Commitment Machines*

Pınar Yolum Munindar P. Singh

Department of Computer Science,
North Carolina State University
Raleigh, NC 27695-7535, USA
{pyolum, mpsingh}@eos.ncsu.edu

Abstract. We develop an approach in which we model communication protocols via *commitment machines*. Commitment machines supply a content to protocol states and actions in terms of the social commitments of the participants. The content can be reasoned about by the agents thereby enabling flexible execution of the given protocol. 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. We show how a commitment machine can be compiled into a finite state machine for efficient execution, and prove soundness and completeness of our compilation procedure.

1 Introduction

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. Multiagent protocols have traditionally been modeled by formalisms similar to those used to specify protocols in distributed computing and computer networks (e.g., finite state machines [2] or Petri Nets [4]). These formalisms specify protocols merely in terms of legal sequences of actions without regard to the meanings of those actions. Consequently, protocols tend to suffer from unnecessary rigidity in execution [7], resulting in redundant interactions and avoidable failures.

Let us consider some desirable properties of a protocol representation.

- *Autonomy:* Promoting the participants' autonomy is crucial for creating effective systems in open environments. Participants must be constrained in their interactions only to the extent necessary to carry out the given protocol, and no more.
- *Opportunities:* Participants should be able to take advantage of opportunities to improve their choices or to simplify their interactions. Depending on the situation, a participant 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.

* 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 preliminary version of this paper appears in the WET ICE 2000 Proceedings.

– *Exceptions:* Participants must be able to modify their interactions to handle exceptions that result from the unexpected behavior of the participants. For example, a deadline may be renegotiated at a discount. This would obviously involve domain knowledge, but the protocol representation should allow it.

We propose *commitment machines (CMs)* as a formalism to formally specify and execute protocols. A commitment machine attaches specific *declarative* 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 declarative 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 example and necessary background information. Section 3 introduces commitment machines and show how they may be applied. Section 4 shows how commitment machines can be compiled into finite state machines and establishes important results regarding the compilation procedure. Section 5 describes our contributions with respect to the most relevant literature.

2 Technical Framework

Following speech act theory, we view communication as a form of action [1]. The actions here reflect progress in the given protocol and may be captured in terms of modifications to the participants' commitments.

Social commitments are commitments made from one agent to another agent to carry out a certain course of action [3, 11]. A social commitment $\mathsf{C}(x, y, p)$ relates a debtor x, a creditor y, and a condition p. When a social commitment of this form is created, x becomes responsible to y for satisfying p. The condition p may involve relevant predicates and commitments, allowing the commitments to be nested or conditional. The commitments are flexible and can be revoked or modified. Almost always, the revocation or modification is constrained through other commitments.

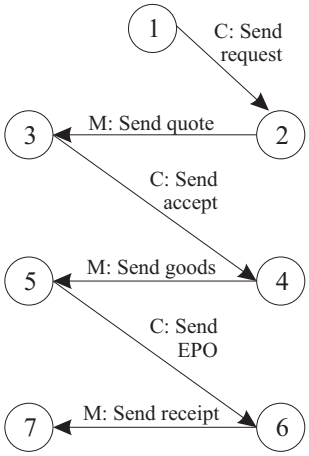
Viewing a commitment as an abstract data type, the creation and the manipulation of the commitments can be described using the following operations [11, 14]. Here, x, y, z denote agents, and c and c' denote commitments of the form $\mathsf{C}(x, y, p)$.

1. *Create(x, c)* establishes the commitment c.
2. *Discharge(x, c)* resolves the commitment c, i.e., the condition p starts to hold.
3. *Cancel(x, c)* cancels the commitment c. Usually, the cancellation of a commitment is accompanied by the creation of another compensating commitment.

4. *Release(y, c)* releases the debtor from the commitment c. It can be performed by the creditor, 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.

As a running example, we consider a simplified version of the NetBill protocol which was developed to handle the buying and selling over the Internet of electronic goods, such as software and electronic documents [12].

Example 1 As shown in the figure below, 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 sends the receipt to the customer, who can then successfully decrypt and use the goods. Some scenarios that will not be handled by this protocol specification are shown on the right side of the figure.



(i)Instead of waiting for a customer to request a quote, a merchant may proactively send a quote, mimicking the idea of advertising.
(ii) 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.
(iii) 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.

Example 2 We define the semantic content of each state in the figure above 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 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.

3 Commitment Machines

We define a commitment machine (CM) in terms of sets of states and actions that are given a declarative 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.

Importantly, unlike in an FSM, the transitions between the states are not explicitly specified. 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 a protocol emphasizes that the aim of executing the protocol is not merely to perform certain sequences of actions, but to reach a desirable state. With this in mind, we can come up with different action sequences or paths that accomplish the same goal as the original path.

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 start symbol *Protocol* gives the syntax of $\mathcal{P}$. In this grammar, *slant* typeface indicates nonterminals; $\longrightarrow$ is a metasymbol of BNF; $\ll$ and $\gg$ delimit comments; { and } indicate that the enclosed item is repeated 0 or more times; the remaining symbols are terminals. For expository ease, we use a simplified notation for commitments. 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. Further, we restrict the nesting of commitments to one level.

- *Protocol* $\longrightarrow$ {*Action*} $\ll$set of actions$\gg$
- *Action* $\longrightarrow$ *Token:* L $\ll$token is a label; L is the associated meaning$\gg$
- *Commitment* $\longrightarrow$ C_x (L) | C_x (M) $\ll$simplified as explained below$\gg$
- $L \longrightarrow$ *Commitment*
- $M \longrightarrow L \rightsquigarrow L$ $\ll$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* ≪atomic propositions≫

The boolean 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 only in commitments. Our formal semantics is given in Appendix A. For commitments where x and y are the same ($\mathsf{C}_x(p \rightsquigarrow \mathsf{C}_x r)$), the simpler form $\mathsf{C}_x(p \rightsquigarrow r)$ suffices.

Example 3 Following Figure 1, we define the following atomic propositions and commitments that will be used to specify the meanings.

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

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 equivalence. $p \vdash q$ means that q can be logically derived from p. $p \equiv q$ means that p and q are logically equivalent, that is, $p \vdash q$ and $q \vdash p$. These two relations are used to compare the execution states of a protocol semantically.

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 1 A set **M** of meanings is *minimal* if and only if the following hold:

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

Definition 2 A set of final meanings **F** is *consistent* with respect to a set of meanings **M** if and only if any meaning that is stronger than a final meaning is also final. That is, $(\forall m_i \in \mathbf{F}, m_j \in \mathbf{M}\text{: } (m_j \vdash m_i) \Rightarrow (m_j \in \mathbf{F}))$. ▌

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

Definition 3 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. ▌

Next we specify the legal meanings the NetBill protocol execution can be in, the actions the agents can perform and the final meanings that the protocol can end. Since each action can be performed by only one party, we do not specify the performers explicitly. Table 1 gives the CM specification.

	Meanings (M)	Actions (Δ)	Final Meanings (F)
1	true	⟨sendRequest: request⟩	request
2	*request*	⟨sendQuote: *offer*⟩	offer
3	*offer*	⟨sendAccept: *accept*⟩	goods $\wedge$ *pay* $\wedge$ *receipt*
4	C_m*goods* $\wedge$ *accept* $\wedge$ *promiseReceipt*	⟨sendGoods: *goods* $\wedge$ *promiseReceipt*⟩	
5	*goods* $\wedge$ C_c*pay* $\wedge$ *promiseReceipt*	⟨sendEpo: *pay*⟩	
6	*goods* $\wedge$ *pay* $\wedge$ C_m*receipt*	⟨sendReceipt: *receipt*⟩	
7	*goods* $\wedge$ C_c*pay* $\wedge$ *receipt*		
8	*goods* $\wedge$ *pay* $\wedge$ *receipt*		

Table 1. The CM representation of the NetBill protocol

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: $q \models_{\langle a:e \rangle} r \triangleq (q \wedge e) \vdash r$. 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, we now present some important rules for reasoning about their consequences. These reasoning rules capture the operational semantics of our approach.

1. A commitment $\mathsf{C}_x p$ ceases to exist when the proposition p becomes true.
2. A commitment $\mathsf{C}_x(p \rightsquigarrow r)$ ceases to exist when the proposition p becomes true, but a new base-level commitment $\mathsf{C}_x r$ is created to capture that x has to satisfy the original commitment by bringing about the proposition r.
3. A commitment $\mathsf{C}_x(p \rightsquigarrow r)$ ceases to exist when the proposition r holds (before p is initiated), and no additional commitments are created.

Example 4 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 CM specification of a protocol can be applied both at run time and compile 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 directly execute the CM. 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. Alternatively, a CM can be compiled into an FSM that abstracts out the meanings of the states and actions. In the next section, we describe this compilation process in detail.

4 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 declarative 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 declarative meanings at run time.

Before considering the requirements of compiling a CM into an FSM, let us give a formal definition of an FSM and a DFSM.

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

For an agent to be able to directly execute the FSM that results from compiling a CM, 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.

4.1 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 ($m_i \models_{\langle a:e \rangle} m_j$, $m_i \not\models m_j$ and ($\forall$ $m_k \in \mathbf{M}$: $m_i \models_{\langle a:e \rangle} m_k \Rightarrow m_j \vdash m_k$))$\}$ ▮

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 $m_i \not\models m_j$.

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 ($\forall$ $m_k \in \mathbf{M}$: $m_i \models_{\langle a:e \rangle} m_k \Rightarrow m_j \vdash m_k$).

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. ▮

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 1, $m_j = m_k$. 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. 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 5 $\tau = \langle m_0, a_0, m_1, a_1, \ldots, a_{f-1}, m_f \rangle$ is a computation if ($\forall i \in \mathbf{I}, j \in \mathbf{J}$: $a_i \in \Sigma$ and $m_j \in \mathbf{M}$). 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 6 $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ 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 ($\exists e_i, \langle a_i : e_i \rangle \in \Delta$: $m_i \models_{\langle a_i : e_i \rangle} m_{i+1}$)). ∎

Definition 7 $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ is *realized* by an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ if and only if $m_0 = s_0$, $m_f \in \mathbf{Q}$, and ($\forall i \in \mathbf{I}$, $\langle m_i, a_i, m_{i+1} \rangle \in \delta$). ∎

Notice that a computation can only be generated by a machine that can manipulate these meanings, that is, a CM. By contrast, an FSM can 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 produces an FSM $\langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ with $\mathbf{S}=\{\}$ would be sound, whereas one that produces an FSM with $\Sigma=\mathbf{M}\times\Delta\times\mathbf{M}$ would be complete. That is, it is trivial to ensure either soundness or completeness. However, it is crucial to ensure both properties. Theorem 2 establishes the soundness of our compilation method: it states that the compiled FSM won't produce a computation that was not allowed by the original CM.

Theorem 2 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be compiled into $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$. 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 realized by Y. By Procedure 1 $\mathbf{S} = \mathbf{M}$, ($\forall i \in \mathbf{I}, m_i \in \mathbf{M}$), and $m_f \in \mathbf{F}$. 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 6, X generates τ. ∎

4.2 Completeness

Interestingly, we must refine the definition of a CM to ensure completeness. 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 8 A *complete* CM is a CM whose meaning set **M** and final meaning set **F** are closed under antecedence. Formally, ($\forall R \subseteq \mathbf{M} : (\exists m_k \in \mathbf{M}$: $(\forall m_i \in R : m_k \vdash m_i)$)) and ($\forall R \subseteq \mathbf{F}$: ($\exists m_k \in \mathbf{F}$: ($\forall m_i \in R : m_k \vdash m_i$))). ∎

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.

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 9 A computation $\tau' = \langle m'_0, a_0, m'_1, \ldots, m'_f \rangle$ is *semantically superior* to a computation $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ (with the same action sequence as τ' if and only if ($\forall i \in \mathbf{J}$: $m'_i \vdash m_i$). This is written as $\tau' \succeq \tau$. ▮

Definition 10 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 11 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 $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$. 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 8), we know that such an m'_{i+1} exists. ▮

Lemma 1 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ 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 10) 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. A computation is *efficient* if and only if it contains no consecutively repeated states. Procedure 3 transforms a given computation into an efficient computation.

Procedure 3 Let $\tau = \langle m_0, a_0, m_1, \ldots, m_f \rangle$ be a computation generated by CM $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$. 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 2 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 equivalence. 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 12 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 the computations may be of different lengths. ▮

Lemma 3 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 3 shows that Procedure 3 preserves 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 in the first place.

Lemma 4 If the computation τ given as input to Procedure 3 is semantically strongest, the computation τ' produced by Procedure 3 is also semantically strongest. ∎

Lemma 5 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a complete CM compiled into an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ 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 semantically strongest, 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, complete CMs can easily be found by making the meaning set closed under conjunction. 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, semantically strongest 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 1, we know τ' is semantically strongest. If we give τ' as input to Procedure 3, this yields a computation that is efficient (Lemma 3), and semantically strongest (Lemma 4). Further, this computation can be realized by Y (Lemma 5). ∎

5 Discussion

Commitments have been studied before [3, 8], but were not used for protocol specification as we have done here. Commitment machines 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.

Verharen [15] 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 simple, undirected 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.

Dignum and van Linder [5] 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.

d'Inverno *et al.* [6] develop interaction protocols for the multiagent framework, Agentis. They model protocols as a composition of various services and tasks requested and offered among agents. d'Inverno *et al.*'s protocol model consist of four levels: registration, service, task and notification. In all levels of Agentis, the protocols are specified with FSMs, which are formal and simple, but low level.

Smith *et al.* [13] develop protocols in which actions are given a content based on joint intentions. We agree with them on the necessity of declarative content. They model the content of actions with mental attributes whereas we use social constructs. They seem to informally map the joint intentions of the agents to particular states of a finite state machine, but their compilation procedure is not clear. Conversely, our compilation procedure is precise to the level of states, and its soundness and completeness has been established.

Pitt and Mamdani [10] develop an agent communication language (ACL) framework in terms of protocols, and show how an agent replies to a communication by choosing one of the communications allowed by the given communication. They give content to messages based on social constructs, similar to the present approach.

References

1. J. L. Austin. *How to Do Things with Words*. Clarendon Press, Oxford, 1962.
2. M. Barbuceanu and M. S. Fox. COOL: A language for describing coordination in multi agent systems. In *Proc. of the International Conf. on Multiagent Systems*, pages 17–24, 1995.
3. C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proc. of the International Conf. on Multiagent Systems*, pages 41–48, 1995.
4. R. S. Cost *et al.* Using colored Petri nets for conversation modeling. In Frank Dignum and Mark Greaves, ed., *Issues in Agent Communication*, LNAI 1916, pages 178–192, 2000.

5. F. Dignum and B. van Linder. Modeling social agents: Communication as action. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pages 205–218, 1997.
6. M. d'Inverno *et al.* Interaction protocols in Agentis. In *Proc. of the 3rd Int. Conf. on Multi-agent Systems*, pages 112–119, 1998.
7. M. Fisher and M. Wooldridge. On the formal specification and verification of multi-agent systems. *Int. Journal of Intelligent and Cooperative Information Systems*, 6(1):37–65, 1997.
8. L. Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. In *[9]*, pages 389–404. 1998. (Reprinted from *Artificial Intelligence, 1991*).
9. M. N. Huhns and M. P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, 1998.
10. J. Pitt and A. Mamdani. A protocol-based semantics for an agent communication language. In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, pages 486–491, 1999.
11. M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.
12. M. A. Sirbu. Credits and debits on the Internet. In *[9]*, pages 299–305. 1998. (Reprinted from *IEEE Spectrum, 1997*).
13. I. A. Smith *et al.* Designing conversation policies using joint intention theory. In *Proc. of the 3rd International Conf. on Multiagent Systems*, pages 269–276, 1998.
14. M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. *Autonomous Agents and Multi-Agent Sys.*, 2(3):217–236, 1999.
15. E. M. Verharen. *A Language-Action Perspective on the Design of Cooperative Information Agents*. Catholic University, Tilburg, Holland, 1997.

A Semantics

The meanings of formulas generated from *L* in our BNF grammar are given relative to a model and a state in the model. The meanings of formulas generated from *Protocol* are given relative to a path and a state on the path. The boolean operators are standard. Useful abbreviations include $\mathsf{false} \equiv (p \wedge \neg p)$, for any $p \in \Phi$, $\mathsf{true} \equiv \neg\mathsf{false}$, $p \vee q \equiv \neg(\neg p \wedge \neg q)$ and $p \rightarrow q \equiv \neg p \vee q$.

$M = \langle \mathbf{S}, <, \approx, \mathbf{N}, \mathbf{A}, \mathbf{C} \rangle$ is a formal model for $\mathcal{P}$. $\mathbf{S}$ is a set of states; $<\subseteq S \times S$ is a partial order indicating branching time, $\approx\subseteq S \times S$ relates states to similar states, and $\mathbf{N} : \mathbf{S} \mapsto 2^{\Phi}$ is an interpretation, which tells us which atomic propositions are true in a given state. $\mathbf{P}$ is the set of paths derived from $<$. $\mathbf{PP}$ gives the powerset of $\mathbf{P}$. For $t \in \mathbf{S}$, $\mathbf{P}_t$ is the set of paths emanating from t. $\mathbf{A}$ is a set of agents. $\mathbf{C} : \mathbf{S} \times \mathbf{A} \times \mathbf{A} \mapsto \mathbf{PP}$ give the modal accessibility relations for commitments, respectively.

For p derived from *Protocol*, $M \models_t p$ expresses "M satisfies p at t" and for p derived from *P*, $M \models_{P,t} p$ expresses "M satisfies p at t along path P."

M1. $M \models_t \psi$ iff $\psi \in \mathbf{N}(t)$, where $\psi \in \Phi$
M2. $M \models_t p \wedge q$ iff $M \models_t p$ and $M \models_t q$
M3. $M \models_t \neg p$ iff $M \not\models_t p$
M4. $M \models_t p \rightsquigarrow q$ iff $M \models_t p$ and $(\forall t' : M \models_{t'} p \Rightarrow (\forall t'' : t' \approx t'' \Rightarrow M \models_{t''} q))$
M5. $M \models_t \mathsf{C}(x, y, p)$ iff $(\forall P : P \in \mathbf{C}(x, y, t) \Rightarrow M \models_{P,t} p)$