Interoperation in Protocol Enactment

Amit K. Chopra and Munindar P. Singh

North Carolina State University
{akchopra,singh}@ncsu.edu

Abstract. Interoperability has been broadly conceptualized as the ability of agents to work together. In open systems, the interoperability of agents is an important concern. A common way of achieving interoperability is by requiring agents to follow prescribed protocols in their interactions with others. In existing systems, agents must follow any protocol to the letter; in other words, they should exchange messages exactly as prescribed by the protocol. This is an overly restrictive constraint; it results in rigid, fragile implementations and curbs the autonomy of agents. For example, a customer agent may send a reminder to a merchant agent to deliver the promised goods. However, if reminders are not supported explicitly in the protocol they are enacting, then the reminder would be considered illegal and the transaction may potentially fail. This paper studies the interoperation of agents, dealing with their autonomy and heterogeneity in computational terms.

1 Introduction

Protocols describe the interactions among autonomous agents. Thus they are crucial to the design and construction of multiagent systems. Previous work on protocols in multiagent systems has dealt with high-level topics such as semantics [8, 18], composition [17], and verification [13]. However, protocols are enacted by agents in physical systems. In particular, considerations of the underlying communication models and how distributed agents are able to make compatible choices would greatly affect whether a protocol may in fact be enacted successfully. The objective of this paper is to study the computational underpinnings of protocol enactment in multiagent systems. It seeks to characterize the operationalization of agents so as to determine whether and when agents may be interoperable.

The agents we consider are set in open systems, and they 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 problems: (1) the agent might no longer be conformant with the protocol, and (2) the agent may no longer be able to interoperate with other agents.

For an agent to be compliant with a protocol, first and foremost it must be conformant with the protocol. Whereas agent compliance can only be checked by monitoring the messages an agent exchanges with its peers at runtime, conformance can be verified from the agent's design. An agent's design is conformant with a protocol if it respects the semantics of the protocol; a useful semantics is obtained when considering the satisfaction of commitments [12]. The distinction between conformance and compliance is important: an agent's design may conform, but its behavior may not comply. This may be 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.

Protocols provide a way of structuring interactions; however, interoperability is not just a test on agents that adopt roles in the same protocol, and then deviate from their roles. Interoperability is a property of a set of agents. The proposed definition of interoperability declares two agents to be interoperable provided from each joint state that they can enter, they can reach a final state. The essential idea is of determining the states that can be entered. In our approach, these are specified based upon the highly realistic constraint that only messages that have been sent (by an agent) can be received (by another agent). Based on this constraint, some transitions cannot in fact be performed: these transitions correspond to an agent receiving a message before the message has been sent. The transitions that can be performed are termed *causally enabled*.

A communication model sets the physical environment for communication between agents. The parameters of this model include whether communication is synchronous or asynchronous, the number of channels to use, the size of the buffers, and the buffer access mechanism. One cannot simply examine a pair of agents in isolation and decide whether they are interoperable; the agents must be analyzed in light of the communication model in force. Agents that are interoperable in one model may be noninteroperable in another. To analyze interoperability, we capture communication models in terms of causal enablement. Causal enablement is a basic building block that identifies the possible nonblocking actions given the current global state of the interaction. Different models of causal enablement correspond to different communication models.

Our contribution in this work is that we present a formal test for interoperability of agents. The rest of the paper is organized as follows. Section 2 presents agents as transition systems. Section 3 formalizes a test for the interoperability of agents. Section 4 concludes with a discussion of the relevant literature.

2 Agents

We represent agents as transition systems. Informally, a transition system is a graph with states as vertices and actions as edges. A state s is labeled with the propositions that hold in that state; a transition is a triple $\langle s, e, s' \rangle$ where s and s' are states, and the edge e is labeled with the actions that occur in the transition. In addition, the initial and final states are marked. Further, the initial state has no incoming transitions.

Definition 1. *A transition system is a tuple $\langle \sigma^{fl}, \sigma^{act}, \eta, S, s_0, F, \rho, \delta \rangle$ where*

- *σ^{fl} is a finite set of propositions*
- *σ^{act} is a finite set of actions*
- *$\eta : \sigma^{act} \mapsto \sigma^{fl}$ is a bijective function*
- *S is a finite set of states*

– $s_0 \in S$ *is the initial state*
– $F \subseteq S$ *is the set of final states*
– $\rho : S \mapsto \mathcal{P}(\sigma^{fl})$ *is an injective labeling function with the requirement that* $\rho(s_0) = \{\}$
– $\delta \subseteq S \times E \times S$ *is the set of transitions where* $E \subseteq \mathcal{P}(\sigma^{act})$ *such that*
  - $\forall s \in S : \langle s, \epsilon, s\rangle \in \delta$ *where* ϵ *is the empty set of actions*
  - $\forall s, s' \in S : \langle s, \epsilon, s'\rangle \in \delta \Rightarrow s = s'$
  - $\langle s, \{a_0, a_1, \ldots, a_n\}, s'\rangle \in \delta \Rightarrow \rho(s') = \rho(s) \cup \bigcup_{i=0}^{n} \eta(a_i)$

The following description explains the elements of Definition 1. The empty set of actions ϵ corresponds to inaction. For each state s, the set of transitions δ contains the transition $\langle s, \epsilon, s\rangle$ to capture the transition where no action happens. Further, as would be expected, inaction cannot cause a transition in a new state. To capture the occurrence of any action a in the transition, the resulting state is labeled with a unique proposition $\eta(a)$ corresponding to the action. We restrict the transition systems such that the only cycles allowed are those because of inaction. This restriction is placed because η returns the same proposition no matter how many times an action happens, and thus is insufficient to model repeated actions. The propositions a state is labeled with serve as a history of all the actions that have occurred previously.

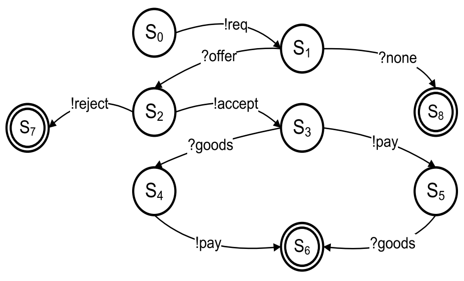

Fig. 1. A customer agent

State	Fluents
s_0	$\{\}$
s_1	*req*
s_2	*req, offer*
s_3	*req,offer,accept*
s_4	*req, offer, accept, goods*
s_5	*req, offer, accept, pay*
s_6	*req, offer,accept, goods, pay*
s_7	*req, offer,reject*
s_8	*req, none*

Table 1. States in Figure 1

We model two types of actions in agents: sends and receives. Let p be a message. A send is indicated by $!p$, whereas a receive is indicated by a $?p$. The sender and receiver of the message are implicit as the agents under consideration can interact only with one agent at a time. Figure 1 shows the transition system of a customer agent. State s_0 is the start state of this agent; the final states are indicated by concentric circles—in this case they are s_6, s_7, and s_8. This agent can interact with a merchant agent to buy goods. The customer's interactions are described below. Further, we assume that the names of the messages that can be sent by any agent are disjoint from those that any other agent can send.

1. The customer starts the interaction by sending a request for quotes to the merchant.
2. The merchant can respond either by sending an offer, or by indicating that there are no offers in which case the customer terminates.

3. If the merchant sends an offer, the customer can respond to the offer by either sending an accept, or a reject in which case, the customer terminates.
4. After the customer accepts, either the customer may send payment or the merchant may send goods.
5. If the merchant sends goods then the customer sends payment; if the customer sends payment, then the merchant sends goods. In either case, after the exchange the customer terminates.

Table 1 shows the labels of states in the transition system.

3 Interoperability

This section formalizes interoperability, and provides a computational method of verifying the interoperability of two agents.

Interoperability depends crucially on the communication model in force. Communication models may differ along the following dimensions.

Synchrony The communication mode is synchronous if an agent can send a message only when another is ready is receive it; equivalently, the send of each message coincides with its receipt. The result is that the agents execute in lock-step fashion. The mode is asynchronous if an agent can send a message regardless of the recipient's availability. The mode of communication has important implications for buffer design, as we shall see.

Channels Channels represent the logical communication medium between agents along which messages are exchanged. A channel can be unidirectional or bidirectional. If it is unidirectional, then it is modeled with a single buffer; if it is bidirectional, then it is modeled with two buffers, one for each direction. A unidirectional channel has two endpoints: one for the sending agent, and another for the receiving agent. A bidirectional channel has four endpoints: two for each agent—one to send, another to receive. Further, the number of channels may vary. For instance, all messages can be exchanged along a single channel, or each message can be exchanged along its own channel. More channels allows for greater concurrency.

Buffers Synchronous communication corresponds to zero-length buffers, whereas asynchronous communication implies nonzero-length buffers. Further, in the asynchronous model buffers may be finite-length or unbounded. Buffers may also differ in how they are accessed. A buffer may be modeled as a FIFO queue, in which case messages are appended to the end of a queue when doing a send, and read from its head when doing a receive. Alternatively, buffers may be modeled as random access memory (RAM), in which case sent messages can be inserted into and read from any location. The sizes of buffers impacts the ways in which an agents can block. If the buffers are unbounded, an attempt to send always succeeds whereas if they are of finite length, then even an attempt to send may block. An attempt to receive, on other hand, may block regardless of buffer size—for an agent to receive a message, another agent must have sent it first.

The proposed definition of interoperability declares two agents to be interoperable provided from each joint state (in the product) that they can enter, they can reach a final state. The essential idea is of determining the states that can be entered. In our approach, these are specified based upon the highly realistic constraint that only messages that have been sent (by an agent) can be received (by another agent). Based on this constraint, some transitions cannot in fact be performed: these transitions correspond to an agent receiving a message before the message has been sent. The transitions that can be performed are termed *causally enabled*. Further, for progress to take place, our definition assumes that if an enabled transition is available then that or another enabled transition is taken.

Operationally, these assumptions can be readily realized in agents that function as follows:

– The agents can perform nonblocking reads on the channels. Thus no agent is stuck attempting to make a transition that is not and will not be enabled.
– The agents try actions corresponding to their various transitions with some sort of a fairness regime. Thus if an agent can perform a send operation in a state, it will not forever stay in that state without performing the send. It may perform some other action to exit that state. Likewise, if an agent can read from a particular channel, it will not forever stay in that state without performing the read.

Although this paper is limited to systems consisting of two agents, it can be expanded to larger systems. For such systems, we would assume the following in addition to the above: the agents have unique incoming channels. That is, the agents do not compete for the messages arriving on their incoming channels.

3.1 Formalization

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 (agents can be globally concurrent even if each agent itself is single-threaded). Definition 2 captures the above intuitions for a product transition system of a pair of agents. For any two agents, we assume that their sets of actions as well as their sets of propositions are disjoint.

Definition 2. Given two agents $\alpha := \langle \sigma_\alpha^{fl}, \sigma_\alpha^{act}, \eta_\alpha, S_\alpha, s_{0_\alpha}, F_\alpha, \rho_\alpha, \delta_\alpha \rangle$ and $\beta := \langle \sigma_\beta^{fl}, \sigma_\beta^{act}, \eta_\beta, S_\beta, s_{0_\beta}, F_\beta, \rho_\beta, \delta_\beta \rangle$, their product is $\times_{\alpha,\beta} := \langle \sigma_\times^{fl}, \sigma_\times^{act}, \eta_\times, S_\times, s_{0_\times}, F_\times, \rho_\times, \delta_\times \rangle$ where,

– $\sigma_\times^{fl} = \sigma_\alpha^{fl} \cup \sigma_\beta^{fl}$
– $\sigma_\times^{act} = \sigma_\alpha^{act} \cup \sigma_\beta^{act}$
– $\eta_\times = \eta_\alpha \cup \eta_\beta$
– $S_\times = S_\alpha \times S_\beta$
– $s_{0_\times} = (s_{0_\alpha}, s_{0_\beta})$
– $F_\times = F_\alpha \times F_\beta$
– the labels on a state (s_α, s_β) is given by $\rho_\times(s_\alpha, s_\beta) = \rho_\alpha(s_\alpha) \cup \rho_\beta(s_\beta)$
– $\delta_\times \subseteq S_\times \times E_\times \times S_\times$ such that $\langle s, e, s' \rangle \in \delta_\times$ if and only if $\langle s_\alpha, e_\alpha, s'_\alpha \rangle \in \delta_\alpha$, $\langle s_\beta, e_\beta, s'_\beta \rangle \in \delta_\beta$ and $s = (s_\alpha, s_\beta)$, $s' = (s'_\alpha, s'_\beta)$, $e = e_\alpha \cup e_\beta$

The technical motivation behind Definition 2 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 sets of actions. When the agents act concurrently, the transitions are labeled by a pair of actions, one from each agent. Figure 2 shows two agents—one does $!x$, and the other $?x$—and their product. In this product, $00'$ is the initial state and $11'$ is a final state.

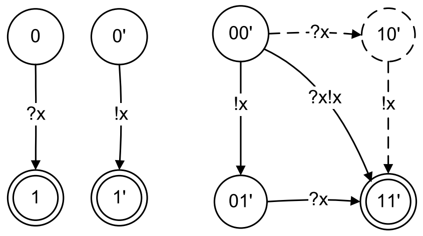

Fig. 2. Simple agents and their causal product (interoperable)

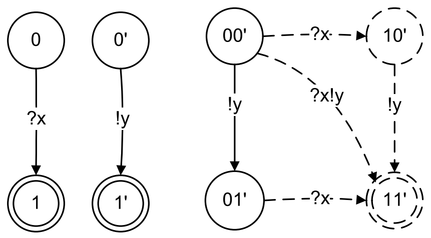

Fig. 3. Simple blocking agents and their causal product. Agents are noninteroperable because no causal fi nal state is reachable from state 01’

Figures 2–8 each contains three transition systems: one for agent α (identified by states labeled with one digit), one for agent β (identified by states labeled with one digit followed by an apostrophe as in $0'$), and their product (identified by states that contain states labeled with two digits—the second with an apostrophe). The start states of the two agents are indicated by 0 and $0'$ respectively. The final states are represented by concentric circles.

Our communication model is one in which agents communicate asynchronously over a bidirectional channel and each agent’s buffer is bounded RAM. As explained earlier, the state of an agent serves to capture the history of actions, and since each action can only occur once, the size of an agent’s buffer is bounded by the size of its set of actions. For the same reason, an attempt to send a message never blocks. A joint state in a product represents the union of both agents’ buffers. The receipt of a message fails if it is attempted before the message is sent. Definition 3 captures these observations formally in terms of causal enablement. Specifically, a state enables a transition if all

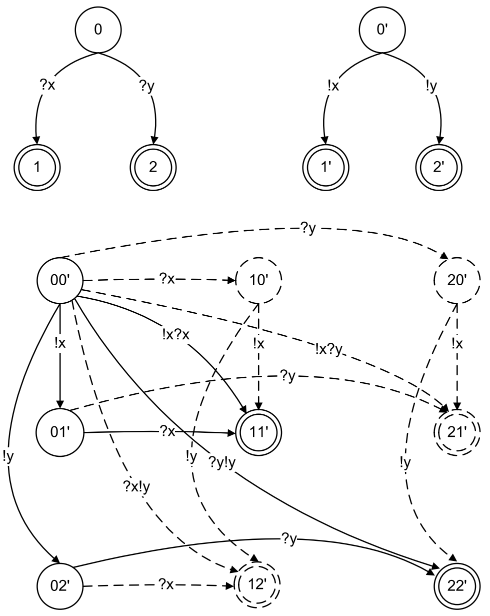

Fig. 4. Agents with a symmetric choice and their product (interoperable)

the actions listed in the transition succeed. Because there is only one channel, it is left implicit in the definition.

Definition 3. Given a transition $\langle s_i, e_i, s_{i+1}\rangle \in \delta_\times$ in a product $\times_{\alpha,\beta} := \langle \sigma^{fl}_\times, \sigma^{act}_\times, \eta_\times, S_\times, s_{0_\times}, F_\times, \rho_\times, \delta_\times\rangle$, s_i causally enables e_i, denoted by $s_i \models_{ce} e_i$ if and only if

$$e_i = \epsilon \text{ or,}$$

$$\forall ?p \in e_i(\eta(!p) \in s_i \text{ or } !p \in e_i)$$

.

Definition 3 means that a transition is enabled if for each receive attempted in it, a corresponding send has been performed previously or is being performed concurrently. For example, in Figure 2

$$01' \models_{ce} \{?x\}$$

$$00' \not\models_{ce} \{?x\}.$$

In Figures 2–8, the solid transitions are causally enabled whereas the dotted ones are not causally enabled. Definition 4 says that if a state is reachable from the initial state by causally enabled transitions, then it is causal.

Definition 4. The set of *causal* states in a product is defined as follows:

(i) s_0 is causal,

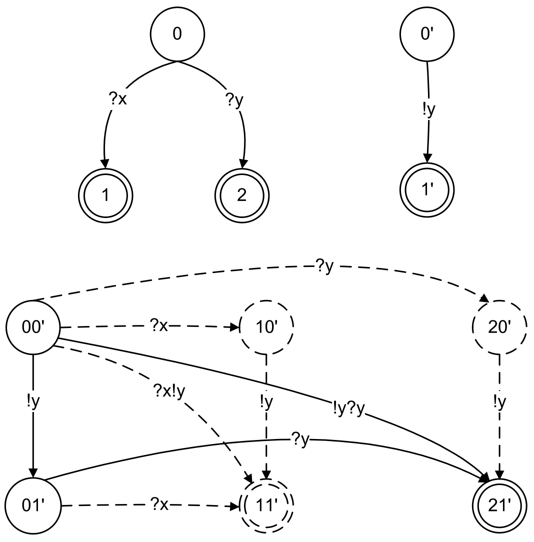

Fig. 5. Agents with limited send choice and their product (interoperable)

(ii) s' is causal if $\exists\langle s, e, s'\rangle \in \delta_\times$: s is causal and $s \models_{ce} e$,
(iii) all states that are not causal according to the above are noncausal.

In Figures 2–8, causal states are indicated with solid circles, whereas the noncausal states are indicated with dashed circles.

Definition 5 say that two agents α and β are interoperable if and only if for each causal state s, there exists a final state that is reachable from s through causally enabled transitions. Note that the definition does not simply state that there must exists some causally enabled path from the start state to some final state in the product for two agents to be interoperable. It is stronger than that. The definition reflects the fact there might be no causally enabled transition in the middle of an interaction, in which case the agents are determined noninteroperable. We introduce a function $causal$ which takes a product state and returns true if and only if the state is causal.

Definition 5. Let $\times_{\alpha,\beta} := \langle \sigma_\times^{fl}, \sigma_\times^{act}, \eta_\times, S_\times, s_{0_\times}, F_\times, \rho_\times, \delta_\times\rangle$ be the product of two agents α and β. Agents α and β are interoperable if and only if

$$\forall s_i : causal(s_i)(\exists\langle s_i, e_i, s_{i+1}\rangle, \langle s_{i+1}, e_{i+1}, s_{i+2}\rangle, \ldots, \langle s_{n-1}, e_{n-1}, s_n\rangle :$$

$$\forall j : (i \leq j \leq n)(causal(s_j) \text{ and } s_n \in F_\times))$$

.

Each pair of agents in the Figures 2–8 is labeled interoperable or noninteroperable based upon the test in Definition 5. The agents in Figure 2 are interoperable as one agent sends x and the other receives x. The agents in Figure 3 are noninteroperable as one agent can only send y, whereas the other may only receive x. Computationally, the

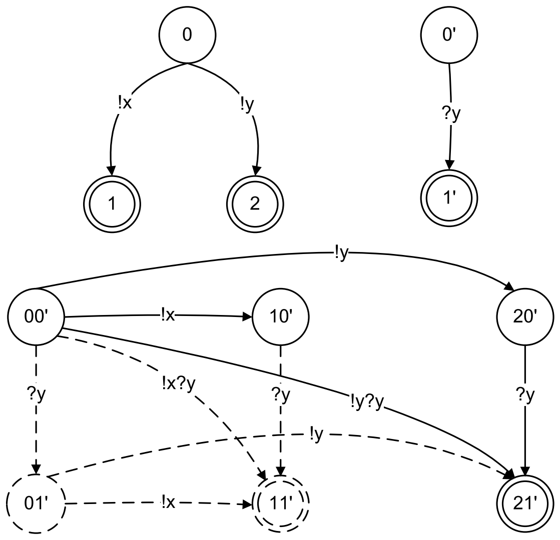

Fig. 6. Agents with no receive choice and their product. Agents are noninteroperable because no fi nal state is reachable from state 10' which itself is causal

problem state is 01', which is a causal state, but no causal final state is reachable from it. The agents in Figure 4 are interoperable—realistic as given the nonblocking receive semantics outlined earlier, eventually some message will be received. In Figure 5, one agent can receive x or y, but the other can only send y. These agents are interoperable and again, realistically so—because of the nonblocking semantics, eventually the receipt of y will succeed. Figure 6 shows two agents, one of which can send either x or y, and the other may only receive y. These agents are noninteroperable because the sending agent may choose to send x, in which case the other agent will never progress to a final state. Computationally, the problem state is 10', a causal state from which no causal final state is reachable. Figure 7 shows two agents, one of which sends x followed by y, whereas the other attempts to receive them in the opposite order, that

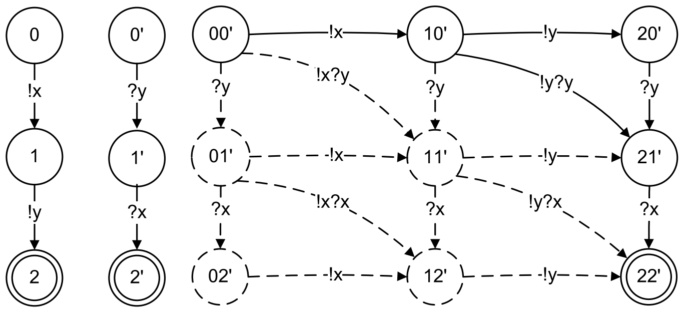

Fig. 7. Agents with out-of-order receives and their product (interoperable)

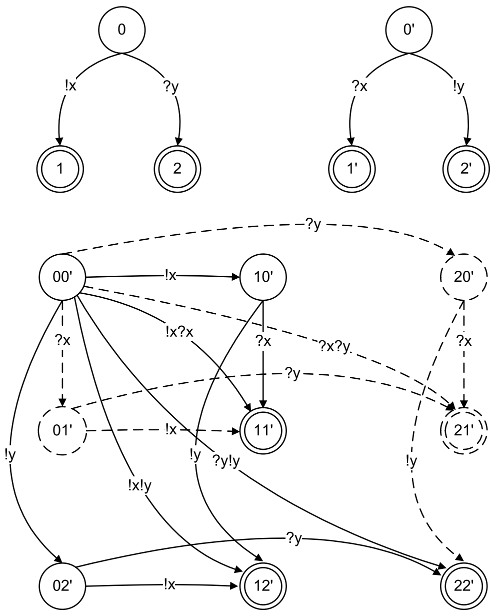

Fig. 8. Agents making nonlocal choice and their causal product (interoperable)

is y followed by x. Since our definition of causal enablement models a random access buffer, the agents turn out to be interoperable. Finally, Figure 8 shows two agents, one of which may either receive x or send y, whereas the other may send x or receive y. These agents turn out to be interoperable, which is supported by our operational assumptions.

4 Discussion

Interoperability is a crucial aspect of compliance; however, it is not the only one. Conformance is another aspect of compliance. For an agent to produce only compliant executions, it has to be both conformant with some stated protocol, and interoperable with the agent it is interacting with. Conformance has been formalized in previous work [4]. In this paper, we have devised a formal interoperability test for agents. The interoperability test is essentially a reachability test for final states in an appropriately marked product transition system.

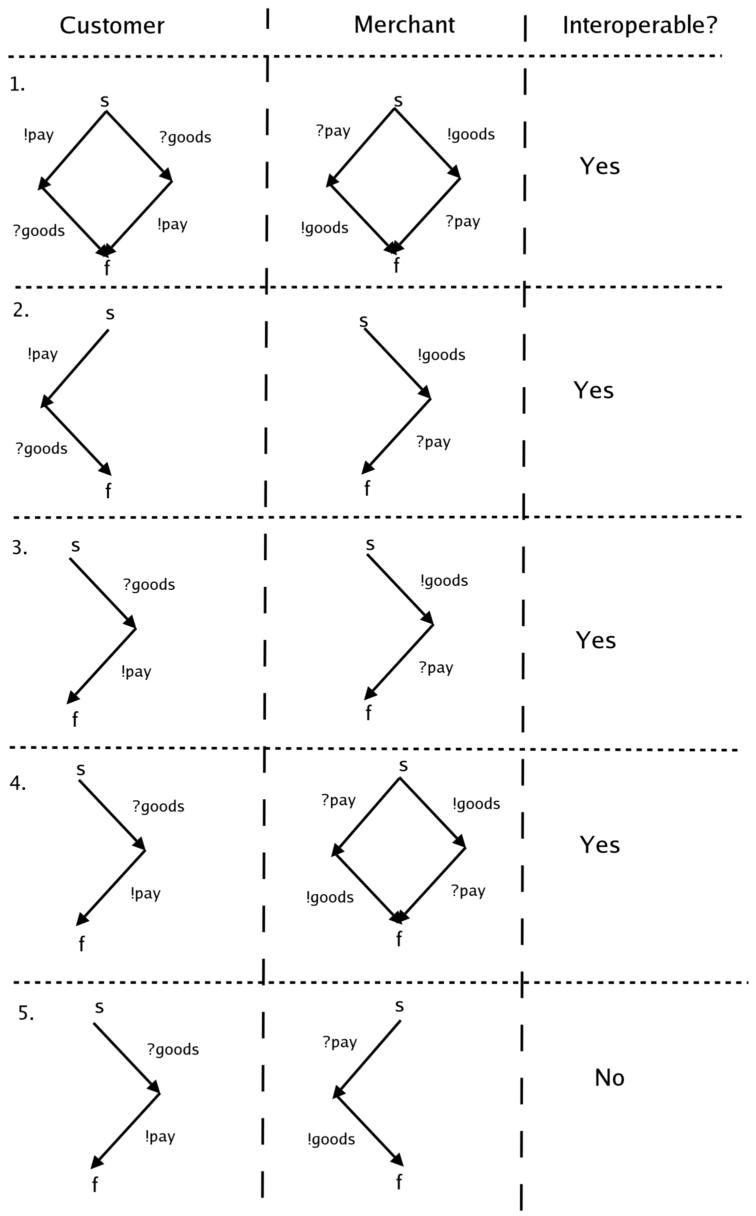

Fig. 9. Variations of customer and merchant agents and their interoperability

4.1 Temporal Logic

We assume the standard branching time temporal logic. Given the product of two agents, computed as above, the agents are interoperable if and if the following formula is true.

$$\mathrm{AG}(causal \rightarrow \mathrm{E}(causal\,\mathrm{U}\,(final \ \wedge causal)))$$

The proposition $causal$ is true if and only if a state in the model is causal, and $final$ is true if and only if a state belongs to the set of final states.

4.2 Blocking Receives

It is interesting to consider what happens when we drop the assumption that receives are nonblocking, and instead assume that they are blocking. In that case, the above definition of interoperability is optimistic: it makes sense only under the assumption of angelic nondeterminism. That is, if there is a possibility of reaching some joint final state, then the agents will magically take only those actions that necessarily take them there. In essence, the choices are not made by agents, but are made automatically for them. For instance, in Figure 5, our test for interoperability determines the agents interoperable. However, note they are not interoperable in case agent α decides to receive x. They are only interoperable under the assumption that an agent reads whatever is available in its end of the channel. For all practical purposes, at the level of abstraction of the transition system, a “good” choice was made nondeterministically. To consider another example, see Figure 8. If α and β choose to do $?y$ and $?x$ respectively—no matter what, they must each receive first before sending—then the agents would deadlock. (In distributed computing, this problem is commonly referred to as the nonlocal choice problem [14]). But again the choice to send or receive is not made by the agents; it is simply made for them nondeterministically.

The question then is: in the operational assumption that receives are blocking, how do we build agents in practice when angelic nondeterminism is not around to help make choices? The answer is: by encoding additional knowledge required to make the choices in the agent’s design. Figure 9 shows variations of the customer and merchant agents with a focus on only the goods-payment exchange. Each row in the Figure depicts a customer-merchant pair and states their interoperability. For each transition system, the labels s and f indicate the start and final states respectively; other states are not made explicit. The interesting thing is that only in case 5—when for both agents there is no choice but to receive first and thus deadlock—are the agents determined noninteroperable by Definition 5. In all other cases, in all stages of the interaction, there is always a causally enabled transition until a joint final state is reached. More importantly, in cases 1 and 4, the agents’ choices have to resolved so that they make compatible choices—which is where angelic nondeterminism helps us. In practice, however, the agents’ design would have to be amended so that the pairs of agents in cases 1 and 4 would resemble either the pair of case 2 or case 3. In previous work [3], we have specified agents in C+ [10], an action description language with a transition system semantics. C+ is elaboration-tolerant meaning that the addition of new axioms (knowledge) to a C+ theory could possibly invalidate old conclusions; in other words, transitions may be removed by adding new knowledge about the actions. This enables a designer to specify agents in C+ corresponding to the pair of case 1, and then depending upon the context in which the agents are to be deployed, the designer could add additional knowledge to turn them into either the pair of case 2 or case 3. For example, if the customer is not willing to pay first but the merchant is willing to send goods first, then the designer could turn them into the agents of case 3, whereas if they both trust each other, then the could turn them into case 2. An optimistic approach to component composability is that two components are composable if there exists some environment under which they can work together [6]. Under the assumption that receives are blocking, our definition of interoperability may be seen as an optimistic one: two agents can work together in practice, if they are interoperable by our definition, and if new axioms encoding additional knowledge may be appended to their specifications to make them work together.

4.3 Literature

Interoperability leads to a useful notion of compositionality: loosely speaking, two arbitrary agents can be composed if they can interoperate. Our formalization thus tells us which pairs of agents can be composed. We plan to extend our formalization to groups of agents.

Dastani *et al.* [5] describe an approach of composing multiagent systems by composing their coordination specifications, which are modeled as connectors. Omicini *et al.* [15] model composition of distributed workflows through programmable tuple spaces. However, none of these works delve into the question of which agents or workflows are composable.

Fu *et al.* [9] propose conditions for the realizability of protocols—a protocol can be realized if a set of finite agents can generate exactly the conversations in the protocol. Realizability of a protocol is orthogonal to interoperability between agents. A protocol may not be realizable, however agents that follow their respective roles in the protocol could be interoperable. On the other hand, interoperable agents could be following roles in distinct protocols.

Kazhamiakin *et al.* [11] construct a hierarchy of communication models depending on factors such as synchrony and buffers among other, and use this framework to find the best fit communication model for a given service composition such that the cost of further verification is cheapest. However, their method assumes that the services are composable under some model; they present no algorithm that decides composability.

Baldoni *et al.* [2] and Endriss *et al.* [7] present alternative notions of conformance that are closely tied to interoperability, thereby violating the orthogonality of conformance and interoperability. As a result, 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 a reminder to the merchant to send goods to be nonconformant.

Approaches based on verifying compliance at runtime [1, 16] 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.

4.4 Directions

Our current formalization supports a useful but limited class of agents: those that interact with only one other agent, and eventually terminate. Our future work involves extending this work to more general agents: specifically, those that interact with multiple other agents and have infinite runs.

References

1. M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specifi cation and verifi cation of agent interaction protocols in a logic-based system. In *Proceedings of the 19th ACM Symposium on Applied Computing*, pages 72–78, 2004.
2. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Verifi cation 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. A. K. Chopra and M. P. Singh. Protocol compliant interactions: Conformance, coverage, and interoperability. In *Declarative Agent Languages and Technologies IV: Selected, Revised, and Invited Papers*, volume 4327 of *LNAI*. Springer, 2006.
5. M. Dastani, F. Arbab, and F. de Boer. Coordination and composition in multi-agent systems. In *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 439–446, 2005.
6. L. de Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*, pages 109–120, 2001.
7. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *Proceedings of the 18th International Joint Conference on Artifi cial Intelligence*, pages 679–684, 2003.
8. N. Fornara and M. Colombetti. Operational specifi cation of a commitment-based agent communication language. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 535–542. ACM Press, July 2002.
9. X. Fu, T. Bultan, and J. Su. Conversation protocols: a formalism for specifi cation and verifi - cation of reactive electronic services. *Theoretical Computer Science*, 328(1-2):19–37, 2004.
10. E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artifi cial Intelligence*, 153(1-2):49–104, 2004.
11. R. Kazhamiakin, M. Pistore, and L. Santuari. Analysis of communication models in web service compositions. In *Proceedings of the 15th International Conference on World Wide Web*, pages 267–276, 2006.
12. 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)*, 14(2):143–163, 2006.
13. H. Mazouzi, A. E. F. Seghrouchni, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 517–526, 2002.
14. A. J. Mooij and N. Goga. Dealing with non-local choice in IEEE 1073.2's standard for remote control. In D. Amyot and A. W. Williams, editors, *System Analysis and Modeling: 4th International SDL and MSC Workshop, SAM 2004*, volume 3319 of *Lecture Notes in Computer Science*, pages 257–270, 2005.

15. A. Omicini, A. Ricci, and N. Zaghini. Distributed workflow upon linkable coordination artifacts. In P. Ciancarini and H. Wiklicky, editors, *Coordination*, volume 4038 of *Lecture Notes in Computer Science*, pages 228–246. Springer, 2006.
16. 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.
17. B. Vitteau and M.-P. Huget. Modularity in interaction protocols. In F. Dignum, editor, *Advances in Agent Communication*, volume 2922 of *Lecture Notes in Computer Science*, pages 291–309, 2004.
18. P. Yolum and M. P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534. ACM Press, July 2002.