Producing Compliant Interactions: Conformance, Coverage, and Interoperability

Amit K. Chopra and Munindar P. Singh

North Carolina State University

Abstract. Agents in an open system interact with each other based on (typically, published) protocols. An agent may, however, deviate from the protocol because of its internal policies. Such deviations pose certain challenges: (1) the agent might no longer be conformant with the protocol—how do we determine if the agent is conformant? (2) the agent may no longer be able to interoperate with other agents—how do we determine if two agents are interoperable? (3) the agent may not be able to produce some protocol computations; in other words, it may not cover the protocol—how we determine if an agent covers a protocol?

We formalize the notions of conformance, coverage and interoperability. A distinctive feature of our 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 traditional idea of *blocking*. We present a number of examples to comprehensively illustrate the orthogonality of conformance, coverage, and interoperability.

Compliance is a property of an agent's execution whereas conformance is a property of the agent's design. In order to produce only compliant executions, first and foremost the agent must be conformant; second, it must also be able to interoperate with other agents.

1 Introduction

We investigate the topic of an agent's compliance with a protocol by checking its design for conformance with the protocol and interoperability with other agents. Our agents are set in an open environment, and thus expected to be autonomous and heterogeneous. The interactions of agents are characterized in terms of protocols. The autonomy of an agent is reflected in its policies, which affect how it interacts with others, possibly resulting in deviations from the given protocol.

Deviations complicate the task of determining compliance. To take a simple example, a customer in a purchase protocol may send reminders to a merchant at its own discretion even though the protocol did not encode sending reminders. Some deviations can be flagrant violations. For example, a customer may not pay after receiving the goods it ordered. What can we say about the compliance of these agents? Sending a reminder seems like an innocuous deviation from protocol, whereas not sending the payment appears more serious. One could argue that sending reminders could have been easily incorporated into the protocol. However, when we consider that deviations in protocol are a manifestation of the individual policies of agents, the number of possible deviations from a protocol is potentially infinite. As more deviations are encoded, the resulting protocol would become large and unwieldy. If each deviant protocol were published as a separate protocol, too many niche protocols would arise. It is better to maintain a smaller number of general protocols and to entertain deviations from such protocols. However, not all deviations are acceptable from the point of view of compliance.

1.1 Compliance: Conformance and Interoperability

For an agent to be compliant with a protocol, first and foremost it must be conformant with the protocol. While agent compliance can only be checked by monitoring the messages the agent exchanges with its peers at runtime, conformance can be verified from its design. The design of an agent involves two primary components: protocols and policies. Protocols are the public part of the design and can be considered fixed for the set of agents that adopt specific roles in the protocol. However, the policies are private to each agent, and potentially unique to each agent. Hence, the design of an agent is a function of its policies. An agent is conformant with a protocol if it respects the semantics of the protocol. A useful criterion when considering conformance is the satisfaction of commitments. Our definition of conformance supports commitments, but it is more general.

The distinction between conformance and compliance is important: an agent's design may conform, but its behavior may not comply. This may be not only because of the agent's failure or unreliable messaging (which do not concern us here), but also because an agent's design may preclude successful interoperation with its peers. In other words, even though an agent is individually conformant, it may not be able to generate compliant computations because of the other agents with whom it interacts, apparently according to the same protocol. Interoperability is distinct from conformance; interoperability is strictly with respect to other agents, whereas conformance is with respect to a protocol.

1.2 Coverage

A protocol may offer a number of alternative execution paths. Some of those paths may be impossible for an agent who deviates from the protocol. Such a reduction in possible paths may be viewed as a reduction in the capabilities of an agent. Conversely, the agent's design may make it possible to interact along paths unforeseen in the protocol. Such an addition may be viewed as an increase in the capabilities of an agent. Informally, we say an agent covers a protocol if it capable of taking any of the paths in the protocol.

This notion of coverage is an important one: if an agent covers a protocol it would appear to be at least as flexible as the protocol. That is, the agent can handle whatever the protocol can "throw" at it. Moreover, in some settings it may be institutionally required that an agent cover a protocol. For example, a tax official must report discrepancies in reviewed filings to the main office; the official cannot ignore them.

1.3 Contributions and Organization

Our contributions include (1) an account of conformance and coverage based on a semantics for protocols suitable for open systems; (2) showing how conformance, coverage, and interoperability are orthogonal concerns; and (3) establishing that in order to only produce compliant interactions, one has to consider both an agent's conformance with the protocol, and its interoperability with other agents.

Section 2 presents the representation of protocols as transition systems. Section 3 discusses the way in which an agent may deviate from protocol. Section 4 defines conformance and coverage. Section 5 discusses the interoperability of agents. Section 6 shows that conformance, coverage, and interoperability are orthogonal; it also discusses the relevant literature.

2 Protocols

We represent protocols as transition systems; the transition systems are similar to those described by $C+$ specifications [5]. The *signature* of a transition system is the set σ of constants that occur in it. Here σ^{act} and σ^{fl} represent the sets of actions and fluents, respectively. Each constant *c* is assigned a nonempty finite domain *Dom(c)* of symbols. An *interpretation* of σ is an assignment *c* = *v* for each $c \in \sigma$ where $v \in$ *Dom(c)*.

Informally, a transition system is a graph with states as vertices and actions as edges. A state s is a particular interpretation of σ^{fl}, the set of fluents; a transition is a triple $\langle s, e, s' \rangle$ where s and s' are states, and e is an interpretation of σ^{act}, the set of actions. In addition, the initial and final states are marked.

Definition 1. A transition system is a $\langle \sigma^{fl}, \sigma^{act}, S, s_0, F, \delta \rangle$, where σ^{fl} is the set of fluents, σ^{act} is the set of actions, S is the set of states such that $S \subseteq 2^{\sigma^{fl}}$, $s_0 \in S$ is an initial state, $F \subseteq S$ is the set of final states, $\delta \subseteq S \times E \times S$ is the set of transitions, where $E \subseteq 2^{\sigma^{act}}$.

Figure 1 shows the transition system of a purchase protocol. The protocol has two roles: *merchant* (*mer*) and *customer* (*cus*) engaging in the steps below:

1. The customer sends a *request* for quotes to the merchant.
2. The merchant responds either by sending an *offer* for the goods for which the customer requested a quote, or by indicating the nonavailability of requested goods in which case the protocol ends. By sending an offer, the merchant creates the conditional commitment *CC(mer, cus, a_price, an_item)* meaning that if the customer pays price *a_price*, then the merchant will send the goods *an_item*.
3. The customer can respond to the offer by either sending an *accept*, or a *reject*. Accepting the quote creates a conditional commitment *CC(cus, mer, an_item, a_price)*, meaning that if the merchant sends the goods, then the customer will pay. If the customer sends a *reject*, the protocol ends.
4. If the customer sends a *payment* to the merchant, then *CC(cus, mer, an_item, a_price)* is discharged and *CC(mer, cus, a_price, an_item)* is reduced to *C(mer, cus, an_item)* meaning that the merchant is now committed to sending the goods. But if the merchant sends *an_item* to the customer, then *CC(mer, cus, a_price,an_item)* is discharged and *CC(cus, mer, an_item, a_price)* is reduced to *C(cus, mer, a_price)* meaning that the customer is now committed to paying for the goods.
5. If the customer has paid in the previous step, then the merchant sends the goods, thereby discharging its commitment. But if the merchant has sent the goods in

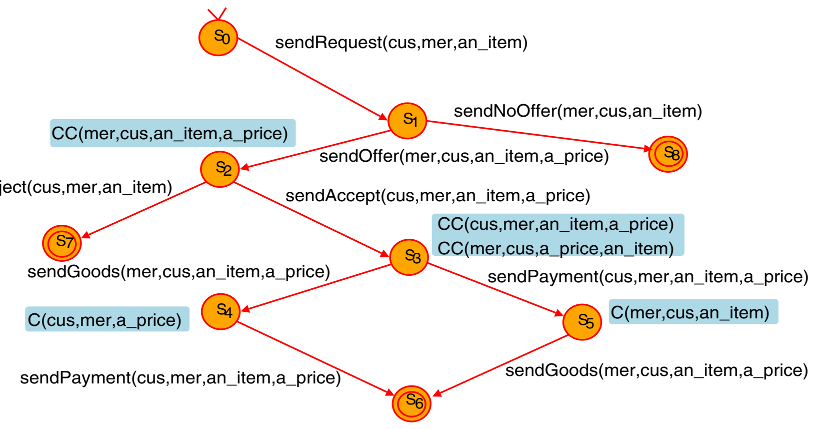
S0
sendRequest(cus,mer,an_item)
S1
sendNoOffer(mer,cus,an_item)
S8
CC(mer,cus,an_item,a_price)
sendOffer(mer,cus,an_item,a_price)
S2
sendReject(cus,mer,an_item)
S7
sendAccept(cus,mer,an_item,a_price)
CC(cus,mer,an_item,a_price)
CC(mer,cus,a_price,an_item)
S3
sendGoods(mer,cus,an_item,a_price)
sendPayment(cus,mer,an_item,a_price)
S4
C(cus,mer,a_price)
S5
C(mer,cus,an_item)
sendPayment(cus,mer,an_item,a_price)
sendGoods(mer,cus,an_item,a_price)
S6

Fig. 1. A purchase protocol

the previous step, then the customer sends the payment, thereby discharging its commitment. In either case, no commitments or conditional commitments hold in the resulting state, which is a final state of the protocol.

Table 1 shows the interpretation of states in the transition system. An action starting with ‘send’ represents a single message exchange between roles with the *sender* role and *receiver* role as the first and second arguments, respectively. The fluents *initial* and *final* mark the start state and the final states respectively.

We now introduce some definitions related to transition systems.

Definition 2. A path in a transition system is a series of transitions $\langle s_0, e_0, s_1\rangle$, $\langle s_1, e_1, s_2\rangle$, . . .,$\langle s_{f-1}, e_{f-1}, s_f\rangle$ such that s_0 is the initial state, and s_f is a final state.

A path may be abbreviated as $\langle s_0, e_0, s_1, e_1, \ldots, e_{f-1}, s_f\rangle$. Given a path $\rho = \langle s_0, e_0,$ $s_1, \ldots, s_i, e_i, \ldots, e_{f-1}, s_f\rangle$, we say $e_i \in \rho$ $(0 \leq i < f)$, and $s_i \in \rho$ $(0 \leq i \leq f)$.

We restrict our attention to two-party protocols. All the actions performed by the agents are communications. We further assume about the transition system of any protocol or agent that (1) only one action is performed along any transition; (2) in any transition $\langle s, e, s'\rangle$, $s \not\equiv s'$; (3) there exist no transitions $\langle s, e, s'\rangle$ and $\langle s, e', s'\rangle$ such that $e \equiv e'$ (in other words, no two distinct actions cause a transition into the same destination state from the same origin state); (4) the transition system is deterministic; and (5) along any path in the transition system, an action is performed at most once.

Definition 3. A *run* in a transition system is a series of states $\langle s_0, s_1, \ldots, s_f\rangle$ such that there exists a path $\langle s_0, e_0, s_1, e_1, \ldots, e_{f-1}, s_f\rangle$ in the transition system.

For example, the protocol of Figure 1 has the runs: $\langle s_0, s_1, s_8\rangle$, $\langle s_0, s_1, s_2, s_7\rangle$, $\langle s_0, s_1,$ $s_2, s_3, s_4, s_6\rangle$, and $\langle s_0, s_1, s_2, s_3, s_5, s_6\rangle$. Note that given the above restrictions, each run maps to a unique path and vice versa.

Definition 4. The t-span $[T]$ of a transition system T is the set of paths in T.

Table 1. States in Figure 1

State	Fluents
s_0	*initial*
s_1	*request(cus, mer, an_item)*
s_2	*request(cus, mer, an_item)*, *offer(mer, cus, an_item, a_price)*, *CC(cus, mer, an_item, a_price)*
s_3	*request(cus, mer, an_item)*, *offer(mer, cus, an_item, a_price)*, *accept(cus, mer, an_item, a_price)*, *CC(cus, mer, an_item, a_price)*, *CC(mer, cus, a_price, an_item)*
s_4	*request(cus, mer, an_item)*, *offer(mer, cus, an_item, a_price)*, *accept(cus, mer, an_item, a_price)*, *goods(mer, cus, an_item, a_price)*, *C(cus, mer, a_price)*
s_5	*request(cus, mer, an_item)*, *offer(mer, cus, an_item, a_price)*, *accept(cus, mer, an_item, a_price)*, *pay(cus, mer, an_item, a_price)*, *C(mer, cus, an_item)*
s_6	*request(cus, mer, an_item)*, *offer(mer, cus, an_item, a_price)*, *accept(cus, mer, an_item, a_price)*, *goods(mer, cus, an_item, a_price)*, *pay(cus, mer, an_item, a_price)*, *final*
s_7	*request(cus, mer, an_item)*, *offer(mer, cus, an_item, a_price)*, *reject(cus, mer, an_item, a_price)*, *final*
s_8	*request(cus, mer, an_item)*, *no_offer(mer, cus, an_item)*, *final*

Notice that t-span is thus defined for protocols, role skeletons, and agents.

For example, $\{\langle s_0, s_1, s_2, s_7\rangle, \langle s_0, s_1, s_8\rangle, \langle s_0, s_1, s_2, s_3, s_4, s_6\rangle, \langle s_0, s_1, s_2, s_3, s_5, s_6\rangle\}$ is the t-span of the purchase protocol of Figure 1.

3 Deviating from Protocol

A role skeleton is a projection of a protocol onto a particular role; it is the transition system of the role. Figure 2 shows the customer skeleton. A customer's policies are combined with the customer role to create a new transition system representing the customer agent. Saying an agent is conformant with a protocol is the same as saying it is conformant with the role it adopts in the protocol; the same holds for coverage. Also note that if the transition system of an agent is identical to the skeleton of the role it adopts, we shall say that the agent *follows* the role.

The policies that go into designing an agent may be such that it follows a protocol. Or, they may be such that the agent encodes deviations from the protocol. Below, we list some common kinds of deviations.

Narrowing. The t-span of an agent is a proper subset of the t-span of the role skeleton it adopts: a typical reason for this would be to simplify its implementation.

Example 1. As shown in the agent's transition system in Figure 3, the customer requires the goods to arrive before it sends the payment. Essentially, the customer has removed a run from the role skeleton, namely, the run in which payment happens before the delivery of goods. ▮

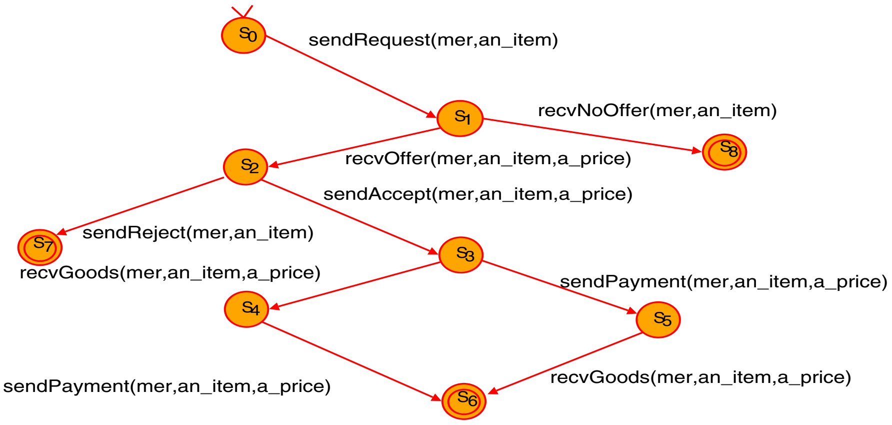
S0
sendRequest(mer,an_item)
S1
recvNoOffer(mer,an_item)
S8
recvOffer(mer,an_item,a_price)
S2
sendAccept(mer,an_item,a_price)
sendReject(mer,an_item)
S7
S3
recvGoods(mer,an_item,a_price)
sendPayment(mer,an_item,a_price)
S4
S5
sendPayment(mer,an_item,a_price)
recvGoods(mer,an_item,a_price)
S6

Fig. 2. Customer role skeleton

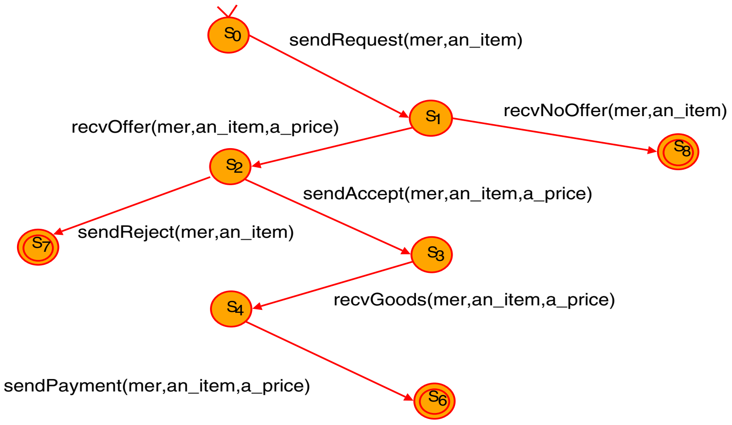
S0
sendRequest(mer,an_item)
recvOffer(mer,an_item,a_price)
S1
recvNoOffer(mer,an_item)
S8
S2
sendAccept(mer,an_item,a_price)
S7
sendReject(mer,an_item)
S3
S4
recvGoods(mer,an_item,a_price)
sendPayment(mer,an_item,a_price)
S6

Fig. 3. Customer who sends payment only after receiving goods

Broadening. The t-span of the role skeleton is a proper subset of the t-span of the agent that adopts that role: a typical reason for this would be to handle scenarios not encoded in the protocol.

Example 2. The customer agent sends a reminder to the merchant about its commitment to send goods. Thus, in addition to the original runs, the customer agent includes the run in which it sends a reminder. For the sake of brevity, Figure 4 only shows the additional run; the remaining runs are as in Figure 2. ∎

Lengthening. The t-span of an agent is similar to that of the role skeleton except that some runs in the t-span of the agent are longer than the corresponding runs in the role skeleton: the reason is that additional actions happen along the path corresponding to the run.

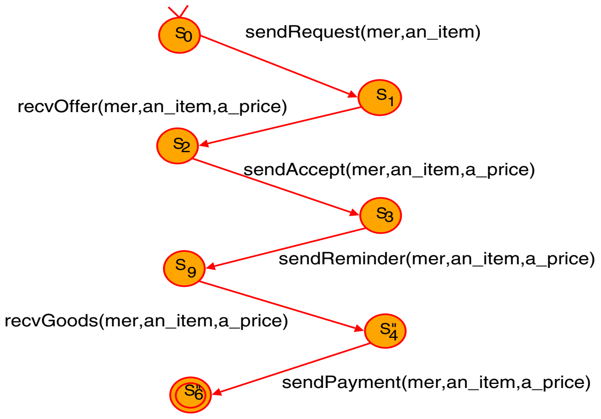
S0
sendRequest(mer,an_item)
S1
recvOffer(mer,an_item,a_price)
S2
sendAccept(mer,an_item,a_price)
S3
sendReminder(mer,an_item,a_price)
S9
recvGoods(mer,an_item,a_price)
S4"
sendPayment(mer,an_item,a_price)
S6"

Fig. 4. The run in which customer sends a reminder

Example 3. If we replace the run $\langle s_0, s_1, s_2, s_3, s_4, s_6 \rangle$ in the customer role skeleton (shown in Figure 2) with the run in which a reminder is sent (shown in Figure 4), then it represents an example of lengthening. ▮

Example 4 illustrates the shortening of runs.

Example 4. Consider the customer of Figure 5. After receiving goods, the customer does not send payment for them. State s_4 is a final state for this customer. ▮

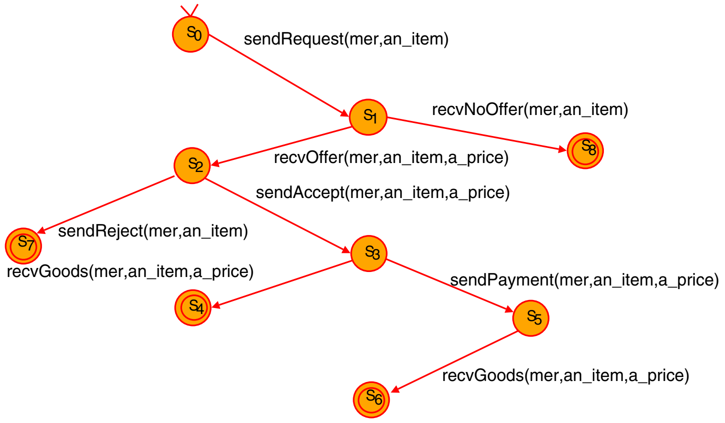
S0
sendRequest(mer,an_item)
S1
recvNoOffer(mer,an_item)
S8
recvOffer(mer,an_item,a_price)
S2
sendAccept(mer,an_item,a_price)
sendReject(mer,an_item)
S7
recvGoods(mer,an_item,a_price)
S3
sendPayment(mer,an_item,a_price)
S4
S5
recvGoods(mer,an_item,a_price)
S6

Fig. 5. Customer who does not pay for received goods

Gating. An agent may broaden or lengthen a protocol in such a way that it expects to receive additional messages from its partners in order to proceed.

Example 5. The customer agent may ask for warranty information upon receiving the goods, in which case it expects the merchant to send the information before it makes the payment. The customer is thus “gated” upon the receipt of warranty information. Figure 6 shows the additional run; the remaining runs are as in Figure 2. ∎

Note that combinations of the above deviations are also possible. Example 5, for instance, represents a case of both gating and broadening. It is also possible that an agent represents narrowing as in Example 1, and at the same time represents broadening and gating as in Example 5.

4 Conformance and Coverage

What can we say about the conformance of the customer agent in each of the above examples? Clearly, no customer is following the purchase protocol. Then, are they are all nonconformant? If we look at the agents from the point of view of commitments, it sheds some light on their conformance. No commitments remain unsatisfied in any run in the customers in Examples 1, 2, 3, and 5. We would expect that these customers are determined conformant to their roles. The customer of Example 4, however, has a pending commitment (to pay) in its final state s_4. Consequently, this customer should be determined to be nonconformant.

Similarly, what can we say about the coverage of the customer agent in each of the above examples? Based on the discussion of coverage in Section 1, we would expect the customers in Examples 2 and 5 to be determined to be covering the protocol, whereas the customers in Examples 1 and 4 to be determined to be noncovering. The customer of Example 3 is more interesting: it could be identified as noncovering since one of the runs of the protocol is missing. However, this run has been replaced by a run that sends a reminder: the replacement run is quite similar to the missing run. Sending a reminder does not affect the commitments. Hence, we would expect the customer in Example 3 to be covering.

Our definitions of conformance and coverage rely on the notion of run subsumption [6]. In the following, we briefly discuss run subsumption, then we formally define conformance and coverage.

We introduce *state similarity* to compare states. A state-similarity function f maps a state to a set of states, i.e., $f : \mathbf{S} \rightarrow 2^{\mathbf{S}}$. From f, we induce a binary relation $\approx_f \subseteq \mathbf{S} \times \mathbf{S}$, where $\approx_f = \{(s, f(s)) : s \in \mathbf{S}\}$. We require f to be such that $\approx_f$ is an equivalence relation. For example, commitments could be used to compare states. Two states are *commitment similar* if the same set of commitments hold in them.

Let $\prec_\tau$ be a temporal ordering relation on states in a run τ. That is, $s \prec_\tau s'$ means that s occurs before s' in τ.

Definition 5. A run τ_j subsumes τ_i under a state-similarity function f, denoted by $\tau_j \gg_f \tau_i$ if for every state s_i that occurs in τ_i, there exists a state s_j that occurs in τ_j such that $s_j \approx_f s_i$, and for all s'_i that occur in τ_i, if $s_i \prec_{\tau_i} s'_i$ then there exists s'_j that occurs in τ_j such that $s_j \prec_{\tau_j} s'_j$ and $s'_j \approx_f s'_i$.

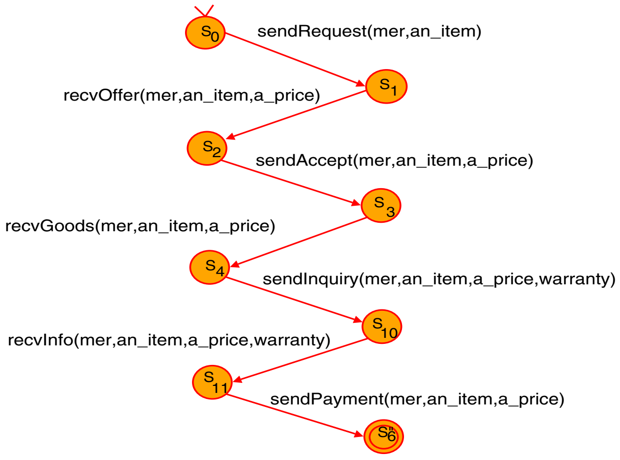
S_0
sendRequest(mer,an_item)
S_1
recvOffer(mer,an_item,a_price)
S_2
sendAccept(mer,an_item,a_price)
S_3
recvGoods(mer,an_item,a_price)
S_4
sendInquiry(mer,an_item,a_price,warranty)
S_{10}
recvInfo(mer,an_item,a_price,warranty)
S_{11}
sendPayment(mer,an_item,a_price)
S_6

Fig. 6. The run in which the customer asks for warranty information

Run subsumption is reflexive, transitive, and antisymmetric up to state similarity [6]. Longer runs subsume shorter runs, provided they have similar states in the same temporal order.

The closure of a protocol is a span that is closed under run subsumption. That is, if a run is in the closure, then all the runs that subsume it (under some state-similarity function) are also in the closure. Closures are unique (under a particular state-similarity function), and provide a firm basis for comparing protocols; a different closure may be obtained by changing the state-similarity function. For the purposes of this paper, we will use the commitment-similarity function.

Definition 6. The closure of a protocol P under a state-similarity function f is given by $[\![P]\!]_f = \{\tau \mid \forall \tau' \in [P] : \tau \gg_f \tau'\}$.

Definition 7. An agent α is conformant with a protocol P under a state-similarity function f if $[\alpha] \subseteq [\![P]\!]_f$.

As expected, Definition 7 renders the customers in Examples 1, 2, 3 and 5 conformant with the protocol, and the customer in Example 4 nonconformant with the protocol.

Definition 8. An agent α covers a protocol P under a state-similarity function f, if for each $\tau \in [P]$, there exists a $\tau' \in [\alpha]$ such that $\tau' \gg_f \tau$.

As expected, Definition 8 renders the customers in Examples 2, 3 and 5 as covering the protocol, and the customers in Examples 1 and 4 as noncovering.

5 Interoperability

Section 4 defines the conformance of an agent with respect to its role skeleton. A conformant agent respects the semantics of the protocol: it encodes only runs in the closure of the protocol. However, just because an agent is conformant does not mean it can always successfully interoperate with other conformant agents. Let us consider the examples below involving conformant customer and merchant agents; the merchant, in particular, follows its role in the protocol.

It is worth considering the idea of how protocols can be operationally interpreted. An agent may ignore messages that are not in its vocabulary (i.e., do not occur in its t-span). Or an agent may ignore all messages that are unexpected, specifically including those that are in the vocabulary but arrive out of order.

Example 6. The customer sends reminders as shown in Figure 4. The reminders are not in the t-span of the protocol, but are in its closure. Thus they are allowed but may not be implemented by agents playing other roles. Such reminders do not cause a problem in interoperation: the merchant cannot handle reminders, but can ignore them. ▮

Example 6 is benign in that even though the customer does something the merchant does not expect, both can execute the protocol to completion. Therefore, we expect these agents to be rendered interoperable.

Example 7. The customer sends payment only upon receipt of goods (as in Figure 3) from a merchant. Even though, the customer and merchant are individually conformant with their respective roles, they exists the possibility of a deadlock: the customer waits for the merchant to send goods first, and the merchant waits for the customer to send payment first. ▮

Because of the possibility of a deadlock, we expect the agents in Example 7 to be rendered noninteroperable.

Example 8. The customer agent asks for warranty according to the run shown in Figure 6. The merchant cannot fulfill the customer's warranty information request: the merchant simply ignores the request. And the customer would not send payment until it receives the warranty information. ▮

We expect the agents in Example 8 to be rendered noninteroperable. In this example, noninteroperability causes a violation of a customer's commitment to pay.

5.1 Verifying Interoperability

The interoperability of two agents depends upon the computations that they can jointly generate. The agents may act one by one or in true concurrency. Definition 9 captures the above intuitions for a product transition system of a pair of agents.

Definition 9. Given two agents $\alpha_1 := \langle \sigma_1^{fl}, \sigma_1^{act}, S_1, s_{0_1}, F_1, \delta_1 \rangle$ and $\alpha_2 := \langle \sigma_2^{fl}, \sigma_2^{act}, S_2, s_{0_2}, F_2, \delta_2 \rangle$, the product transition system $\alpha_\times = \alpha_1 \times \alpha_2$ is given by $\alpha_\times := \langle \sigma_\times^{fl}, \sigma_\times^{act}, S_\times, s_{0_\times}, F_\times, \delta_\times \rangle$ where,

- $\sigma_\times^{fl} = \sigma_1^{fl} \cup \sigma_2^{fl}$
- $\sigma_\times^{act} = \sigma_1^{act} \cup \sigma_2^{act}$
- $S_\times = S_1 \times S_2$

- $s_{0_\times} = (s_{0_1}, s_{0_2})$
- $F_\times = F_1 \times F_2$
- $\delta_\times \subseteq S_\times \times E_\times \times S_\times$ where $E \subseteq 2^{\sigma_\times^{act}}$ such that $\langle s, e, s' \rangle \in \delta_\times$, where $s = s_1 \times s_2$ and $s' = s'_1 \times s'_2$, $(s_1, s'_1 \in S_1)$, $(s_2, s'_2 \in S_2)$, if and only if
  - $\langle s_1, e, s'_1 \rangle \in \delta_1$, or
  - $\langle s_2, e, s'_2 \rangle \in \delta_2$, or
  - $e = (e_1, e_2)$ and $\langle s_1, e_1, s'_1 \rangle \in \delta_1$ and $\langle s_2, e_2, s'_2 \rangle \in \delta_2$.

The technical motivation behind Definition 9 is that it accommodates the transitions that would globally result as the agents enact the given protocol. When the agents act one by one, the transitions are labeled with an action from their respective σ^{act}. When the agents act concurrently, the transitions are labeled by a pair of actions, one from each agent.

Interoperability can become challenging in light of the fact that communication between agents is asynchronous. The essence of these challenges is that an agent might block indefinitely upon doing a receive. We verify the interoperability of agents by analyzing their product transition system for the absence of such problems.

The sending of a message m by an agent α is represented by $send(\alpha, m)$. Similarly, receiving a message is represented by $recv(\alpha, m)$. When the identity of the agent does not matter, we write only $send(m)$ and $recv(m)$ instead. Below $x, y, \ldots$ range over messages $m_1, m_2, \ldots$, and $\alpha, \beta, \ldots$ are agents.

Definition 10. An action a strictly precedes an action b on a path ρ in the product transition system, denoted by $a \prec_\rho b$, if and only if $a \in e_i$ and $b \in e_j$ such that $e_i, e_j \in \rho$ and $i < k$. If we change the index condition to $i \leq k$, we say $a \preceq_\rho b$.

Next, we identify all the pathological paths in the product, i.e., those that can be never be realized during execution. A kind of pathological path is one whose execution is impossible under considerations of asynchrony, for instance, a path where the receipt of a message happens precedes its sending. Another kind of pathological path can be identified when we associate angelic determinism with attempted receipts of messages. The idea is that if an action $send(\alpha, x)$ happens, and it it is possible to execute one of either $recv(\beta, x)$ or $recv(\beta, y)$ $(x \neq y)$, then $recv(\beta, x)$ is executed. Definition 11 allows for angelic nondeterminism on receives; specifically, it identifies paths that appear "bad" as they appear to block on a receive, but are "saved" by angelic nondeterminism.

Definition 11. A path $\rho = \langle \ldots, s_i, e_i, s_j, \ldots \rangle$ is said to be locally matched by a path $\rho' = \langle \ldots, s_i, e'_i, s_k, \ldots \rangle$ in the product transition system if and only if

- $recv(\alpha, x) \in e_i$ and $send(\beta, x)$ never occurs on ρ, and
- $recv(\alpha, y) \in e'_i$ and $send(\beta, y)$ occurs on ρ'.

Definition 12 defines a product transition system that contains only paths that can be realized (paths which block are considered realizable).

Definition 12. A causal product transition system based on two agents is a transition system whose set of paths is a subset of the paths of the product transition system of the two agents such that it contains no path $\rho = \langle \ldots, s_i, e_i, s_{i+1}, \ldots \rangle$ that satisfies one of the conditions below:

- for some i, $e_i = recv(x)$ or $e_i = (recv(x), send(y))$, such that $recv(x) \prec_\rho send(x)$, or
- it is locally matched.

Definition 13. A path $\rho = \langle \ldots, s_i, e_i, s_{i+1}, \ldots \rangle$ in the product transition system is a deadlock path if and only if $e_i = (recv(x), recv(y))$ and $recv(x) \prec_\rho send(x)$ and $recv(y) \prec_\rho send(y)$.

Note that Definition 12 does not consider $e_i = (recv(x), recv(y))$ as they might be the indication of deadlocks. However, it might remove other paths that are an indication of deadlocks, for example, ρ where $recv(\alpha, x) \prec_\rho recv(\beta, y)$ and $recv(\alpha, x) \prec_\rho send(\beta, x)$ and $recv(\beta, y) \prec_\rho send(\alpha, y)$. However, from the construction of the product, if there is such a path ρ in the product, there must be a corresponding deadlock path ρ'. Hence, ρ may be removed without compromising our ability to detect deadlocks.

Definition 14. A path $\rho = \langle \ldots, s_i, e_i, s_{i+1}, \ldots \rangle$ is a blocking path if and only if for some $recv(x) \in e_i$, there occurs no $send(x)$ on the path.

Definition 15. A path $\rho = \langle \ldots, s_i, e_i, s_{i+1}, \ldots \rangle$ is an out-of-order path if and only if $recv(\alpha, x) \preceq_\rho recv(\alpha, y)$ and $send(\beta, y) \preceq_\rho send(\beta, x)$.

Definition 16. Two agents are interoperable if and only if in the causal product of their transition systems

- there exists no deadlock path, and
- there exists no blocking path, and
- there exists no out-of-order path.

As expected, Definition 16 renders the agents in Example 6 interoperable. Also as expected, it renders the agents in Examples 7 and 8 as noninteroperable; it also renders the customer in Example 4 noninteroperable with a merchant that follows protocol. In fact, if the customer and merchant each follows their roles, they will be rendered noninteroperable (and rightly so) because of the nonlocal choice between sending goods and sending payment. It may be argued that a protocol with nonlocal choice is inherently incomplete and, therefore, may be considered as an abstract protocol. For such protocols further negotiation is necessary between the agents or their designers to ensure interoperability.

6 Discussion

We have defined conformance and coverage in a way that respects the semantics of the protocol. Although we have used a commitment-based semantics, the semantics primarily depend on the state-similarity function. Our definitions of conformance and coverage allow agents flexibility in their interactions, which is crucial in open settings. By contrast, interoperability is strictly about an agent receiving a message it expects to receive. In that manner, interoperability is less semantic, and imposes strict restrictions on agents to interoperate.

6.1 Proving Orthogonality

We prove the orthogonality of conformance, coverage, and interoperability with the help of examples. We have already seen that agents that follow the customer and merchant roles respectively in the purchase protocol of Figure 1 are not interoperable. Now we consider a variant of the purchase protocol which does not have the pay-before-goods path. Specifically, let's consider the protocol of Figure 1, but without the run $\langle s_0, s_1, s_2, s_3, s_5, s_6 \rangle$. Let this variant be P'. Figure 3 would then depict the customer role skeleton; the merchant's role skeleton would be symmetric except that the sends and receives would be swapped. Table 2 considers the conformance and coverage of different merchant agents with respect to P', and the agents' interoperability with a customer that follows the customer role. (Figure 7 shows the paths that the table refers to.) It is clear from the table that the conformance, coverage, and interoperability are orthogonal concerns, because examples of all possible combinations of their truth and falsity exist.

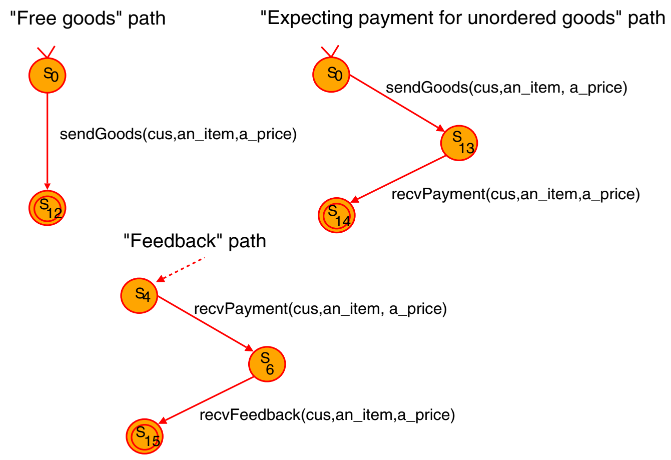
"Free goods" path
S0
sendGoods(cus,an_item,a_price)
S12
"Expecting payment for unordered goods" path
S0
sendGoods(cus,an_item, a_price)
S13
recvPayment(cus,an_item,a_price)
S14
"Feedback" path
S4
recvPayment(cus,an_item, a_price)
S6
recvFeedback(cus,an_item,a_price)
S15

Fig. 7. Some runs used in Table 2

6.2 Literature

Baldoni *et al.* [2] and Endriss *et al.* [4] present alternative formalizations of conformance and interoperability. Both formalizations, however, violate the orthogonality of conformance and interoperability with the result that many agents that should be considered conformant in a practical setting—and are determined to be conformant according to our formalization—are rendered nonconformant in theirs. For example, they would both determine the customer who sends reminders to be nonconformant. Agents that are conformant by our definition, but gate on some message not in the role they are playing, seem to present a problem. Such agents would not be interoperable with agents that are conformant but do not send the message being gated upon. Importantly,

Table 2. Orthogonality of conformance (**C**), coverage (**V**), and interoperability (**X**)

Merchant agents	**C**	**V**	**X**
Only has path expecting payment for unordered goods	×	×	×
Has additional free goods path and no reject path	×	×	✓
Has additional path expecting payment for unordered goods	×	✓	×
Has additional free goods path	×	✓	✓
Has the goods-pay path gated on feedback and no reject path	✓	×	×
Has no reject path	✓	×	✓
Has the goods-pay path gated on feedback	✓	✓	×
Follows role	✓	✓	✓

such agents could potentially violate their commitments because of the gating; in other words, they could potentially produce noncompliant executions. For example, the customer in Example 5 would violate its commitment to pay if it fails to receive the warranty information.

Deeming such an agent nonconformant would, however, be unduly restrictive. The noninteroperability of such an agent with other agents could be detected, and special measures taken to ensure that an agent does not blindly enter such interactions. Specifically, such measures include developing agents who can negotiate with others about the possibilities of deviating from their chosen roles. Even if the agents involved cannot negotiate—then the agents are effectively noninteroperable—this situation is more acceptable than potentially violating a commitment. Our formalization of conformance and interoperability supports such scenarios.

Approaches based on verifying compliance at runtime [1,7] are important in the context of open systems since agents may behave in unpredictable ways; also it is necessary to have independent arbiters in case of disputes involving agents. Such approaches are complementary to this work.

6.3 Directions

A possible direction is to extend this work to more general protocols and agents: specifically, multiparty protocols and agents with infinite runs. Also, this work may be tied together with work on protocol transformations [3]: it would be interesting to be able to determine which transformers, when applied to a protocol, would preserve conformance, coverage, and interoperability.

References

1. M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and verification of agent interaction protocols in a logic-based system. In *Proceedings of the 19th ACM Symposium on Applied Computing (SAC 2004)*, pages 72–78, 2004.
2. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Verification of protocol conformance and agent interoperability. In *6th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA VI)*, pages 265–283, 2005.

3. A. K. Chopra and M. P. Singh. Contextualization of commitment protocols. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006.
4. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 679–684, 2003.
5. E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2):49–104, 2004.
6. A. U. Mallya and M. P. Singh. An algebra for commitment protocols. *Journal of Autonomous Agents and Multiagent Systems special issue on Agent Communication (JAAMAS)*, Apr 2006.
7. M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols: Enabling open Web-based multiagent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, Sept. 1999.