Commitment Machines

Pınar Yolum and Munindar P. Singh
Interaction, Media, and Commerce Laboratory
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535, USA

{pyolum, mpsingh}@eos.ncsu.edu

Abstract

Protocols are structured interactions among communicating agents. Protocols are essential in applications such as electronic commerce where it is necessary to constrain the behaviors of autonomous agents. Traditional representations of protocols specify legal sequences of actions but define neither the content of the actions nor of the intervening states. Thus traditional representations are inadequate in open settings where autonomous agents must flexibly interact, e.g., to handle exceptions and exploit opportunities.

We develop an approach in which we model communication protocols via *commitment machines*. Commitment machines supply a content to the protocol states and actions in terms of the social commitments of the participants to one another. The content can be reasoned about by the agents thereby enabling flexible execution. We provide reasoning rules to capture the evolution of commitments through the agents' actions. Because of its representation of content and its operational rules, a commitment machine effectively encodes a systematically enhanced version of the original protocol, which allows the original sequences of actions as well as other legal moves to accommodate exceptions and opportunities.

Next we show how a commitment machine can be compiled into a finite state machine for efficient execution. We motivate and describe technical conditions under which our compilation procedure yields a finite state machine that is *deterministic* (easy to execute), *sound* (never produces a computation not allowed by the commitment machine), and *complete* (can produce the effect of any computation allowed by the commitment machine).

1 Introduction

Key agent applications in open environments rely crucially upon communication among interacting participants (Fisher & Wooldridge, 1997). Protocols streamline this communication and enable the development of interoperable software components by independent vendors. Unfortunately, current techniques for specifying and enacting protocols result in protocols that are more rigid than traditional, human-oriented protocols. This rigidity subverts some of the motivations behind why agents are attractive application in the first place.

In ideal circumstances, agents participating in a protocol would be able to identify and take valid shortcuts, exploit opportunities offered by the environment, negotiate and benefit from special relationships with other agents, and handle certain kinds of exceptions. Agents that are forced to follow rigid protocols cannot readily exercise any of the above possibilities and cannot adapt their behavior in light of their interactions with one another.

We develop an approach that enables the flexible execution of protocols. This approach is based on the following key conceptual definitions. *Agents* are persistent computations that can perceive, reason, act, and most importantly *communicate*. Agents can be autonomous and heterogeneous, and can represent different interacting components. The agents enter into *social commitments* with other agents. The agents' communications affect, and are affected by, their commitments. In turn, the agents' commitments reflect the protocols they are following and the communications they have already made or received. There are three main contributions of this paper.

- We propose a new formalism, the *commitment machine (CM)*, to formally specify and execute protocols. A commitment machine attaches specific *declerative* meanings to states and actions within a protocol. These meanings are based on commitments of the participating agents. When the meanings of states and actions are formally defined, the legal computations can be logically inferred. This enables the protocols to be systematically enhanced with additional transitions, allowing a broader range of interactions. The enhancement enables agents to exploit opportunities and handle exceptions.

- We show how a commitment machine may be automatically compiled into a finite state machine (FSM) in which no commitments or other declerative meanings are explicitly mentioned, but which can be efficiently executed.

- We give technical results proving that the compilation procedure, sometimes with additional restrictions, produces an FSM that is *deterministic* (easy to execute), *sound* (never produces a computation not allowed by the commitment machine), and *complete* (can produce the effect of any computation allowed by the commitment machine).

The rest of this paper is organized as follows. Section 2 describes our technical motivations in greater detail. Section 3 explains the technical background dealing with communications. Section 4 introduces commitment machines and show how they may be applied. Section 5 shows how commitment machines can be compiled into finite state machines and establishes important results regarding the compilation procedure. Section 6 describes our contributions with respect to the most relevant literature, and some future directions.

2 Technical Motivation

Our approach applies to multiagent protocols in general. However, for concreteness, we consider e-commerce as a running theme and study an e-commerce protocol as a running example in this paper. Multiagent protocols have traditionally been modeled by formalisms similar to those used to specify protocols in distributed computing and computer networks. Although these protocol specifications in general model communication among entities, traditional formalisms cannot adequately represent the interactions required in open applications such as e-commerce where agents are most useful. Consequently, protocols tend to suffer from unnecessary rigidity in execution, resulting in redundant interactions and avoidable failures. We now investigate the desired properties of the agents and the underlying components that our protocol representation must preserve. Next we analyze a well-known e-commerce payment protocol to understand what enhancements we can seek through our approach.

2.1 Dynamic Properties

Current formalisms used in modeling networking protocols, such as finite state machines and Petri Nets, specify protocols merely in terms of legal sequences of actions without regard to the meanings of those actions. For this reason, protocols tend to be over-constrained to the level of specific sequences of actions. However, protocols should not only constrain the actions of the participants, but also recognize the open, dynamic nature of interactions by accommodating the key aspects of autonomy, heterogeneity, opportunities, and exceptions.

- *Autonomy:* Enabling participants to exercise autonomy as much as they can in the social environment in which they are situated is a crucial factor in creating effective multiagent systems. The components should be able to exercise autonomy in deciding what actions they want to perform, who they want to interact with, or how they want to carry out their tasks. Thus, in an e-commerce setting, components must be constrained in their interactions only to the extent necessary to carry out the given protocol and may negotiate and adapt their behavior as they like.

- *Heterogeneity:* Agents can be diverse and may adopt different strategies in their interactions. For example, in many e-commerce protocols, all participants are assumed to be untrustworthy, and each step of the protocol ensures that appropriately safe actions are taken by the various participants. This assumption degrades the performance of the protocols since the actions could be better selected based on who the agent is dealing with. In particular, agents that have established trust with each other could safely skip certain steps in the given protocol.

- *Opportunities:* Agents should be able to take advantage of opportunities to improve their choices or to simplify their interactions. Depending on the situation certain steps in a protocol can be skipped. An agent may take advantage of domain knowledge, and jump to a state in a protocol without explicitly visiting one or more intervening states, since visiting each state may require additional messages and cause delays.

- *Exceptions:* Agents must be able to modify their interactions to handle unexpected conditions. The sort of exceptions of interest here are not programming or networking exceptions such as loss of messages and network delays, but are higher-level exceptions that result from the unexpected behavior of the participants.

2.2 Semantic Analysis

We now analyze the concepts and challenges underlying communication protocols from the standpoint of open environments. As a running example, we consider the NetBill protocol which was developed to handle the buying and selling over the Internet of electronic goods, such as software and electronic documents (Sirbu, 1998).

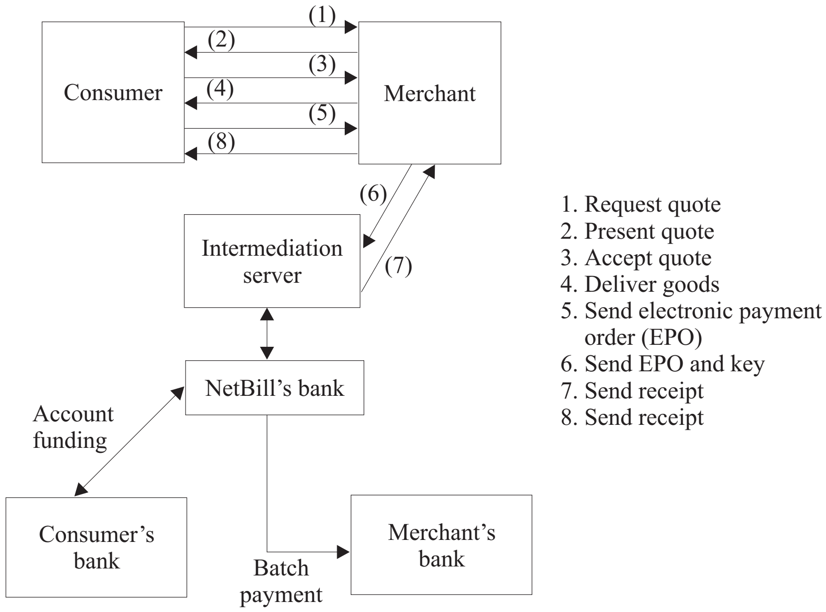

Figure 1: The NetBill payment protocol

Example 1 As shown in Figure 1, the protocol begins with a customer requesting a quote for some desired goods, followed by the merchant sending the quote. If the customer accepts the quote, then the merchant delivers the goods and waits for an electronic payment order (EPO). The goods delivered at this point are encrypted, that is, not usable. After receiving the EPO, the merchant forwards the EPO and the key to the intermediation server, which then contacts the bank to take care of the funds transfer. When the funds transfer completes, the intermediation server sends a receipt back to the merchant. The receipt contains the decryption key for the sold goods. As the last step, the merchant forwards the receipt to the customer, who can then successfully decrypt and use the goods.

For our present purposes, we can abstract out the details of the transactions that take place among the banks and the underlying security and encryption mechanisms. Therefore, we simplify the protocol by assuming that once the merchant receives an EPO, the funds transfer will succeed. Figure 2 shows this simplified version of the NetBill protocol. We use this simplified version as our running example. We later discuss how to add some of the details back to the protocol when desired.

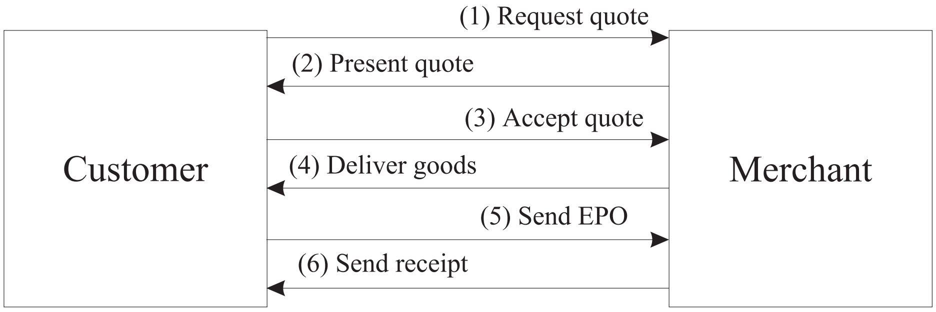

Figure 2: Simplified version of the NetBill protocol

The parties participating in an e-commerce protocol are self-interested and eager to interact in any way that would benefit them. Thus, the parties should be permitted a choice of actions, and be able to select the actions that benefit them the most.

Example 2 The rigid specification of Figure 2 cannot handle some of the natural situations that arise in e-commerce:

- Instead of waiting for a customer to request a quote, a merchant may proactively send a quote, mimicking the idea of advertising.

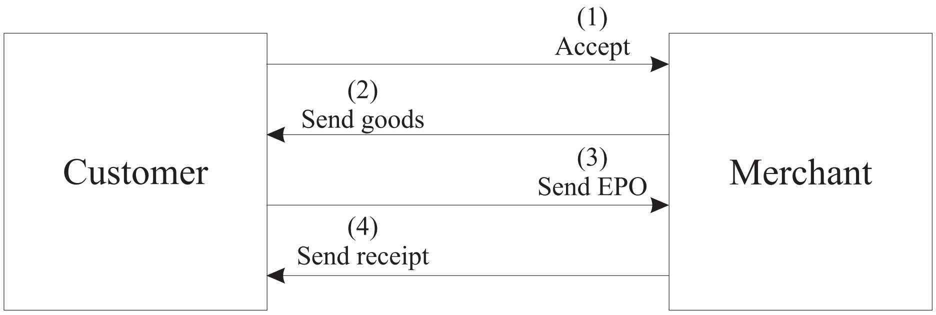

Figure 3: Alternative execution where the customer accepts any price prior to receiving a quote

- The customer may send an "accept" message without first exchanging explicit messages about a price. This situation would reflect the level of trust the customer places in the merchant. If the customer trusts the merchant to give him the best quote, he may accept the price without a prior announcement or quote. Alternatively, this action could result from the customer's lack of interest in the price, the emergency of the transaction, the insignificance of money to the customer, and so on. This scenario is shown in Figure 3.

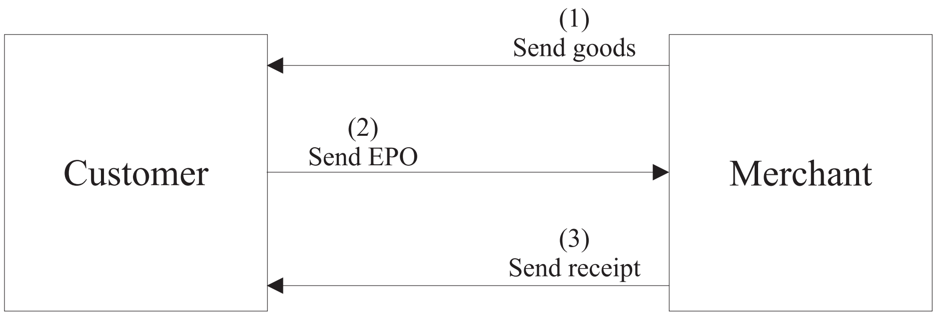

Figure 4: Alternative execution where the merchant sends the goods first

- As shown in Figure 4, a merchant may send the goods without an explicit price quote. Sending unsolicited goods would correspond to "try before you buy" deals, common in the software industry. Here after a certain period the customer is expected to pay to continue using the software.
- After receiving the goods, the customer may send the EPO to the bank instead of the merchant. By delegating the payment to the bank, the customer makes the bank responsible for ensuring that the money gets to the merchant.

3 Technical Framework

Our approach to accommodating flexible execution of e-commerce protocols combines ideas from social commitments and finite state machines. Accordingly, we introduce the key concepts from these areas here.

Following speech act theory, we view communication as a form of action (Austin, 1962). Specifically, by making an utterance, an agent may not only describe the current state of the world, but may also change it. The changes of interest here reflect the progress in the given protocol and may be captured in terms of modifications to the participants' commitments. Each utterance results in an action (such as creation or discharge) on one or more commitments.

The commitments are social in that they are directed from one participant to another and roughly correspond to a participant's obligations to another. By considering social rather than mental concepts, our approach diverges from traditional formalizations of speech act theory. In this regard, our approach is closer in spirit to the language-as-action approaches (Winograd & Flores, 1987).

3.1 Social Commitments

Social commitments are different from both internal and collective commitments. An internal commitment relates an agent and a particular action or a condition that it commits to bringing about. A collective commitment relates a group of agents to a particular action or condition that they will bring about (Castelfranchi, 1995). Social commitments, on the other hand, are commitments made from one agent to another agent to carry out a certain course of action (Singh, 1999).

Definition 1 A social commitment C*(x, y, G, p)* relates a debtor *x*, a creditor *y*, and a condition *p*, in the scope of a context group *G*. ∎

When a social commitment of this form is created, *x* becomes responsible to *y* for satisfying *p*. The context group *G* is the organization within which the commitment exists. Making the context explicit enables us to control the evolution of the commitments and to accommodate compliance and resolution of disputes. The condition *p* may involve relevant predicates and commitments, allowing the commitments to be nested or conditional. In contrast with databases, here the commitments are flexible and can be revoked or modified. Almost always, the revocation or modification is constrained through *metacommitments*.

Definition 2 A commitment $c = \mathsf{C}$*(x, y, G, p)* is *base-level* if p does not refer to any other commitments; c is a *metacommitment* if p refers to a base-level commitment. ∎

Viewing a commitment as an abstract data type, the creation and the manipulation of the commitments can be described using the following operations (Singh, 1999; Venkatraman & Singh, 1999). Here, x, y, z denote agents, and c and c' denote commitments of the form C*(x, y, G, p)*.

1. *Create(x, c)* establishes the commitment c. The *create* operation can only be performed by the debtor of the commitment.

2. *Discharge(x, c)* resolves the commitment c. Again, the *discharge* operation can only be performed by the debtor of the commitment to mean that the commitment has successfully been carried out. Thus, after the *discharge* operation the condition p starts to hold.

3. *Cancel(x, c)* cancels the commitment c. Usually, the cancellation of a commitment is followed by the creation of another commitment to compensate for the former one.

4. *Release(y, c)* or *Release(G, c)* releases the debtor from the commitment c. It can be performed either by the creditor or the context group, to mean that the debtor is no longer obliged to carry out his commitment.

5. *Assign(y, z, c)* eliminates the commitment c, and creates a new commitment c' for which z is appointed as the new creditor.

6. *Delegate(x, z, c)* eliminates the commitment c, and creates a new commitment c' in which the role of the debtor is transferred to z.

3.2 Finite State Machines

One of the most common methods of specifying protocols is through the use of the deterministic finite state machines (DFSMs). DFSMs are also extensively used to specify e-commerce protocols.

Definition 3 A finite state machine is a five-tuple, $M = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$, where $\mathbf{S}$ is the set of states, Σ is the input alphabet, $s_0 \in \mathbf{S}$ is the start state, $\mathbf{Q} \subseteq \mathbf{S}$ is the set of final states, and $\delta \subseteq \mathbf{S} \times \Sigma \times \mathbf{S}$ is the transition relation. ▮

A protocol execution starts from the start state s_0. With every action performed, the execution moves to a new state based on the transition function. An FSM accepts a sequence of actions if the execution ends in one of the final states in $\mathbf{Q}$.

An FSM can be represented graphically by a labeled directed graph, where nodes represent the states and the arcs represent the transitions. In a protocol specification, the alphabet of the FSM is given by the set of possible actions. The transition relation describes how an execution of the protocol may progress as the agents perform legal actions resulting in new states.

Definition 4 A finite state machine $\langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ is *deterministic* if any action has at most one transition from a state, i.e., $(\forall s, s', s'' \in \mathbf{S}, a \in \Sigma : (s, a, s'), (s, a, s'') \in \delta \Rightarrow s' = s'')$. ▮

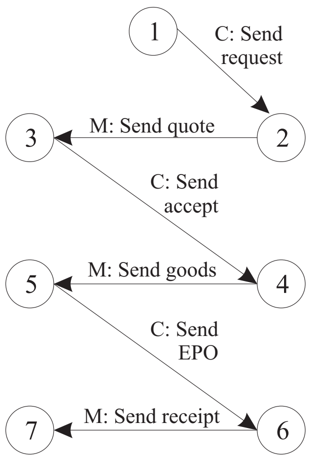

Figure 5: FSM representation of the NetBill protocol

Example 3 Figure 5 shows the simplified version of the NetBill payment protocol as an FSM labeled with the actions of merchant agent M and customer agent C.

4 Commitment Machines

FSMs are easy to operationalize but they hide the meanings of states and lead to rigid executions. This motivates us to define a commitment machine (CM) in terms of sets of states and actions that are given a declerative semantic content in terms of commitments. A CM specifies

- the possible states an executing protocol can be in.
- the actions that are used for a transition from one state to another.
- the possible final states of the protocol.

The meaning associated with each state specifies which commitments are in force in that particular state, and the meaning associated with each action defines how the commitments are affected by that action (thereby leading to a state change).

Like an FSM, a CM has a current state; zero or more actions are allowed in each state; every allowed action causes the CM to transition to a new state. Unlike an FSM, the representation of a CM does not specify a starting state. The participants may start the protocol from a state by accepting the commitments that are in force in that state. Usually, the protocol will have some states where no commitments are in force. A CM also has final states, which reflect the acceptable or desirable termination states of the protocol. Recall that *liveness* means that progress will be made. Starting from a particular state does not guarantee liveness.

Importantly, unlike in an FSM, the transitions between the states are not explicitly specified. The meanings of the states and the actions are logically represented. Based on the intrinsic meaning of the actions, the new state that is reached by performing an action at a particular state can be logically inferred. Thus, instead of specifying the sequences of actions that can be performed, a CM simply specifies the meanings that are legal in the protocol and, of these, the meanings that are final.

A CM specification of an e-commerce protocol emphasizes that the aim of executing the protocol is not merely to perform certain sequences of actions, but to reach a state that represents the result of performing these sequences of actions. With this in mind, we can come up with different paths that accomplish the same goal as the original path. Thus, a protocol can be reconfigured (enhanced or abbreviated) by finding alternative paths between states.

Example 4 We define the semantic content of each state in Figure 5 based on the participants' commitments:

- In state 3, having sent a quote to a customer, the merchant commits to delivering goods and sending a receipt afterwards, if the customer promises to pay.
- In state 4, having sent an accept to a merchant, the customer agrees to pay, but only if the merchant promises to send a receipt afterwards.
- In state 5, the merchant has fulfilled the first part of his promise by sending the goods.
- In state 6, the customer has discharged his commitment of sending the EPO.
- In state 7, the merchant has discharged his commitment of sending the receipt.

The CM specification of a protocol can be applied in two main ways.

- *Run time.* A CM specification of a protocol gives the states and the effects of performing the various actions. Given a CM, an agent that can process logical formulas can compute the transitions between states. In this respect, the choice of actions is a planning problem for each agent. That is, from the possible final states, the agent first decides on the desired final state, and then logically infers a path that will take it from the current state to the desired final state. Effectively, the agent interprets the CM directly at run time.

- *Compile time.* To reduce the computation required at run time, a CM can be compiled into an FSM that abstracts out the meanings of the states and actions. An FSM can be mechanically executed without explicit logical inference. This transformation is based on systematically producing paths between pairs of meanings. The meanings in the CM map to states in the FSM. The set of actions remains the same. The transitions between the FSMs states follow from the transitions in the CM. The set of final states will be the designated final meanings in the CM.

Section 4.1 formalizes CMs including the main reasoning rules for operationalizing commitments. Section 4.2 shows how CMs help capture an enhanced protocol. Section 4.3 shows how CMs support flexible execution.

4.1 Formalization

We now look more closely at how CMs can be applied in the above two ways. Our formalization is based on a language used to represent the legal meanings in a CM. Our formal language, $\mathcal{P}$, is based on the language of propositional logic with the addition of a commitment operator to represent commitments, and a *leads to* operator to capture strict implication.

The following Backus-Naur Form (BNF) grammar with a distinguished start symbol *Protocol* gives the syntax of $\mathcal{P}$. In this grammar, *slant* typeface indicates nonterminals; $\longrightarrow$ is a metasymbol of BNF specification; $\ll$ and $\gg$ delimit comments; $\{$ and $\}$ indicate that the enclosed item is repeated 0 or more times; the remaining symbols are terminals.

- *Protocol* $\longrightarrow$ $\{$*Action*$\}$ $\ll$set of actions$\gg$
- *Action* $\longrightarrow$ *Token*: L $\ll$token is a label; L is the associated meaning$\gg$
- *Commitment* $\longrightarrow$ $\mathsf{C}_x\ (L) \mid \mathsf{C}_x\ (M)$ $\ll$simplified as explained below$\gg$
- $L \longrightarrow$ *Commitment*
- $M \longrightarrow L \leadsto L$ $\ll$leads to, indicating a strict implication$\gg$
- $L \longrightarrow L \wedge L$ $\ll$conjunction$\gg$
- $L \longrightarrow \neg L$ $\ll$negation$\gg$
- $L \longrightarrow$ *Prop* $\ll$atomic propositions$\gg$

The logical operators are given the usual semantics. The strict implication, $p \rightsquigarrow q$, requires q to hold when p holds. Contrary to the material implication ($p \rightarrow q$), which is true when p is false, the strict implication is false if p is false. The strict implication is used in representation of participants' metacommitments. For expository ease, we use a simplified notation to represent commitments in the remainder of this paper. Here $\mathsf{C}_x p$ means that x (which can be the customer or the merchant in NetBill) is committed to the other party to carry out p (the group is implicit). Furthermore, we only restrict the nesting of commitments to one level.

Definition 5 A metacommitment is a commitment of the form $\mathsf{C}_x(p \rightsquigarrow \mathsf{C}_x r)$. This expresses the conditional commitment where if the proposition p becomes true, then the debtor x will become committed to bringing about r. Thus, $\mathsf{C}_x(p \rightsquigarrow \mathsf{C}_x r)$ is operationally the same as $\mathsf{C}_x(p \rightsquigarrow r)$ and we use the latter, simpler form in the remainder of this paper. ▮

The meaning of a state is given by any formula derivable from the nonterminal *L*. We define two logical relations among meanings: logical derivation and equivalance.

Definition 6 $p \vdash q$ means that q can be logically derived from p. ▮

Definition 7 $p \equiv q$ means that p and q are logically equivalent, that is, $p \vdash q$ and $q \vdash p$. ▮

A minimal set of meanings is one in which all meanings are logically distinct. A set of final meanings is consistent if it is well-behaved with respect to logical consequence.

Definition 8 A set $\mathbf{M}$ of meanings is *minimal* if and only if the following conditions hold:

- $(\forall m_i, m_j \in \mathbf{M}$: $(m_i \equiv m_j) \Rightarrow (m_i = m_j))$
- $\mathsf{true} \in \mathbf{M}$.
- $\mathsf{false} \notin \mathbf{M}$. ▮

Definition 9 A set of final meanings $\mathbf{F}$ is *consistent* with respect to a set of meanings $\mathbf{M}$ if and only if any meaning that is stronger than a final meaning is also final. That is, $(\forall m_i \in \mathbf{F}, m_j \in \mathbf{M}$: $(m_j \vdash m_i) \Rightarrow (m_j \in \mathbf{F}))$. ▮

Actions are represented as a pair whose first element is the token (name) of the action, and whose second element is the effect of the action.

Definition 10 $\langle a : e \rangle$ is an action, if a is the token and e is the meaning (effect) of the action. ▮

Definition 11 A CM is a triple $\langle \mathbf{M}, \Delta, \mathbf{F} \rangle$, where $\mathbf{M}$ is a finite minimal set of meanings, Δ is a finite set of actions defined in terms of commitments, and $\mathbf{F} \subseteq \mathbf{M}$ is a consistent set of final meanings. ▮

A CM transitions from meaning q to meaning r under action $\langle a : e \rangle$ if and only if after applying the effect e on q, r can be logically derived. We formalize the CM transitions as follows:

Definition 12 $q \models_{\langle a:e \rangle} r \triangleq (q \wedge e) \vdash r$. ▮

Thus, deriving the resulting meaning from a given meaning and action involves computing the logical consequence (that is, the $\vdash$ relation). For the propositional part of the language this is as usual. For commitments and metacommitments, we now present some important rules for reasoning about their consequences. These rules capture the operational semantics of our approach.

Reasoning Rule 1 A commitment $\mathsf{C}_x p$ ceases to exist when the proposition p becomes true. ▮

Reasoning Rule 2 A metacommitment $\mathsf{C}_x(p \rightsquigarrow r)$ ceases to exist when the proposition p becomes true, but a new base-level commitment $\mathsf{C}_x r$ is created to capture the fact that x has to satisfy the original metacommitment by bringing about the proposition r. ▮

Reasoning Rule 3 A metacommitment $\mathsf{C}_x(p \rightsquigarrow r)$ ceases to exist when the proposition r holds (before p is initiated), and no additional commitments are created. ▮

The following example illustrates the applicability of the above reasoning rules.

Example 5 Consider the metacommitment $\mathsf{C}_m(\mathit{pay} \rightsquigarrow \mathit{receipt})$, which denotes the commitment that the merchant is willing to send a receipt if the customer pays. After the creation of this metacommitment, the following scenarios may take place:

- The customer pays, making the proposition *pay* true. In this case, the metacommitment is terminated and a new commitment, $\mathsf{C}_m \mathit{receipt}$, is created in its stead (Reasoning Rule 2). When the merchant actually sends the receipt, i.e., when the proposition *receipt* becomes true, then the commitment $\mathsf{C}_m \mathit{receipt}$ is discharged (Reasoning Rule 1).
- Before the customer pays, the merchant sends the receipt, making the proposition *receipt* true. In this case, the metacommitment is terminated, but no other commitment is created since the customer did not commit to paying in the first place (Reasoning Rule 3).

The following example describes the atomic propositions and commitments used in the NetBill protocol, and then defines the NetBill protocol formally as a CM.

Example 6 Following Figure 5, the messages and the states can be given a content based on the following definitions:

- **Atomic propositions**
  - *request* ≪the customer has requested a quote.≫
  - *goods* ≪the merchant has delivered the goods.≫
  - *pay* ≪the customer has paid the agreed amount.≫
  - *receipt* ≪the merchant has delivered the receipt.≫

- **Abbreviations for the metacommitments**
  - *accept* ≪an abbreviation for $\mathsf{C}_c(goods \rightsquigarrow pay)$ meaning that the customer is willing to pay if he receives the goods.≫
  - *promiseGoods* ≪an abbreviation for $\mathsf{C}_m(accept \rightsquigarrow goods)$ meaning that the merchant is willing to send the goods if the customer promises to pay.≫
  - *promiseReceipt* ≪an abbreviation for $\mathsf{C}_m(pay \rightsquigarrow receipt)$ meaning that the merchant is willing to send the receipt if the customer pays.≫
  - *offer* ≪an abbreviation for $(promiseGoods \wedge promiseReceipt)$≫

Based on the above definitions, we can now define the NetBill protocol formally as a CM. Since each action can be performed by only one party, we do not specify the performers explicitly.

- **Meanings (M):**
  1. `true`
  2. *request* ≪The customer has requested a quote.≫
  3. *offer* ≪The merchant has made an offer.≫
  4. $\mathsf{C}_m goods \wedge accept \wedge promiseReceipt$ ≪The customer has accepted the merchant's offer.≫
  5. $goods \wedge \mathsf{C}_c pay \wedge promiseReceipt$ ≪The merchant has delivered the goods, and therefore the customer is committed to paying and the merchant is willing to send the receipt after the customer pays.≫
  6. $goods \wedge pay \wedge \mathsf{C}_m receipt$ ≪The merchant has delivered the goods, the customer has paid, and the merchant is committed to sending the receipt.≫
  7. $goods \wedge pay \wedge receipt$ ≪The merchant has delivered the goods and the receipt, and the customer has paid.≫

- **Actions (Δ):**
  1. $\langle sendRequest: request \rangle$ ≪sending a request for quote.≫
  2. $\langle sendQuote: offer \rangle$ ≪sending a quote.≫
  3. $\langle sendAccept: accept \rangle$ ≪sending an accept.≫
  4. $\langle sendGoods: goods \wedge promiseReceipt \rangle$ ≪delivering the goods.≫
  5. $\langle sendEpo: pay \rangle$ ≪sending an EPO.≫
  6. $\langle sendReceipt: receipt \rangle$ ≪sending the receipt.≫

- **Final meanings (F):**
  1. *request*
  2. *offer*
  3. $goods \wedge pay \wedge receipt$

4.2 Enhancement

One of the main advantages of specifying e-commerce protocols using CMs is their ease of enhancement. By adding new meanings to the meaning set, **M**, and appropriately, to the final meaning set, **F**, the protocol can be enhanced to allow additional computations. Similarly, by reducing the meanings set **M** and **F** the possible computations can be restricted. The following example depicts how an existing CM specification can be enhanced.

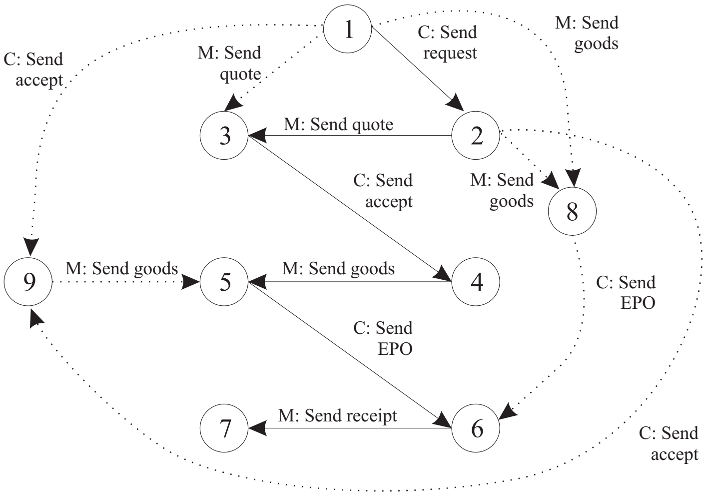

Figure 6: Enhanced commitment machine

Example 7 We now reconsider the shortcomings of Figure 2. In order to remedy these shortcomings, we introduce two new meanings, *accept* and (*goods* ∧ *promiseReceipt)*. The first meaning denotes the state where the customer is accepting any price, and the second meaning denotes the state where the merchant has delivered the goods and is willing to send the receipt if the customer pays. We now add these two new meanings to the meaning set, **M**, and the final meaning set, **F**, defined in an example in Section 4.1, the previous specification of our protocol.

Although the formal descriptions of CMs and FSMs are quite different, we can use the same pictorial representation for both. Thus, we use the graphical representation in Figure 6 to show the additional possible moves that can be achieved by enhancing the CM specification of the NetBill protocol.

- The merchant can now start the protocol by sending a quote to a customer. As was the case in Section 4.1, by performing this action the merchant creates a metacommitment in state 3, namely that he is willing to send the goods and the receipt if the customer agrees to pay.

- By sending an *accept* message without prior conversation about the quote, the customer commits to paying if the merchant makes an offer. Thus, we move from a state where no commitments exist (state 1) to a state where the customer is commited to pay if the merchant makes an offer (state 9). If the merchant does not make an offer, then the protocol ends at this point. On the other hand, if the merchant makes an offer by sending the goods, then both parties have to carry out their commitments, so the protocol moves to state 5.

- By sending the goods without an explicit *accept* message, the merchant makes an offer to send the receipt if the customer agrees to pay. Thus, we move from state 1 to a state where the merchant is making an offer (state 8). If the customer does not accept the offer, then the protocol ends at this point. Conversely, if the customer actually sends an *accept* message, then both parties need to fulfill their commitments, so the protocol moves to state 5.

Compared to the original version of the protocol in Figure 5, we have introduced two new states, state 8 and state 9, for the new added meanings.

- State 8: $goods \wedge promiseReceipt$
- State 9: *accept*

In addition to the above states, we have added new transitions from state 1 to states 3, 8, and 9; from state 2 to states 8 and 9; from state 8 to state 6 and from state 9 to state 5. The new transitions are shown with dashed lines in Figure 6.

No matter what path is chosen to reach a state, each state captures a well-defined meaning in terms of commitments. Once, a state is reached, the path that took us there can be discarded, making the choice of action independent of the previous actions performed. At any state, the reasoning can be based solely on the meaning associated with that state.

4.3 Flexible Execution of Commitment Machines

At this point, it is important to restate what we mean by flexibility. Although we want the agents to enact a flexible protocol, we still want to preserve an ordering that will allow only meaningful conversations. For example, a merchant should not send a quote after sending the goods, and the customer should not start the conversation by sending an EPO.

Importantly, flexibility can be introduced only to the point where the intended meanings of the actions are not violated. The interactions among the parties can be given a meaning based on their commitments to each other. When we allow more flexible interactions, we need to ensure that the original commitments are in force or that they are altered by mutual agreement. The execution of a protocol is then minimally constrained only to satisfy those metacommitments. This is a major advantage over low-level representations, which require specific execution sequences and provide no basis for deciding on the correct state independent of the execution sequence.

4.3.1 Opportunism

Depending on the circumstances, a participant may choose to start a protocol from the middle to take advantage of an opportunity. An opportunity in this context corresponds to a commitment-based meaning of a state that provides some convenience to the participant. By starting a protocol in this way, the agent accepts the commitments that are in force in that particular state. Similarly, during the execution of a protocol an agent may be able to jump across several steps at once. In our running example, the merchant may send a price quote proactively. We discussed such cases in Section 2.2.

4.3.2 Composition

Protocols may be combined into larger protocols, as long as one protocol ends in a commitment state that another protocol can accommodate. This can be applicable in two cases, first to combine protocols to be executed one after the other, and second to allow a side protocol to be followed during the execution of the first protocol.

A protocol can be composed by combining modules of subprotocols through a common commitment state. This common state need to be one of the final states of the first CM, and a possible state in the second CM. If the execution of the first protocol ends at the common commitment state, the execution can continue from that particular state present in the second protocol, provided that the execution of the protocol never goes back to a state in the first protocol.

Alternatively, participants of a protocol can interact with agents that are not taking place in the protocol through side protocols. In this case, the crucial point is to ensure that the agent that gets into a side protocol needs to returns to the point where it left off in the main protocol.

Example 8 In Section 2.2, we abstracted out the banking procedures to simplify the NetBill protocol, although there is a convention used between the merchant and the bank to deal with the EPO processing. If we consider the banking operations as a separate protocol, we can see that the banking protocol may be combined with our protocol through state 6. Recall that at state 6, the customer has discharged his commitment of sending an EPO, but the merchant still has to send the receipt. The merchant at this state can participate in a side protocol with a bank to verify that the EPO has actually cleared. After taking part in this side protocol, the merchant will come back to state 6 to send the receipt to the customer. Since the commitments of the main protocol are preserved in the side protocol, the composition of the two protocols yields a valid protocol.

5 Compiling Commitment Machines

A protocol specification that allows the above characteristics can drastically improve the flexibility and thus the quality of the solution provided by agent-based applications. However, the flexibility comes at the price of reasoning with declerative representations at run-time, which can be expensive and may increase the code footprint of the agents who implement such reasoning. Fortunately, it is possible to compile a CM into an FSM so that the desired affect can be obtained without representing and reasoning about declerative meanings at run time.

Let us consider the requirements of compiling a CM into an FSM. For an agent to be able to directly execute the resulting FSM, we seek an FSM that is deterministic and with no irrelevant transitions. Given a CM, the states of the FSM can be generated from the meanings of the CM.

However, the execution of the FSM follows the usual regime of state-transition-state without regard to the formulas that exist in CM meanings on which the FSM states are based.

To infer the valid transitions in the FSM, we consider the possible entailments between the meanings. To ensure determinism and efficiency, we constrain the allowed transitions with two major restrictions.

Restriction 1 $\langle m_i, a, m_j \rangle \in \delta$ entails that $m_i \not\vdash m_j$.

If the source meaning already entails the content captured in the target meaning, no transition from the source to the target is necessary. This restriction ensures that the meaning that will be reached is not already captured at the current state, and that the FSM has no transitions that do not add to the content. ▮

Restriction 2 $\langle m_i, a, m_j \rangle \in \delta$ entails that ($\forall\ m_k \in$ **M**: $m_i \models_{\langle a:e \rangle} m_k \Rightarrow m_j \vdash m_k$).

This is to ensure that if applying an action at a particular meaning entails several possible meanings, then the transition will end in a state that contains the maximal information. Later we will restrict our CMs so that a maximal meaning always exists. ▮

Example 9 Let us assume that the meanings depicted in Figure 7 constitute the meaning set **M** of a CM (it is easy to verify that **M** is minimal). Among these four meanings, **F** contains m_2 and m_3. Let the action set Δ contain two actions: $\langle$*sendaccept*: *accept*$\rangle$ and $\langle$*sendoffer*: *offer*$\rangle$.

We now look at some possible entailments and demonstrate which of these entailments result in a valid transition converting a CM to an FSM.

- $\langle m_0$, *sendoffer*, $m_1\rangle$: `true` $\models_{\langle sendoffer:offer \rangle}$ *offer*

  $\langle m_0$, *sendaccept*, $m_2\rangle$: `true` $\models_{\langle sendaccept:accept \rangle}$ *accept*

  Since neither restriction applies to these entailments, both transitions are accepted.

- $\langle m_1$, *sendoffer*, $m_1\rangle$: *offer* $\models_{\langle sendoffer:offer \rangle}$ *offer*

  $\langle m_1$, *sendaccept*, $m_1\rangle$: *offer* $\models_{\langle sendaccept:accept \rangle}$ *offer*

  These two transitions will not be allowed due to Restriction 1, which does not allow transitions where the starting state already entails the end state.

  $\langle m_1$, *sendaccept*, $m_2\rangle$: *offer* $\models_{\langle sendaccept:accept \rangle}$ *accept*

  $\langle m_1$, sendaccept, $m_3\rangle$: *offer* $\models_{\langle sendaccept:accept \rangle}$ (*offer* $\wedge$ *accept)*

  Based on Restriction 2, only the latter transition will be allowed.

- $\langle m_2$, *sendaccept*, $m_2\rangle$: *accept* $\models_{\langle sendaccept:accept \rangle}$ *accept*

  $\langle m_2$, *sendoffer*, $m_2\rangle$: *accept* $\models_{\langle sendoffer:offer \rangle}$ *accept*

  Again, based on Restriction 1, these two transitions will not be allowed.

  $\langle m_2$, *sendoffer*, $m_1\rangle$: *accept* $\models_{\langle sendoffer:offer \rangle}$ *offer*

  $\langle m_2$, *sendoffer*, $m_3\rangle$: *accept* $\models_{\langle sendoffer:offer \rangle}$ (*offer* $\wedge$ *accept*)

  The only transition that will be allowed will again be the latter, since it yields more information than the former.

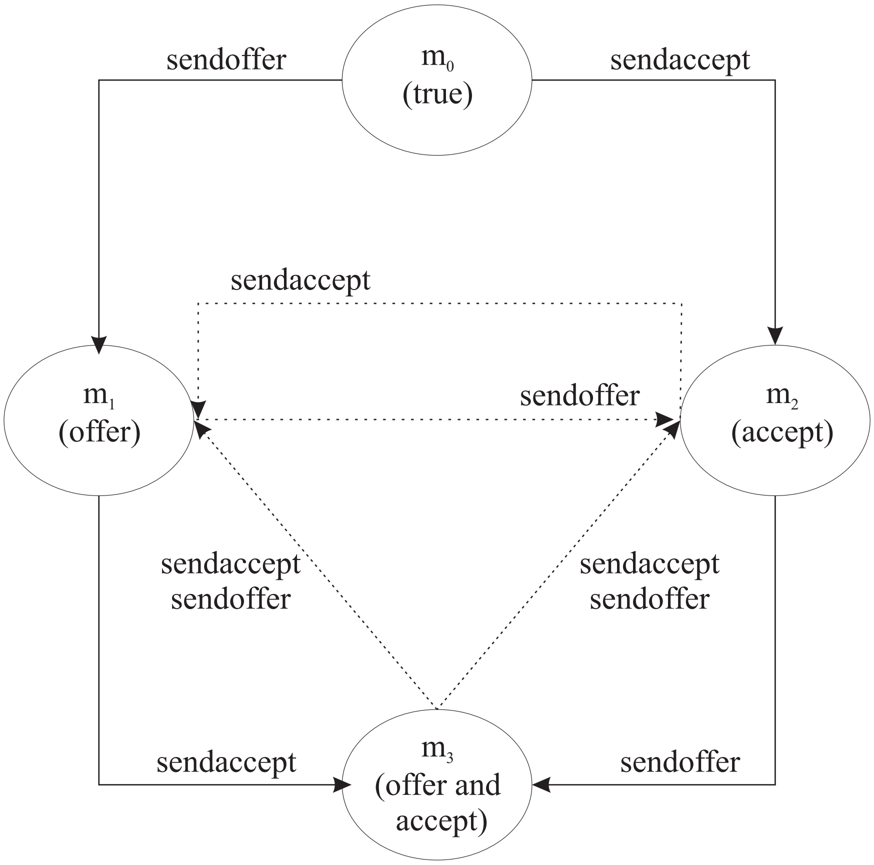

Figure 7: Sample transitions in CMs

- $\langle m_3$, *sendaccept*, $m_2\rangle$: (*offer* $\wedge$ *accept*) $\models_{\langle sendaccept:accept\rangle}$ *accept*
  $\langle m_3$, *sendaccept*, $m_3\rangle$: (*offer* $\wedge$ *accept*) $\models_{\langle sendaccept:accept\rangle}$ (*offer* $\wedge$ *accept)*
  $\langle m_3$, *sendaccept*, $m_1\rangle$: (*offer* $\wedge$ *accept*) $\models_{\langle sendaccept:accept\rangle}$ *offer*
  $\langle m_3$, *sendoffer*, $m_2\rangle$: (*offer* $\wedge$ *accept*) $\models_{\langle sendoffer:offer\rangle}$ *accept*
  $\langle m_3$, *sendoffer*, $m_3\rangle$: (*offer* $\wedge$ *accept*) $\models_{\langle sendoffer:offer\rangle}$ (*offer* $\wedge$ *accept)*
  $\langle m_3$, *sendoffer*, $m_1\rangle$: (*offer* $\wedge$ *accept*) $\models_{\langle sendoffer:offer\rangle}$ *offer*
  It is easy to see that no transitions can start from m_3, since m_3 already entails all the meanings associated with the other states.

In Figure 7, the solid lines show the allowed transitions and the dashed lines show the other possible entailments that are not allowed.

5.1 Compilation Formalized

We now show how a CM can be formally compiled into an FSM. We establish soundness and completeness results regarding our compilation procedure. The compilation procedure can be realized in an automatic tool. Recall that a CM allows multiple starting states, whereas an FSM allows only one. In the compilation below, we show how an FSM can be constructed after choosing a start state for the CM, namely, true.

Procedure 1 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a CM. Construct an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ as follows.

- **S = M**
- $\Sigma = \{a : \langle a : e \rangle \in \Delta\}$
- s_0 = true
- **Q = F**
- $\delta = \{\langle m_i, a, m_j \rangle : m_i, m_j \in \mathbf{M}, \langle a : e \rangle \in \Delta$ and $(m_i \models_{\langle a:e \rangle} m_j, m_i \not\vdash m_j$ and $(\forall\, m_k \in \mathbf{M}$: $m_i \models_{\langle a:e \rangle} m_k \Rightarrow m_j \vdash m_k))\}$

Notice that δ is defined so as to satisfy Restrictions 1 and 2. ▮

When an FSM Y is produced from a CM X by Procedure 1, we say that X is *compiled* into Y. A compiled FSM can be directly executed by an agent with a single thread, using only constant space. Theorem 1 establishes this result.

Theorem 1 An FSM produced by compiling a CM according to Procedure 1 is deterministic.
Proof. Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a CM compiled into $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$. Let $\langle m_i, a, m_j \rangle \in \delta$ and $\langle m_i, a, m_k \rangle \in \delta$ be two transitions of Y. From Restriction 2, we know $m_j \vdash m_k$ and $m_k \vdash m_j$, that is, $m_j \equiv m_k$. Then, by Definition 8, $m_j = m_k$. Thus, by Definition 4, Y is deterministic. ■

The correctness of a compilation procedure must be based on the computations that can result from it. Therefore, we formalize the notion of a computation, how a computation may be generated by a CM, and how a computation may be realized by an FSM.

For the following discussion, let $f \in \mathcal{N}$ be a finite ordinal. Let **I** be the set of indices $\{0, 1, \ldots, (f-1)\}$ (**I** is empty when $f = 0$). Let **J** be the set of indices $\{0, 1, \ldots, f\}$.

Definition 13 $\tau = \langle m_0, a_0, m_1, a_1, \ldots, a_{f-1}, m_f \rangle$ is a computation if ($\forall i \in \mathbf{I}, j \in \mathbf{J} : a_i \in \Sigma$ and $m_i \in \mathbf{M}$). Intuitively, the action labels that occur in τ, $\langle a_0, a_1, \ldots, a_{f-1} \rangle$, describe the externally visible part of the computation because these are the actions seen by other agents. ▮

Definition 14 $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ is *generated* by a CM $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ if and only if $m_f \in \mathbf{F}$, and ($\forall i \in \mathbf{I}$: $m_i \in \mathbf{M}$ and ($\exists e_i, \langle a_i : e_i \rangle \in \Delta$: $m_i \models_{\langle a_i:e_i \rangle} m_{i+1}$)). ▮

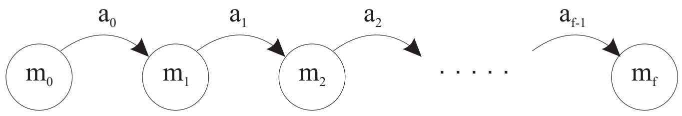

Figure 8: A sample computation

Definition 15 $\tau = \langle m_0, a_0, m_1, \ldots, m_f\rangle$ is *realized* by an FSM $Y = \langle$**S**, Σ, s_0, **Q**, $\delta\rangle$ if and only if $m_0 = s_0$, $m_f \in$ **Q**, and ($\forall i \in$ **I**, $\langle m_i, a_i, m_{i+1}\rangle \in \delta$). ▮

Notice that our definition of a computation includes the meanings associated with the states along with an action sequence. Therefore, a computation can only be generated by a machine that can manipulate these meanings, that is, a CM. By constrast, an FSM can only realize this computation by following the action sequence. With this distinction in mind, we define two main aspects of correctness. *Soundness* means that only allowed computations are realized. *Completeness* means that all allowed computations can be realized. A compilation procedure that produced an FSM $\langle$**S**, Σ, s_0, **Q**, $\delta\rangle$ with **S**=$\{\}$ could be sound, whereas one that produced an FSM with **S**=**M**$\times\Delta\times$**M** would be complete. That is, it is trivial to ensure one of soundness or completeness, but it is not helpful to ensure both. [helpful] Theorem 2 establishes the soundness of our compilation method. $\Leftarrow$
Effectively, it states that the compiled FSM won't produce a computation that was not allowed by the original CM.

Theorem 2 Let $X = \langle$**M**, Δ, **F**$\rangle$ be compiled into $Y = \langle$**S**, Σ, s_0, **Q**, $\delta\rangle$. Then any computation realized by Y is generated by X.
Proof. Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f\rangle$ be a computation generated by Y. By Procedure 1 **S** = **M**, ($\forall i \in$ **I**, $m_i \in$ **M**), and $m_f \in$ **F**. Consider the ith transition in τ, $\langle m_i, a_i, m_{i+1}\rangle \in \delta$. This implies, ($\exists e_i : \langle a_i : e_i\rangle \in \Delta$ and $m_i \models_{\langle a_i:e_i\rangle} m_{i+1}$). By Definition 14, X generates τ. ▮

5.2 Completeness

Interestingly, with the general definition of a CM, our compilation is not complete. First, a computation that is generated by a CM may begin from any arbitrary meaning. Second, there may be no transitions in the FSM corresponding to some transition in the CM. Restriction 2 forces the transitions to yield a meaning that carries maximal information among the possible meanings. It might be the case that a transition emanating from a source state entails several meanings, none of which entails the rest. In this case, no meaning has the maximal information, and no corresponding transition is included in δ. In order to ensure that there is always a meaning with the most information, we need to ensure that the meaning set **M** of the CM is closed under antecedence, that is, for any set of meanings there is a meaning that is stronger than each meaning in the set.

Definition 16 A *complete* CM is a CM whose meaning set **M** and final meaning set **F** are closed under antecedence. Formally, ($\forall R \subseteq$ **M** : ($\exists m_k \in$ **M**: ($\forall m_i \in R : m_k \vdash m_i$))) and ($\forall R \subseteq$ **F**: ($\exists m_k \in$ **F**: ($\forall m_i \in R : m_k \vdash m_i$))). ▮

A complete CM guarantees that in any subset of both the meaning set **M** and the final meaning set **F**, there exists a meaning that entails all the meanings in the subset. A more restrictive requirement is that the conjunction of any set of meanings is also a meaning.

Definition 17 A conjunctive CM is a CM whose meaning set **M** and final meaning set **F** are closed under conjunction. Formally, ($\forall R \subseteq$ **M**: ($\exists m_k \in$ **M**: $m_k \equiv \bigwedge_{m_i \in R} m_i$)) and ($\forall R \subseteq$**F**: ($\exists m_k \in$ **F**: $m_k \equiv \bigwedge_{m_i \in R} m_i$)). ▮

Lemma 1 Every conjunctive CM is complete. ▮

The main idea underlying a CM execution is that instead of specifying protocols in terms of legal sequences of actions, a CM specifies them in terms of meanings to reach. Thus, two computations may follow different sequences of actions but still achieve the same meaning. Since the computations are characterized with the achieved meanings, rather than pure sequences of actions, computations generated by a CM can be compared semantically.

Definition 18 A computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ is *semantically superior* to a computation $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ if and only if ($\forall i \in$ **J**: $m'_i \vdash m_i$). This is written as $\tau' \succeq \tau$. ▮

Definition 19 A computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ generated by a CM X is the *semantically strongest* computation if and only if for all computations $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ generated by X (that involve the same action sequence as τ'), τ' is semantically superior to τ. ▮

Definition 20 A CM X' is *semantically superior* to a CM X, written $X' \succeq X$, if and only if ($\forall \tau : \tau$ is generated by $X \Rightarrow$ ($\exists \tau' : \tau'$ is generated by X' and $\tau' \succeq \tau$)). ▮

In the following discussion, we provide two procedures to convert one computation into another. First, Procedure 2 transforms a given computation into the semantically strongest computation based on a particular action sequence.

Procedure 2 Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by a complete CM $X = \langle$**M**, Δ, **F**$\rangle$. We construct the computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ in which $m'_0 = m_0$ and m'_{i+1} is the strongest state that follows m'_i and $\langle a_i : e_i \rangle$. By the definition of a complete CM (Definition 16), we know that such an m'_{i+1} exists. ▮

Lemma 2 Let $X = \langle$**M**, Δ, **F**$\rangle$ be a complete CM and let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by X. Then Procedure 2 on τ yields a computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ which is the semantically strongest computation (Definition 19) on $\langle a_0, a_1, \ldots, a_{f-1} \rangle$.

Proof. In Procedure 2, after each action a_i at state m'_i, τ' will move to a new state m'_{i+1}, such that m'_{i+1} is the strongest state that can result from doing action a_i in m'_i. Since each m'_i is the strongest state (by the inductive hypothesis), τ'_i is the strongest computation for the given sequence of actions. ▮

Recall that in compiling a CM into an FSM, we have not allowed computations to transition from a meaning to itself. Although a compiled FSM does not allow such a transition, a CM does allow it. In other words, a CM may possibly contain redundant transitions. To classify computations that do not have redundant transitions, we introduce the concept of *efficient* computations. Procedure 3 transforms a given computation into an efficient computation.

Definition 21 A computation is *efficient* if and only if it contains no consecutively repeated states. ▮

Procedure 3 Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by CM $X = \langle$**M**, Δ, **F**$\rangle$. We produce the computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ that does not have any equivalent consecutive states. That is, we start by copying m_0 to τ'. Iteratively, we check whether applying a_i from state m_i move the computation to a new meaning m_{i+1}. If that is the case, we copy both a_i and m_{i+1} to τ'. Otherwise, we skip a_i and continue the iteration with a_{i+1}. ▮

Lemma 3 Procedure 3 yields an efficient computation.
Proof. Since Procedure 3 removes consecutively repeated states, the resulting computation is efficient. ■

Procedure 3 can transform a computation into one that is shorter. Thus, step-by-step comparisons among computations would not apply. For this reason, we introduce the notion of endpoint equivalance. This notion matches our basic intuition of flexible execution because we only care about where a computation ends, not what intermediate states it went through.

Definition 22 A computation $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ is *endpoint equivalent* to computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_{f'} \rangle$ if and only if $m_0 \equiv m'_0$ and $m_f \equiv m'_{f'}$. Notice that f and f' may be different, i.e., the computations may be of different lengths. ▮

Lemma 4 Procedure 3 preserves endpoint equivalence of computations.
Proof. Procedure 3 produces a computation τ' by removing redundant transitions from a computation τ. Any transition that results in a new meaning is kept. Thus, the last state in τ' equals the last state in τ. Hence, endpoint equivalence is preserved. ■

Importantly, Lemma 4 shows that Procedure 3 preserve the property of a computation being the semantically strongest for its actions. That is, although the resulting computation has fewer actions, it is the strongest for the actions in it if the input computation had that property.

Lemma 5 If the computation τ given as input to Procedure 3 is semantically strongest (Definition 19), the computation τ' produced by Procedure 3 is also semantically strongest. ■

Lemma 6 Let $X = \langle$**M**, Δ, **F**$\rangle$ be a complete CM compiled into an FSM $Y = \langle$**S**$, \Sigma, s_0,$ **Q**, $\delta\rangle$ according to Procedure 1. Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by X, such that τ is efficient (Definition 21), semantically strongest (Definition 19), and begins from true. Then τ can be realized by Y.
Proof. By Procedure 1, s_0 = true. Since τ is efficient, we know there are no consecutively repeated states. Thus no transition will be disallowed by Restriction 1. Also, since τ is semantically strongest, each state in τ will be the strongest possible state. Recall that

Restriction 2 enforces this requirement on transitions of FSMs. Since Restriction 1 is never exercised, and Restriction 2 does not cause deviation from the flow of τ, τ can be realized by Y. ∎

We pointed out above that with the general definition of CMs, our compilation is not complete. However, after restricting the class of CMs to complete CMs, we can achieve the following completeness result.

Theorem 3 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a complete CM compiled into an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$. Then for any computation that is generated by X and begins from true, there exists an efficient (Definition 21), semantically strongest (Definition 19) computation realized by Y.

Proof. Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by X. Let $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ be a computation produced by Procedure 2 when given τ as input. By Lemma 2, we know τ' is semantically strongest. Now, if we give τ' as input to Procedure 3, this yields a computation that is efficient (Lemma 4). Further, the computation is semantically strongest (Lemma 5), and can be realized by Y (Lemma 6). ∎

5.3 Finding Complete CMs

The above development has established the soundness and completeness of the compilation procedure from CMs to FSMs. That is, we have imposed some restrictions on the kinds of CMs we can consider. But what about an arbitrary CM? Given an arbitrary CM X, can we find a conjunctive CM X' such that $X' \succeq X$? And how large would X' have to be to support our completeness result? To answer these questions, consider the following procedure meant to produce conjunctive CMs.

Procedure 4 Starting with a CM $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$, construct $\mathbf{M}'$, Δ', and $\mathbf{F}'$ as follows:

- $\mathbf{M}'$ will contain all the meanings present in $\mathbf{M}$ and all the new meanings that can be generated by applying every action in Δ to every meaning in $\mathbf{M}$.
  $\mathbf{M}'$=$\mathbf{M}$
  for all m $\in$ $\mathbf{M}$ do
      for all a $\in \Delta$
          if ($\nexists m' \in \mathbf{M}'$: $m \models_a m'$)
              then $\mathbf{M}'$ =$\mathbf{M}' \cup \{m'\}$

- Δ'=Δ

- $\mathbf{F}'$ will contain all the meanings in $\mathbf{F}$ and all the new meanings in $\mathbf{M}$ that entail the meanings in $\mathbf{F}$. ∎

Lemma 7 Procedure 4 must terminate.

Proof. Follows directly from the fact that the set of possible meanings is finite because the set of atomic propositions is finite. ∎

A CM is coherent only if the different interpretations it provides for a given action in a given state are all compatible. If a CM is not coherent, the results of the same action may be mutually incompatible. This would have the risk that the participating agents will end up in incompatible states leading to the breakdown of their interaction.

Definition 23 A CM is *coherent* if and only if ($\forall m_i$, a, e: $\{m_j$: $m_i \models_{\langle a:e \rangle} m_j\} \neq \mathsf{false}$) ▮

Lemma 8 If X is coherent, then Procedure 4 on X yields a CM that is conjunctive.
Proof. Follows from the definition of conjunctive CMs (Definition 17). ∎

Lemma 9 ensures that the CM created by the above procedure is meaningfully related to the original CM.

Lemma 9 Let Procedure 4 map CM X to CM X'. Then every computation generated by X is generated by X' and for every computation τ' generated by X', there is a computation τ generated by X such that $\tau' \succeq \tau$.
Proof. ∎

Definition 24 The *size* of a CM $X = \langle$**M**, Δ, **F**$\rangle$ is given by $(|M| + |\Delta|)$. ▮

Theorem 4 Given an input CM $X = \langle$**M**, Δ, **F**$\rangle$, Procedure 4 yields a CM that is conjunctive and whose size is bounded from above by $(|M| * |\Delta|)$.
Proof. The worst case happens when we must create a new state in X' for each state and out edge in X. This increases the cardinality of the meaning set and doesn't affect the cardinality of the transition relation. ∎

Now we can answer the above question in the affirmative. For any CM, a corresponding conjunctive CM can always be found. Moreover, we suffer at most a quadratic blowup in creating the conjunctive CM. As a result, the restrictions on CMs applied by Theorem 3 are not a practical problem.

6 Conclusions

Agents can provide the autonomy and heterogeneity required from participants in open, dynamic applications. However, in order to fully exploit the agents' capabilities, the underlying communication protocols must be flexible yet easy to execute, so as to accommodate the varying interactions that can take place among agents. Traditionally, communication protocols are specified by defining only the allowed orders in which communicative acts may take place. This holds for protocol formalisms such as FSMs, formal grammars, and Petri Nets. Some of these formalisms are quite powerful, but they are used only to specify allowed sequences of actions. The actions are just labels, and the states, if explicit, do not capture the conceptual meaning of a protocol.

We proposed a new formalism, the commitment machine, to specify and execute protocols in multiagent systems. CMs provide flexibility by capturing the semantic content of the actions in a protocol. By specifying communication protocols using commitments, we can analyze the interactions among participants through the intrinsic meaning of those interactions. We showed how CMs can be applied at run time or compile time, depending on the computational restrictions and agent architectures.

6.1 Literature

There is a substantial body of literature on agent communication languages (ACLs) and their semantics. The Foundation for Intelligent Physical Agents (FIPA) has been standardizing an ACL along with a formal semantics based on a logic of beliefs and intentions. FIPA also includes interaction protocols, which are characterized purely operationally. Labrou and Finin (1998) describe a grammar for constructing conversations or protocols. This is a formal grammar but it only describes the sequencing of tokens. Labrou and Finin also give a belief and intention based semantics for Knowledge Query Manipulation Language (KQML). There is thus a major disconnect between the FIPA ACL semantics and protocol operationalization, and likewise between Labrou and Finin's ACL semantics and protocol execution. By contrast, in our approach the semantics given to the messages is directly operationalized. In our approach, the tokens are chosen for each protocol. We don't attempt to give a semantics for tokens like *inform* that can apply in all possible protocols. Recently, Moore (2000) has developed an ACL called Formal Language for Business Communication (FLBC) that offers a first-order syntax and specialized terms for describing business contracts. FLBC has not been applied as widely as KQML. However, FLBC can be a useful content language for business communication. It complements our approach, which by contrast, emphasizes protocols.

Commitments have been studied before (Castelfranchi, 1995; Gasser, 1998), but were not used for protocol specification as we have done here. Tambe (1997) apply commitments as part of a general architecture for teamwork. In order to achieve flexible teamwork, he develops an agent architecture, STEAM which is based on joint-intentions theory. STEAM utilizes team and communication operators to represent the team goals, and joint commitments explicitly. This explicit representation brings along flexibility by enabling modification of goals and commitments at run time.

Verharen (1997) develops a contract specification language, CoLa, to specify transactions and contracts. Verharen's approach benefits from commitments in expressing actions, but it treats commitments as obligations, and does not allow manipulation of commitments as in our approach. Further, Verharen only considers base-level commitments, without capturing conditional commitments as we have done through metacommitments.

Barbuceanu and Fox (1995) develop a language, COOL, for describing coordination among agents. Their approach is based on modeling conversations through FSMs, where the states denote the possible states a conversation can be in, and the transitions represent the flow of the conversation through message exchange. Barbuceanu and Fox handle exceptions through error recovery rules. They give content to messages based on speech acts, but they allow only predefined transitions, disallowing dynamic transitions based on the meaning of messages.

Fisher and Wooldridge develop a programming language, Concurrent MetateM, to enable the development of flexible protocols (1997). Concurrent MetateM includes a richer structure for time than we have studied. However, it lacks the richness of commitments that we have discussed here.

Dignum and van Linder (1997) propose a framework for social agents based on dynamic logic, in which they distinguish messages based on speech acts. In addition to employing commitments, they use *directions* and *declarations* to denote the semantic content of messages. Further, they employ an authority relation between agents to decide on the success of directions and declarations. In our work, we assumed peer-to-peer interactions, in that we do not consider the interactions based on different authority among agents. But it would be a simple matter to handle various authority relations, e.g., to require that requests from one agent are always honored by the other.

Haddadi (1998) develops a formal semantics based on beliefs, desires and intentions of agents. She describes the means-ends reasoning process of agents, through the concepts of chance, choice, and commitment. Haddadi's description of chance is similar to our description of opportunity, but her treatment of commitments varies from our treatment, in that she focuses on formation of commitments whereas we study how commitments can be applied. Haddadi's emphasis on architectures for agents who can interact flexibly is crucial, and complements our emphasis on the interactions themselves.

6.2 Future Directions

In order to present commitment machines in the simplest form, this research has simplified the concept of meanings and transitions. First, the meanings as presented here do not capture temporal relations, and secondly meanings transition monotonically. Investigating extensions to enabling non-monotonic transitions and temporal relations among meanings are two directions of research.

The reasoning rules that are supplied here benefit only from two operations on commitments (create and discharge). It would be very useful to develop reasoning rules to manipulate the remaining operations on commitments. Similarly, enhancing the reasoning rules to handle the transformation of commitments as the context group changes is another direction of research.

Although we believe in the importance of context group in commitments, this study has not investigated how commitments would be modified based on varying context group. Yet another direction would be extending the reasoning rules we have developed here, to handle the transformation of commitments as the context group changes. These are all interesting topics, which we defer to future research.

Acknowledgments

We would like to thank James Lester and Peter Wurman for helpful comments. This research was supported by the National Science Foundation under grant IIS-9624425 (Career Award). A previous version of this paper appears in the WET ICE 2000 Workshop proceedings.

References

Austin, J. L. (1962). *How to Do Things with Words*. Clarendon Press, Oxford.

Barbuceanu, M., & Fox, M. S. (1995). COOL: A language for describing coordination in multi agent systems. In *Proceedings of the International Conference on Multiagent Systems*, pp. 17–24.

Castelfranchi, C. (1995). Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems*, pp. 41–48.

Dignum, F., & van Linder, B. (1997). Modelling social agents: Communication as action. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pp. 205–218. Springer-Verlag.

Fisher, M., & Wooldridge, M. (1997). On the formal specification and verification of multi-agent systems. *International Journal of Intelligent and Cooperative Information Systems*, *6*(1), 37–65.

Gasser, L. (1998). Social conceptions of knowledge and action: DAI foundations and open systems semantics. In *(Huhns & Singh, 1998)*, pp. 389–404. (Reprinted from *Artificial Intelligence, 1991*).

Haddadi, A. (1998). Towards a pragmatic theory of interactions. In *(Huhns & Singh, 1998)*, pp. 443–449. (Reprinted from *Proceedings of the International Conference on Multiagent Systems, 1995*).

Huhns, M. N., & Singh, M. P. (Eds.). (1998). *Readings in Agents*. Morgan Kaufmann, San Francisco.

Labrou, Y., & Finin, T. (1998). Semantics and conversations for an agent communication language. In *(Huhns & Singh, 1998)*, pp. 235–242. (Reprinted from *Proceedings of the International Joint Conference on Artificial Intelligence, 1997*).

Moore, S. A. (2000). KQML and FLBC: Contrasting agent communication languages.. To appear in *International Journal of Electronic Commerce*.

Singh, M. P. (1999). An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, *7*, 97–113.

Sirbu, M. A. (1998). Credits and debits on the internet. In *(Huhns & Singh, 1998)*, pp. 299–305. (Reprinted from *IEEE Spectrum, 1997*).

Tambe, M. (1997). Agent architectures for flexible, practical teamwork. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 22–28.

Venkatraman, M., & Singh, M. P. (1999). Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems*, *2*(3), 217–236.

Verharen, E. M. (1997). *A Language-Action Perspective on the Design of Cooperative Information Agents*. Catholic University, Tilburg, Holland.

Winograd, T., & Flores, F. (1987). *Understanding Computers and Cognition: A New Foundation for Design*. Addison-Wesley, Reading, MA.