which agents are modelled by Labelled Transition Systems, communication is synchronous, and models are deterministic (no two actions labelled by the same name can be applied in a given state). While compatibility is concerned with the interoperability of agents, in [5] a notion of *substitutability* is introduced, which is related to the notion of conformance. The problem of substitutability is that of determining if a agent  $A'$  can substitute a agent  $A$ , while preserving the compatibility with all the agents  $B$  with whom  $A$  is compatible. [5] introduces two distinct notions of substitutability: the first one requires that  $A'$  at each state can have less emissions and more receptions than  $A$ , and this, in essence, corresponds to requirements (1) and (2) in our definition of conformance. This notion of substitutability does not preserve deadlock-freeness. The second notion of substitutability is more restrictive and requires that  $A'$  and  $A$  have the same emissions and receptions in the corresponding states. As a difference with our proposal, in [5] agent executions are always terminating.

In [13] a notion of conformance is defined to check if an implementation model  $I$  extracted from a message-passing program conforms with a signature  $S$ . Both  $I$  and  $S$  are CCS processes, communication is asynchronous, and the paper, in particular, focuses on stuck-freeness of communication.

The approach presented in the paper can be generalized to an arbitrary number  $n$  of agents, although the generalization is not straightforward. More precisely, in the general case, we would like to show that, given a protocol  $P$  with  $k$  roles and a set of agents  $S_1, \dots, S_k$ , if the behavior of each agent  $S_i$  is conformant with the protocol  $P$ , then the interaction of the agents does not lead to deadlock situations and it gives rise only to executions of the protocol  $P$ .

The main difficulty in generalizing the proposed approach to an arbitrary number of agents comes from the fact that, given a protocol  $P$  involving  $k$  agents, it is not guaranteed that the constraints in  $P$  (which declaratively define which are the wanted executions) can be enforced directly on the agents  $S_1, \dots, S_k$ . Consider, for instance, a protocol involving four agents  $A, B, C, D$  containing the constraint:

$$[m_1(A, B)] < m_2(C, D) > T$$

meaning that message  $m_2$ , sent from  $C$  to  $D$  has to be executed after  $m_1$ , sent from  $A$  to  $B$ . Assume that  $A$  and  $B$  do not exchange messages with  $C$  and  $D$ . It is clear that this constraint cannot be enforced by agents  $A$  or  $B$  alone, as they do not see message  $m_2$ , nor by agents  $C$  or  $D$  alone, as they do not see message

$m_1$ , while both the messages are involved in the constraint. Intuitively, only constraints which are defined on the language (fluents and actions) of single agents should be allowed, as they can be enforced by single agents. A full discussion of the general problem and its solutions will be subject of further work.

## References

1. W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. WS-FM 2006. 1-23, 2006.
2. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and M. Montali. An abductive framework for a-priori verification of web agents. In Principles and Practice of Declarative Programming, PPDP06). ACM Press, 2006.
3. M. Baldoni, C. Baroglio, A. Martelli, and Patti. Verification of protocol conformance and agent interoperability. In Post-Proc. of CLIMA VI, LNCS 3900, pages 265–283, 2006.
4. M. Baldoni, C. Baroglio, A. Martelli, and Patti. A Priori Conformance Verification for Guaranteeing Interoperability in Open Environments. ICSOC 2006, LNCS 4294, pp. 339351, 2006.
5. L. Bordeaux, G. Salaün, D. Berardi, M. Mecella, When are two web-agents compatible, VLDB-TES 2004.
6. A. K. Chopra, M. P. Singh: Producing Compliant Interactions: Conformance, Coverage, and Interoperability. DALT 2006: 1-15, 2006.
7. X. Fu, T. Bultan and J. Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. 328(1-2): 19-37, 2004.
8. L. Giordano and A. Martelli. Tableau-based Automata Construction for Dynamic Linear Time Temporal Logic. *Annals of Mathematics and Artificial Intelligence*, 46(3): 289-315, Springer 2006.
9. L. Giordano, A. Martelli, and C. Schwind. Verifying Communicating Agents by Model Checking in a Temporal Action Logic. *Proc. JELIA 2004*, Lisbon, Portugal, Springer LNAI 3229, 57-69, 2004.
10. L. Giordano, A. Martelli and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic *Journal of Applied Logic (Special issue on Logic Based Agent Verification)*, Accepted for publication, 2006, Elsevier.
11. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, vol.96, n.1-3, 187–207, 1999
12. A. Martelli and L. Giordano. Reasoning About Web Services in a Temporal Action Logic. In O. Stock and M. Schaerf, editors, Reasoning, Action and Interaction in AI Theories and System, Springer LNAI 4155, 229–246, 2006.
13. S.K. Rajamani and J. Rehof. Conformance checking for models of asynchronous message passing software. In *Proc. of Conf. on Computer Aided Verification CAV'02 2005*. 166–179, Springer, 2002.
14. Raymond Reiter. *Knowledge in Action*. The MIT Press, 2001.
15. P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *AAMAS'02*, 527–534, Bologna, Italy, 2002.

## A Büchi automata product

A generalized Büchi automaton  $(Q, \Delta, Q^0, F)$  is like a standard Büchi automaton except for the acceptance condition.  $F = \{F_1, \dots, F_n\}$  is a set of sets of states, and an accepting run must contain infinitely many times at least a state for each  $F_i$ . A generalized Büchi automaton with  $k$  states can be easily translated to a standard Büchi automaton with  $k \times n$  states.

Given two Büchi automata  $\mathcal{B}_1 = (Q_1, \Delta_1, Q_1^0, F_1)$  and  $\mathcal{B}_2 = (Q_2, \Delta_2, Q_2^0, F_2)$ , the product automaton  $\mathcal{B}_1 \otimes \mathcal{B}_2$  that accepts  $\mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$  is a *generalized* Büchi automaton  $\mathcal{B}_1 \otimes \mathcal{B}_2 = (Q, \Delta, Q^0, F)$  where

- $Q = Q_1 \times Q_2$
- $((q_1, q_2), a, (q'_1, q'_2)) \in \Delta$  iff  $(q_1, a, q'_1) \in \Delta_1$  and  $(q_2, a, q'_2) \in \Delta_2$
- $Q^0 = Q_1^0 \times Q_2^0$
- $F = \{F_1 \times Q_2, Q_1 \times F_2\}$

In particular, if all states of  $\mathcal{B}_1$  are accepting, the product automaton is a standard Büchi automaton, where  $F = Q_1 \times F_2$ .