Nonmonotonic Commitment Machines [⋆]

Amit Chopra and Munindar Singh

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

Abstract. Protocols for multiagent interaction need to be flexible because of the open and dynamic nature of multiagent systems. Such protocols cannot be modeled adequately via finite state machines (FSMs) as FSM representations lead to rigid protocols. We propose a commitment-based formalism called Nonmonotonic Commitment Machines (NCMs) for representing multiagent interaction protocols. In this approach, we give semantics to states and actions in a protocol in terms of commitments. Protocols represented as NCMs afford the agent flexibility in interactions with other agents. In particular, situations in protocols when nonmonotonic reasoning is required can be efficiently represented in NCMs.

1 Introduction

A protocol is a means of achieving meaningful interaction. Agents that constitute a multiagent system use protocols to guide their interactions with each other. Protocols have traditionally been specified as FSMs that specify sequences of states. The protocol designers have certain scenarios in mind that they directly incorporate in an FSM. As a result, agents using a protocol specified as FSMs are limited to behaving in a rigid manner. Such agents cannot handle exceptions or take advantage of opportunities that might arise during interactions with other agents. In this paper, we present an alternative way of specifying protocols that is based on commitments which we formalize below. Our approach is based on the general notion that an agent does not violate a given protocol as long the agent does not violate the commitments prescribed by the protocol. Using commitments makes the protocol flexible and enables the agent to handle exceptions and opportunities without violating the given protocol.

Protocols for interaction in multiagent systems often resemble protocols routinely used by humans in their social interactions. The Contract Net [1] and NetBill [2] are examples of such protocols. Such protocols have traditionally been represented by FSMs that represent sequences of states and transitions. Since FSMs are a low level representation, it becomes cumbersome to capture multiple scenarios in an FSM. Thus FSMs designed by hand tend to be rigid and do not allow scenarios other than the specified "normal" ones. A protocol transitions from state to state as a result of the actions of

[⋆] We are grateful to the anonymous referees for their useful comments. We would like to thank Ashok Mallya and Pinar Yolum for the helpful discussions. This work is supported by the National Science Foundation under grant DST-0139037.

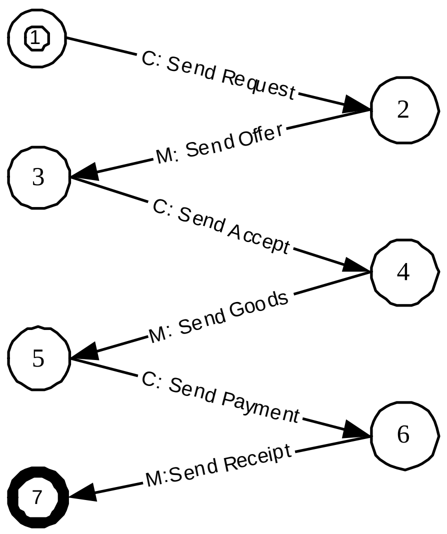

Fig. 1. FSM representation of the simplified NetBill protocol

the interacting agents. A transition is usually labeled with the actions that cause it. In an FSM, the states and the actions in the protocol are meaningless tokens. The agent is limited to executing one of the sequences of actions hard-coded by its designer. These sequences represent the only legal behaviors of the agent. Anything the agent does outside of this protocol is considered a violation. This makes the protocol inflexible and, therefore, undesirable in open multiagent systems where agents are autonomous and heterogeneous, and opportunities and exceptions need to be handled appropriately.

Acting flexibly presupposes reasoning about the protocols. Reasoning formally presupposes that the protocols have a formal semantics. We base our semantics on the notion of commitments. Protocols can naturally be seen as an exchange and manipulation of commitments. A commitment is a directed obligation from one agent to another for achieving or maintaining a state of affairs. A commitment is social because it involves two parties and is publicly observable by all the agents in the agent society. Since a commitment is public, it is also possible to verify whether an agent has fulfilled its commitment, thereby making it possible to check an agent's compliance with a protocol.

This paper uses the NetBill e-commerce protocol [2] as a running example throughout the paper. Figure 1 shows an FSM representation of a NetBill simplified to focus on the core part. The customer, represented by *c*, sends a request for offers to the merchant, represented by *m*. The merchant sends an offer in response. If the customer accepts the offer, the merchant sends the goods. The customer then sends the payment for the goods in return for which the merchant sends a receipt. The only execution scenario possible in the protocol starts with the customer sending a request and ends with the merchant sending a receipt. The FSM in Figure 1 does not accommodate scenarios that would arise naturally in open and dynamic multiagent systems and is, therefore, unnecessarily rigid. Protocols for multiagent interaction should be flexible in the following ways:

– *Autonomy:* A protocol specification should not impinge on the autonomy of an agent beyond the essential nature of the interaction it describes. Consider a scenario where a customer wants to buy goods from a merchant. A desirable specification should not limit the autonomy of an merchant by preventing him from advertising his wares by sending an *offer* message prior to receiving a request for offers.
– *Opportunities:* A protocol should enable an agent to take advantage of opportunities that may arise. For example, if a merchant advertises an attractive deal to a customer, the customer should be able to entertain this offer.
– *Exceptions:* A protocol should enable an agent to deal with exceptions instead of aborting the interaction altogether. For example, a customer who doesn't have enough money might delegate a commitment to pay the merchant to some other agent. This is not allowed in the NetBill protocol as specified above.

The above forms of flexibility can be achieved only if we are able to reason about the content of the states and actions in a protocol. Often, the essential element of content in many protocols is the commitments of the different parties in a protocol. Specifically, we claim that if the protocol representation uses commitments and the criterion for protocol compliance is the satisfaction of commitments, then the above scenarios would be valid behaviors in the protocol.

We propose a formalism for specifying protocols called Nonmonotonic Commitment Machines (NCMs) that uses commitments for representing states and actions. The meaning of a state is given by the commitments that hold in that state; a state is a description of the world. The meaning of an action is given by how it manipulates commitments. An NCM does not directly specify sequences of states and transitions. Instead, it specifies rules in Nonmonotonic Causal Logic (NCL) [3]. These rules model the changes in the state of a protocol as a result of execution of actions. The inference mechanism of NCL computes new states at runtime. Yolum and Singh [4] first studied commitment machines. They did not consider situations during the execution of a protocol when agents must act with incomplete information. In such situations, the agent would need nonmonotonic or defeasible reasoning. Since NCL supports nonmonotonic reasoning, NCMs can express defaults in a protocol in a natural manner. Protocols represented as NCMs are more elaboration tolerant [5] than those represented using classical logic or FSMs. We develop a causal theory of commitments and represent the NetBill protocol as an NCM using that theory.

The rest of the paper is organized as follows. Section 2 motivates the need for nonmonotonic logic for protocol representation and describes the NCL that we employ for this purpose. Section 3 provides a description of commitments. Section 4 formalizes commitments and NetBill in this logic. Section 5 discusses the relevant literature and section 6 discusses future directions.

2 Nonmonotonic Causal Logic

In logic, the consequence relation $\vdash$ is a relation between sets of propositions and individual propositions. $A \vdash x$, where A is a set of propositions and x is a proposition, means that *x* is a logical consequence of *A*. Classical logic is monotonic meaning that if $A \subseteq B$, where $A \vdash x$ and *B* is a set of propositions, then $B \vdash x$. Informally, monotonicity means that the addition of new information does not invalidate old information. Therefore, making rules defeasible in the face of change poses difficulties. Consider Example 1.

Example 1. A customer may not return goods received and should pay for them. However, if the goods are damaged, then the customer may return the goods and then cancel his commitment to pay.

Example 1 involves defeasible reasoning. The *default* rule is that the commitment to pay cannot be canceled. (A more accurate rule is that the debtor of a commitment cannot cancel his commitment. However, the above simplified version is adequate for this example.) The general rule is defeasible, i.e., if a special condition applies, like when the goods are damaged and they are returned, the commitment can be canceled. The default rule should be applicable when no information about the condition is available. This is the essence of nonmonotonic reasoning. Such defeasible reasoning is beyond the realm of classical logic.

2.1 Introduction

To overcome the aforementioned difficulties in specifying protocols, we use NCL. We choose NCL because it has an intuitive syntax and semantics. The language *C*+ [3], which is based on NCL, has a semantics based on state transition systems which agrees with our intuition about protocols. Further, it has been shown to be elaboration tolerant [6]. NCL is a logic of *universal causation* meaning that every fact that is caused holds and every fact that holds is caused. Universal causation is not so much a philosophical stance as a practical one, as universal causation yields a uniform semantics for causal theories. Taking this stance makes NCL suitable for simulation and planning since everything can be explained. Also, in our domain, a commitment holds or does not hold only because there is a reason for it to hold or not hold. Moreover, as we show below, universal causation can be disabled for selected formulas.

The *signature* of a causal theory is the set α of symbols called *constants*. Each constant *c* is assigned a nonempty finite domain *Dom(c)* of symbols. An *atom* is of the form $c = v$ where $v \in Dom(c)$. An *interpretation* of α is an assignment $c = v$ for each $c \in \alpha$ where $v \in Dom(c)$. Since we consider only boolean atoms, either $c = true$ or $c = false$. A *formula* in NCL is a combination of atoms using the connectives of classical logic. A causal rule is of the form $F \Leftarrow G$, where *F* and *G* are formulas of classical logic and are called the head and the body of the rule, respectively. This means that there is cause for *F* to be true if *G* is true. It does not say that *G* is the cause for *F*. This reflects the intuition that it is sufficient to know the conditions under which a fact is caused. As an example, consider a switch *S* that, when closed, lights two bulbs *A* and *B*. Even though *A* being lit is not the cause for *B* being lit, it is correct to say that there is a cause for *B* to be lit when *A* is lit. A theory in NCL consists of a set of causal rules.

Constants in causal theories are either *fluents* or *actions*. A causal theory describes histories of length *m*+1, ($m \geq 0$), by creating for all *i*, ($i \in \{0, \ldots, m\}$), a copy of every fluent and, for all *i*, ($i \in \{0, \ldots, m-1\}$), a copy of every action. The interpretation of fluents for a particular *i* represents state s_i and the interpretation of actions in state s_i represent the transition to state s_{i+1}.

2.2 NCL Semantics

Models for formulas in NCL are defined in the same way as classical logic. An interpretation is a model of a set *X* of formulas iff it satisfies all the formulas in *X*. If every model of *X* satisfies a formula *F*, then *X* entails *F*, or symbolically $X \models F$. We can now define models of a causal theory. Let *I* be an interpretation of the signature α of a theory *T*. The reduct T^I is the set of all heads whose bodies are satisfied by the interpretation *I*. If *I* is also a unique model of T^I, then *I* is a model of *T*. If *I* is not a unique model of the reduct, then some constant is missing from the reduct, and therefore there is no explanation for that constant. But NCL is a logic of universal causation, therefore, no constant should be unexplained.

As an example, consider the following theory T_1 consisting of rules *R1* and *R2*.

R1. $p \Leftarrow q$
R2. $q \Leftarrow q$

R1 says that there is a cause for *p* if *q* is true. R2 says that there is a cause for *q*. Reasoning informally using the principle of universal causation, we see that there is no cause for $\neg p$ to be true. Therefore, *p* has to be true. Therefore, *p* must be caused. The only way *p* can be caused is if *q* is true. And, *q* is caused by R2. Therefore, the only possible interpretation for this theory is $I = \{p = true, q = true\}$.

Intuitively, T^I represents facts that are caused, according to theory *T* under interpretation *I*. If a causal theory *T* has a model *I*, we say that it is consistent or satisfiable. If all models of *T* satisfy a formula *F*, that means *T* entails *F* or $T \models F$.

Coming back to our example theory T_1 with rules R1 and R2, we consider all the possible interpretations to see which, if any, is a model of T_1:

1. $I^1 = \{p = true, q = true\}$: $T_1^{I_1} = \{p, q\}$. I_1 is a unique model of $T_1^{I_1}$. Therefore, I_1 is a model of T_1.
2. $I^2 = \{p = false, q = true\}$: $T_1^{I_2} = \{p, q\}$. I_2 is not a model of $T_1^{I_2}$. Therefore, I_2 is not a model of T_1.
3. $I^3 = \{p = true, q = false\}$: $T_1^{I_3} = \{\}$. $T_1^{I_3}$ has no models. Therefore, I_3 is not a model of T_1.
4. $I^4 = \{p = false, q = false\}$: $T_1^{I_4} = \{\}$. $T_1^{I_4}$ has no models. Therefore, I_4 is not a model of T_1.

Note that $I^1 = \{p = true, q = true\}$ is the only model of this theory that matches the result of our informal reasoning.

To see how NCL is nonmonotonic, consider a theory T_2 consisting of the rule $c = 1 \Leftarrow c = 1$. This rule is like a default rule. The only model of T_2 is *I(c) = 1*. Now consider a theory T_3 such that it had two rules, $c = 1 \Leftarrow c = 1$ and $c = 2 \Leftarrow true$. Note that $T_2 \subseteq T_3$. However, the only model of T_3 is *I(c) = 2*.

2.3 Action Descriptions in C+

Recall that *C*+ is a high level action description language based on NCL. It is easier to specify theories in *C*+ than directly in NCL because of it's concise notation. Before we describe the syntax and semantics of NCL and *C*+ formally, we describe informally the meanings of some of *C*+ rules that we use later. A formula in *C*+ is a propositional combination of constants which could either be *action* constants or *fluent* constants. Actions in NCL are interpreted to be unit-length. This paper is restricted to boolean constants. An action constant being true represents the execution of the action. The meanings of the rules we use follow.

- *a causes b*, where *a* and *b* are actions.
  This means that action *a* causes action *b* and both happen concurrently.
- *a causes f*, where *a* is an action and *f* is a fluent.
  This means that action *a* causes *f* to hold in the next state.
- $A \wedge F$ *causes b*, where *A* is a conjunction of actions, *b* is an action and *F* is a conjunction of fluents.
  This means that in a state where *F* is true and actions in *A* happen, then action *b* happens concurrently.
- *a causes a*, together with the rule $\neg a$ *causes* $\neg a$, where *a* is an action.
  These two rules mean that there is a cause for *a* and there is a cause for $\neg a$ respectively. In other words the *a* is exogenous, it simply happens or does not happen. Without these rules, the action is not exogenous. Universal causation is disabled for these rules.
- *a may cause f*, where *a* is an action and *f* is a fluent.
  This means that *f* may hold after *a*'s execution if it does not already hold. Thus, this rule expresses nondeterminism.
- *caused a after f*, where *a* is an action and *f* is a fluent.
  This means that *f* causes *a* in the same state. Notice that this rule uses the *caused* form and not the *causes* form because no suitable formulation in terms of *causes* exists. This rule expresses the causation of an action and differs from the rule *a causes f* above which expresses the causation of a fluent.
- Fluents are declared as *inertialFluents* meaning that their assignment persists from one state to the next unless changed by some other rule.

2.4 Translating C+ to NCL

Let's describe *C*+ formally and show the translation from rules in *C*+ to rules in NCL. *C*+ includes three kinds of rules, namely, static rules, fluent dynamic rules and action dynamic rules. A fluent can be either a *statically determined fluent* or a *simple (dynamic) fluent*. Static fluents can appear in the heads of only static rules. A *fluent formula* consists only of fluents. An *action formula* consists only of actions. A *static rule* is an expression of the form

R3. *caused F if G*

where *F* and *G* are fluent formulas. Static rules express indirect effects of actions that are instantaneous with respect to the causal fluent formula. A dynamic rule is an expression of the form

R4. *caused F if G after H*

It is called an *action dynamic rule* if *F* and *G* are both action formulas. It is called a *fluent dynamic rule* if *F* and *G* are both fluent formulas. Action dynamic rules express the causation of an action and fluent dynamic rules the causation of a fluent.

An action description D consisting of such rules is turned into a causal theory D_m where *m* is the length of the history. The signature α of D_m then consists of

1. i:c for every fluent $c \in \alpha$ for every $i \in \{0, \ldots, m\}$
2. i:c for every action $c \in \alpha$ for every $i \in \{0, \ldots, m\text{-}1\}$

The domain of i:c is the same as *Dom(c)* and i:F means i is inserted in front of every occurrence of every constant in *F*. The rules of D_m are then:

R5. i:$F \Leftarrow i$:G, for every static rule R3 in *D* and every $i \in \{0, \ldots, m\}$;

R6. i:$F \Leftarrow i$:$G \wedge i$:H, for every action dynamic rule R4 in *D* and every $i \in \{0, \ldots, m\text{-}1\}$;

R7. $i{+}1$:$F \Leftarrow i{+}1$:$G \wedge i$:H, for every fluent dynamic rule R4 in *D* and every $i \in \{0, \ldots, m\text{-}1\}$;

R8. 0:$c = v \Leftarrow 0$:$c = v$, for every simple fluent constant $c \in \alpha$ and every $v \in$ *Dom(c)*. (Notice that every simple fluent has all possible values in the initial state and therefore, they are exogenous in the initial state. Thus, universal causation is disabled for them.)

All examples that follow are in the abbreviated *C+* notation. We list the relevant *C+* abbreviations below, extracted from [3].

A1. A dynamic rule of the form *caused F if true after H* abbreviates to *H causes F*.
A2. An action dynamic rules of the form *caused F if G after H* abbreviates to $G \wedge H$ *causes F*
A3. The action dynamic rules *caused a if a* and *caused ¬a if ¬a* where *a* is an action, together abbreviate to *exogenous a*. In *C+* such an action is called an *exogenousAction*.
A4. The fluent dynamic rules *caused p if p after p* and *caused ¬p if ¬p after ¬p* where *p* is a fluent together abbreviate to *inertial p*
A5. The fluent dynamic rule *caused F if F after H* abbreviates to *H may cause F*.

3 Commitments

Commitments among agents have been recognized as a fundamental notion in cooperative problem solving [7–9]. Castelfranchi [10] and Krogh [11] present, respectively, a social and logical perspective on commitments. In our work, we do not reason about the commitments from the point of view of cooperation among agents. We use commitments to specify protocols. As agents interact with each other using some protocol, they create and manipulate commitments. The breach of a commitment represents a violation of a protocol. The agent that is bound to fulfi ll the commitment is called the debtor of the commitment. The agent that is the benefi ciary of the commitment is called the creditor.

Definition 1. *A base-level commitment C(x,y,G,p) binds a debtor x to a creditor y for fulfilling the condition p in context G.*

Definition 2. *A conditional commitment CC(x,y,G,p,q) denotes that if a condition p is brought about, then the commitment C(x,y,G,q) will hold.*

Both commitments and conditional commitments are created in a context *G*, which can be thought of as an institution or society whose rules are binding on the agents that join it. The context also defi nes the meanings of the terms used in the context. Henceforth, we omit *G* to reduce clutter.

Singh[12] lists operations for the creation and manipulation of commitments. These operations cannot be arbitrarily carried out. They are subject to metacommitments that are rules that govern the commitment operations and are part of the context *G*. The operations are listed below.

- *Create(x,y,p)* creates a new commitment *C(x,y,p)*.
- *Discharge(x,y,p)* discharges the existing commitment *C(x,y,p)* so that it no longer holds.
- *Cancel(x,y,p)* cancels the existing commitment *C(x,y,p)* so that it no longer holds.
- *Delegate(x,y,p,z)* delegates the commitment *C(x,y,p)* to a new debtor *z*. More specifically, the original commitment *C(x,y,p)* no longer holds and a new commitment *C(z,y,p)* is created in its place.
- *Assign(x,y,p,z)* assigns the commitment *C(x,y,p)* to a new creditor *z*. More specifically, the original commitment *C(x,y,p)* no longer holds and a new commitment *C(x,z,p)* is created in its place.
- *Release(x,y,p)* releases the debtor *x* from the commitment *C(x,y,p)* so that the commitment no longer holds.

4 NCM Representation of Protocols

In our approach, we represent protocols as NCMs. An NCM is a causal theory in *C+* that consists of two parts. The fi rst part is a protocol-independent causal theory of commitments in which we capture the representation of commitments and the operations on them. The second part is protocol specifi c and includes constants and rules describing the given protocol's domain. The distinction between the two parts is only to separate out the domain independent part, logically they form a complete causal theory as we shall see later. We first present the theory of commitments and then model the NetBill protocol in causal logic. Together they represent the NetBill NCM.

4.1 Commitments in Causal Logic

We represent commitments and operations on them in the causal logic. Commitments in causal logic are declared to be constants of the type inertial fluents.

- *C(x,y,p), CC(x,y,p,q) :: inertialFluents*

where *x* and *y* are variables of the sort *agent* and *p* and *q* are variables of the sort *condition*. By declaring commitments to be *inertialFluents*, we include rules of the kind A4 for each commitment. Conditional commitments are declared as *CC(x,y,p,q)* where *q* is also a variable of sort *condition*. Conditional commitments are also declared as *inertialFluents*. For each of the operations on commitments listed in Section 3, there is a declaration of the form

- $\langle$ *Operation* $\rangle$ *:: action*

Constants of type *action* are not exogenous, that is, rules of the form A3 are not included in the theory. Their execution, therefore, has to be *caused* by other actions or fluents. We add two more operations for handling conditional commitments.

- *CDischarge(x,y,p,q), CCreate(x,y,p,q) :: action*

The following rules capture the meaning of the operations:

R9. *Create(x,y,p) causes C(x,y,p)*
R10. *Discharge(x,y,p) causes* $\neg$*C(x,y,p)*
R11. *Cancel(x,y,p) causes* $\neg$*C(x,y,p)*
R12. *Delegate(x,y,p,z) causes* $\neg$*C(x,y,p) & C(z,y,p)*
R13. *Release(x,y,p) causes* $\neg$*C(x,y,p)*
R14. *Assign(x,y,p,z) causes* $\neg$*C(x,y,p) & C(x,z,p)*
R15. *CCreate(x,y,p,q) causes CC(x,y,p,q)*
R16. *CDischarge(x,y,p,q) causes* $\neg$*CC(x,y,p,q) & C(x,y,q)*

All the variables are grounded such that $x \neq y$ and $p \neq q$. We omit the rules specifying the grounding of the variables. Since we want the operations to be *caused* by other things, then in those states where an operation is not caused, there must be a reason for it to be not caused. In other words, we want the operations to be partially exogenous. So for each of the commitment operations, we include rules of the form

R17. $\neg\langle$ Operation $\rangle$ *causes* $\neg\langle$ Operation $\rangle$

An example is the rule $\neg$*Create(x,y,p) causes* $\neg$*Create(x,y,p)*. The specification also includes rules to capture the restriction that no two commitment operations are concurrent.

4.2 NetBill Specification in Causal Logic

We now represent the NetBill protocol in causal logic. The following rules together with the theory of commitments given above represent the specifi cation of NetBill NCM. We declare

- *m, c* to be of the sort *agent*
- *goodsc, payc, acceptc, receiptc* to be of the sort *condition*
- *request, offer, accept, goods, pay, receipt* to be *inertialFluents*
- *SendRequest, SendOffer, SendAccept, SendGoods, SendPayment, SendReceipt* to be *exogenousActions*.

The meanings of the above constants are as their name indicates. Conditions are an artifact of conditional commitments. We assume that all fluents and actions have unique identifi ers. We have the following rules.

R18. *SendRequest causes request*
R19. *SendOffer causes offer*
R20. *SendOffer causes CCreate(m, c, acceptc, goodsc)*
R21. *SendAccept causes accept*
R22. *SendAccept & CC(m,c,acceptc,goodsc)*
causes CDischarge(m,c,acceptc,goodsc)
R23. *SendAccept causes CCreate(c,m,goodsc,payc)*
R24. *SendGoods causes goods*
R25. *SendGoods causes CCreate(m, c, payc, receiptc)*
R26. *SendGoods & CC(c,m,goodsc,payc) causes CDischarge(c,m,goodsc,payc)*
R27. *SendGoods & C(m,c,goodsc) causes Discharge(m,c,goodsc)*
R28. *SendPayment causes pay*
R29. *SendPayment & CC(m,c,payc,receiptc) causes CDischarge(m,c,payc,receiptc)*
R30. *SendPayment & C(c,m,payc) causes CDischarge(c,m,payc)*
R31. *SendReceipt causes receipt*
R32. *SendReceipt & C(m,c,receiptc) causes Discharge(m,c,receiptc)*

In our representation no two commitment operations are concurrent. Also, no two protocol actions are concurrent. However, when a protocol action causes a commitment operation, they are concurrent. By rule R6, the interpretation of *ActionA causes ActionB* is such that *ActionA* and *ActionB* are concurrent. This ensures that the protocol action is concurrent with the commitment operation it causes is satisfi ed. There could be other concurrency models possible for NetBill. We choose this one because of its simplicity.

We now add rules for our motivating example, Example 1, to this protocol specifi - cation. We introduce *SendReturn* and *SendGoods* as exogenous actions. We introduce a new action *Ab* and a new fluent *damagedGoods* to indicate that the goods are damaged. We also include the nondeterministic rule R34 to say that as a result of the *SendGoods* action, the goods may be damaged. Rule R33 captures the condition that the cancel operation is not allowed for any commitment. Rule R35 however says that a commitment can be canceled under abnormal conditions. Rules R36 and R37 ensure that *Ab* is false, except when *damagedGoods* is true. We add the Rules R35 − R37 to accommodate Example 1. Rules R33 and R34 are already in the theory. *Ab* is an action because it has no meaning in the states. It acts as a qualifi er for the exogenous action *SendReturn*.

R33. *¬Cancel(x,y,p) causes ¬Cancel(x,y,p)*
R34. *SendGoods may cause damagedGoods*
R35. *SendReturn ∧ C(c,m,pay) ∧ Ab causes Cancel(c,m,pay)*
R36. *caused Ab if damagedGoods*
R37. *¬Ab causes ¬Ab*

The theory also contains rules that place constraints on the actions so that the execution of actions makes sense. For example, we specify that the *SendRequest* action cannot happen after the payment has been made. Rules

4.3 Executing NetBill in CCalc

CCalc (Causal Calculator) is a reasoning tool that implements causal logic. Given a causal theory and a goal in the form of a query, *CCalc* finds paths to the goal. We load the NetBill NCM into *CCalc* and pose queries one after the other. *CCalc* then finds paths to satisfy each query.

```
:- query
label::0;
maxstep:: 3;
0: -offer,
   -accept,
   -returned,
   -goods,
   -C(x,y,p),
   -CC(x,y,p,q),
   -pay,
   -request,
   -receipt,
   -damagedGoods;
maxstep: returned.
```

Fig. 2. Example Query

Figure 2 shows an example query. This query asks for an execution sequence of three of fewer steps, beginning from a state in which all fluents are false, in which the goods have been returned. Running this query in *CCalc* produces the output as shown in Figure 3 (formatted for readability).

The action *SendAccept* is caused which in turn causes the *CCreate* which creates the conditional commitment that if the goods are sent then the customer will pay. The result of these actions is reflected in state 1. *SendGoods* is then caused which reflects the fact that the merchant has sent the goods. *SendGoods* causes the discharge of the conditional commitment created by the customer resulting in the customer's commitment to pay. *SendGoods* also creates a new conditional commitment that if the customer

```
Solution:

State 0:

ACTIONS:  CCreate(c,m,goodsc,payc)
          SendAccept

State 1:  CC(c,m,goodsc,payc) accept

ACTIONS:  CCreate(m,c,payc,receiptc)
          CDischarge(c,m,goodsc,payc)
          SendGoods

State 2:  C(c,m,payc) CC(m,c,payc,receiptc)
          accept goods damagedGoods

ACTIONS:  Cancel(c,m,payc) Ab Return

State 3:  CC(m,c,payc,receiptc) accept
          goods returned damagedGoods
```

Fig. 3. Answer

pays then the merchant will send the receipt. This example is interesting as *SendGoods* also causes *damagedGoods*, resulting in state 2. *damagedGoods* causes *Ab* in state 2. So the *SendReturn* action is successful, which in turn causes the cancellation of the commitment to pay. State 3 is the resulting state which also satisfies the goal state of our query.

5 Discussion

Our focus in this work is to develop meaningful representations of agent communication protocols. We do so by using commitments to declaratively represent states and actions. This gives our representation a verifiable semantics [13]. By using commitments to model protocols, we constrain protocols no more than is necessary. We have highlighted the need for a nonmonotonic logic for commonsense reasoning in protocols. We used NCL towards this end and showed how a protocol can be represented in NCL. Like the NCL, there are a few other noteworthy formalisms for reasoning about action and change. Dynamic Logic [14] is a modal logic augmented with an algebra of regular events. However, it is monotonic and therefore not suitable for our purposes. Event Calculus [15] and situation calculus [16] have been extended with circumscription to enable nonmonotonic reasoning.

Both FSMs and NCMs are formal representations of protocols. Both approaches are verifiable and in the case of an FSM, trivially so. An NCM, though, represents meaning.

Agents that can reason about commitments take actions accordingly. For example, if, as part of a particular protocol, an agent enters into a commitment, then the agent can plan its actions, even those not directly related to the protocol, so that the commitment is never violated. Alternate paths through the protocol may be selected based on criteria like safety or number of messages exchanged. For example, an agent can adopt an approach where it does not commit unless another agent also commits for some desirable condition. Also it is not convenient to express defaults in FSMs. In fact, the defaults wouldn't be obvious at all in an FSM. FSMs are also not as elaboration tolerant as NCMs.

5.1 Conventional Protocols and Protocol Modeling

Conventional protocols like TCP/IP, RPC, HTTP, and so on have a well-defined environment and scope. Their focus is on the correct delivery of data and they are therefore strict in the sense that they prescribe all paths for correct execution as well as for error recovery. Modeling such protocols as FSMs is usually sufficient. Notable among other formalisms for modeling protocols are Petri Nets [17] and statecharts [18]. Petri Nets have proven to be especially useful in modeling concurrency. Petri nets specify transitions (T-elements) between places (P-elements) which are sets of conditions. Statecharts is a visual formalism that extends state machines by adding support for hierarchy, concurrency and communication. Statecharts provide the designer the power to cluster states into super-states as well as refine states, thereby leading to compact representations for complex behavior. Statecharts can also represent defaults. However, statecharts also specify the transitions between states. As such, neither petri nets nor statecharts afford much flexibility. Statecharts though, are easier to comprehend because of the ability to cluster and refine states unlike NCMs where the specification may become difficult to manage as the number of rules increase.

5.2 Commitments

Commitments have been studied in the context of distributed problem solving and coordination. Bratman [7] argues that for shared cooperative activity, among other things, commitment to joint activity and commitment to mutual support are required. Grosz and Kraus [9] investigate the formulation of shared plans for coordinating group action. In their framework, an agent can adopt two types on intentions, intend-to and intend-that, that commit the agent to an action and state of affairs respectively. Jennings [19] presents commitments as a fundamental notion for efficient coordination in distributed systems. Jennings also mentions conventions which monitor the commitments and state when a commitment may be reassessed. Jennings further reformulates different models of coordination in terms of commitments. A distinguishing feature of all of the above work is that they present commitments as a mentalistic notion, assuming a system of cooperative agents. Shoham's agent oriented computing paradigm [8] introduces obligation as a modality required to describe the mental state of an agent. Sandholm and Lesser [20] study automated negotiation among self-interested agents whose computations are resource bounded. They argue that protocols that have leveled commitments, that is, when commitments vary from breakable to unbreakable in a continuum by assigning a function to evaluate the cost of breach of each commitment, are more suitable for contracts than full commitment protocols. Krogh [11] examines the possibility of using of deontic logic for analyzing multiagent systems. Castelfranchi [10] presents an ontology of commitments with the aim of understanding organizational activity. He defines social commitment as a relation between two or more agents and discusses its various aspects. Singh [13] defends a commitment-based social semantics for agent communication.

We do not study commitments from the point of view of coordination. However as we pointed out earlier in this section, an agent can reason about its future actions depending upon the commitments it already has or ones that the agent might have to make in the future. Also, we do not specify that commitments necessarily have to be represented in an agent's state. We are exploring the possibility of compiling NCMs into FSMs that have no representations of commitments (see section 6 for details). We are also not concerned directly with the economic impact of the breach of a commitment. Though we specify protocols in a logical language using commitments, we do not present a deontic logic. Social commitments, that is, directed obligation from one agent to another, represent the cornerstone of our research. Our scope is limited to the flexible specification of protocols and their verification.

5.3 Protocols

Yolum and Singh [4] proposed commitment machines and also showed how event calculus can be used to represent protocols and generate new paths in the protocols [21]. They do not consider commonsense reasoning situations. Koning and Huget [22] describe a methodology for designing interaction protocols for multiagent systems. In their work, the focus is on reusability and modularity of protocols. They achieve this by composing protocols out of microprotocols using a formal language called *Communication Protocol Description Language* (CPDL). A formula in CPDL corresponds to an edge going from an initial state to a final state. The edge is labeled with a sequence of microprotocols which makes the protocol quite rigid. Koning and Huget also do not consider which microprotocols can be composed. This job is presumably left to the designer. With our commitment-based approach, it is possible for the agent to determine which protocols can be composed by looking at the states in the protocol.

5.4 Agent Communication Languages

An agent communication language (ACL) allows agents of heterogeneous designs to interact. Developing a semantics for agent communication languages (ACLs) that is expressive and verifiable has been a long standing goal of the agent community. To be verifiable, an ACL should have social semantics [23]. Earlier efforts at standardization of ACLs like Arcol [24] and KQML [25] promoted mental agency, that is, they were based on mental concepts like beliefs and intentions, and were therefore, not verifiable. The ongoing efforts at standardizing ACLs are based on communicative acts. The problem with giving a communicative acts based semantics is that it is not clear what the meanings of the communicative acts should be. Also it is not clear how many communicative acts are needed. Another challenge is relating the communicative acts to the conversation in which they occur. By giving semantics to protocols directly we are able to give a simple, operational characterization of protocols without getting bogged down with the above issues.

Dignum and van Linder [26], and Guerin and Pitt [27] define ACLs in term of communicative acts. Dignum and van Linder's framework considers four components, the *information* component, the *action* component, the *motivational* component and the *social* component as constituting an agent framework and formally describes and relates them. The framework is developed in dynamic logic which is monotonic. They postulate a COMMIT communicative act, but other communicative acts have mentalistic preconditions which makes the framework suitable only for a system of cooperative agents.

Guerin and Pitt propose an ACL specification in which declarative ACL specifications are given procedural interpretations. An ACL specification in their approach, consists of three parts: a *Converse Function* that specifies permissions and obligations for subsequent speech acts based on the conversation state, a *Protocol Semantics* that captures protocol dependent meanings of speech acts, and a *Speech-Act Semantics* that give the protocol independent part of the meaning. It is not clear how useful it is to model the protocol-independent part of the meaning, since most meaning comes from the protocol.

6 Future Directions

Our main aim in this work is to come up with protocols that have verifiable semantics and constrain the agent no more than to the extent necessary to carry out legal interactions. To carry our work further, we have identified the following future directions.

Compiling NCMs into FSMs: The preceding sections present commitment-based semantics for interaction protocols and show how an agent can reason with commitments. However, it is not necessary that agents be able to reason about commitments to execute a protocol. For some applications efficient execution might be important. It may also be the case that an agent designer wants to exclude certain risky or lengthy paths (behaviors) in the protocol. For such agents, it would be useful if we could compile an NCM into an FSM. The complete NCM need not be compiled into an FSM. A designer could selectively compile behaviors (sequences of transitions) from an NCM into an FSM. The FSM can then be executed without an inference engine. Another advantage of compilation instead of directly designing or extending an FSM is that it is not always clear how to add states and transitions to an FSM. Compilation from an NCM makes this process automatic. Figure 4 shows an example FSM for NetBill, that may be compiled from the action description presented in Section 4.2.

Since *C+* action descriptions have a transition system semantics, it should be relatively straightforward to compile NCMs into FSMs. An important future direction is to formally define NCMs in terms of *C+* action descriptions and present a procedure for compiling NCMs into FSMs. It is equally important to prove that the generated FSM

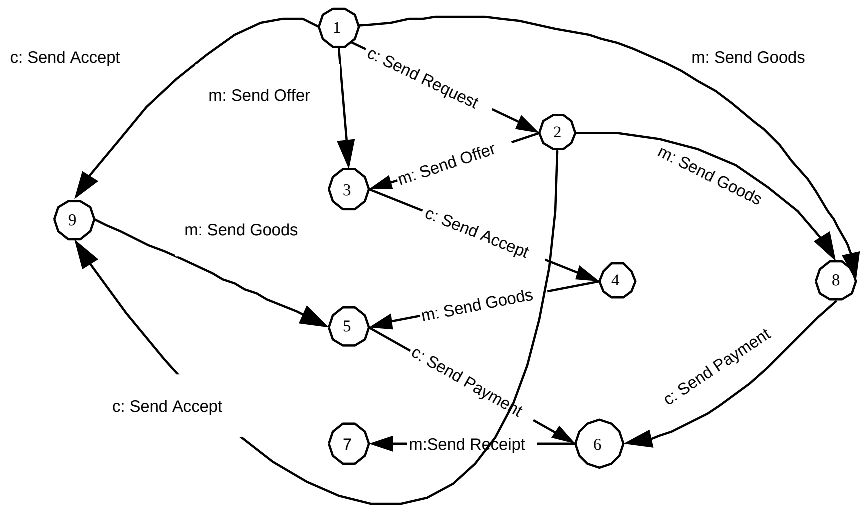

Fig. 4. FSM representation of the extended NetBill protocol

is sound and complete with respect to the NCM it was compiled from. We are also investigating ways to automatically compile FSMs from NCMs. Selective compilation is also a subject of future research.

Protocol Specification: While our formulation of NCM has some desirable properties like declarative rules and elaboration tolerance, it lacks other properties desirable in protocols such as a comprehensible graphical representation, role bindings for agents and temporal model checking [28]. For example, we cannot prove satisfactorily if the NetBill NCM is correct and complete. A related piece of work is to compare NCMs with other formalism like statecharts and Petri Nets in more depth. Another interesting avenue to explore is the compilation of NCMs into more expressive graphical formalism like statecharts.

Protocol Distribution: In the NCM representation of NetBill, we did not specify roles in the protocol. However, an agent will be executing the role that it adopts in the protocol. We want to formulate procedures based on symbol manipulation to distribute a centralized protocol among the various roles in the protocol. The result will be roles skeletons that are also NCMs. It is not clear what the specifi cation of a role itself should be. Is a role just a label, or does it specify the required capability of an agent to adopt a role along with normative rules and authorizations that come along with the role? Another technical challenge is proving that a distributed protocol is sound and complete with respect to the centralized protocol.

Creating Role Skeletons from BPEL Flows: BPEL [29], the Business Process Execution Language is a draft standard for specifying and coordinating business processes. Intuitively, it is a process fbw graph with nodes as tasks and edges as messages. Business fbws, as they are currently specifi ed, are like FSMs in the sense that they have been designed with certain scenarios in mind. As such, they are quite rigid. To make business processes more fexible, we envisage the following design-time methodology.

1. The designers start with an interaction diagram or state machine or some other graphical representation for the protocol that is to be modeled as a BPEL fbw.
2. The designers build NCM representation of the protocol with enhancements to make it more fexible and then partition the NCM into the role skeletons.
3. The role skeletons are compiled into FSMs
4. Compile FSMs into a BPEL fbw. This should be easier than compiling an NCM into a BPEL fbw.

We plan to build a tool which incorporates this methodology. The tool would suggest enhancements to the designer for a given protocol, build an NCM based on the choices of the designer and compile it into an FSM and then, perhaps with annotations from the user, compile it into a BPEL fbw.

Verifying Strategies: It will often be the case that an agent is confronted with the problem of selecting between multiple paths that it can take to reach a goal. The agent selects a path based on some strategy. For example, if the customer does not trust a merchant, the customer might adopt a strategy where it never pays before receiving the goods. On the other hand, if it does, it could adopt a strategy where it accepts to buy goods for a certain price without even asking the merchant for offers or pays before getting the goods. It is possible to imagine more complex strategies. An interesting problem is the specifi cation of strategies with respect to commitments and determining which protocols (more specifi cally, paths in the protocol) satisfy a given strategy. An agent can then inspect a role to check whether it satisfi es its strategy.

We conclude by saying that this work represents the fi rst step in the development of a comprehensive methodology for designing fexible multiagent interaction protocols. Development of the protocol design tool that incorporates this methodology is the primary objective of this research.

References

1. Smith, R.G.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers **29** (1980) 1104–1113
2. Cox, B., Tygar, J., Sirbu, M.: Netbill security and transaction protocol. In: Proceedings of the First USENIX Workshop on Electronic Commerce. (1995) 77–88

3. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artificial Intelligence (2003) To appear.
4. Yolum, P., Singh, M.P.: Commitment machines. In: Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL-01), Springer-Verlag (2002) 235–247
5. McCarthy, J.: Elaboration tolerance. In progress (1999) http://www-formal.stanford.edu/jmc/elaboration.html.
6. Lifschitz, V.: Missionaries and cannibals in the causal calculator. In: Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning. (2000) 85–96
7. Bratman, M.E.: Shared cooperative activity. The Philosophical Review **101** (1992) 327–341
8. Shoham, Y.: Agent-oriented programming. Artificial Intelligence **60** (1993) 51–92
9. Grosz, B.J., Kraus, S.: Collaborative plans for complex group action. Artificial Intelligence **86** (1996) 269–357
10. Castelfranchi, C.: Commitments: From individual intentions to groups and organizations. In: Proceedings of the AAAI-93 Workshop on AI and Theories of Groups and Organizations: Conceptual and Empirical Research. (1993)
11. Krogh, C.: Obligations in multiagent systems. In Åmodt, A., Komorowski, J., eds.: Scandinavian Artificial Intelligence Conference 1995 (SCAI'95), Trondheim (1995) 19–30
12. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law **7** (1999) 97–113
13. Singh, M.P.: A social semantics for agent communication languages. In: Proceedings of the 1999 IJCAI Workshop on Agent Communication Languages, Springer-Verlag (2000)
14. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge, MA (2000)
15. Kowalski, R., Sergot, M.: Logic-based calculus of events. New Generation Computing **4** (1986) 67–95
16. McCarthy, J.: Situations, actions and causal laws. TR, Stanford University (1963)
17. Girault, C., Valk, R.: Petri Nets for System Engineering. (2003)
18. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming **8** (1987) 231–274
19. Jennings, N.R.: Commitments and conventions: The foundation of coordination in multi-agent systems. Knowledge Engineering Review **2** (1993) 223–250
20. Sandholm, T., Lesser, V.: Issues in automated negotiation and electronic commerce: Extending the contract net framework. In: [30]. (1998) 66–73 (Reprinted from *Proceedings of the International Conference on Multiagent Systems, 1995*).
21. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: Applying event calculus planning using commitments. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), ACM Press (2002) 527–534
22. Koning, J.L., Huget, M.P.: A semi-formal specification language dedicated to interaction protocols. In Kangassalo, H., Jaakkola, H., Kawaguchi, E., eds.: Information Modeling and Knowledge Bases XII, Frontiers in Artificial Intelligence and Applications. IOS Press (2001)
23. Singh, M.P.: Agent communication languages: Rethinking the principles. IEEE Computer **31** (1998) 40–47
24. Sadek, D.: Compliance in Arcol (1997) Personal communication.
25. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an agent communication language. In: Proceedings of the International Conference on Information and Knowledge Management, ACM Press (1994) 456–463
26. Dignum, F., van Linder, B.: Modelling social agents: Towards deliberate communication. In: Handbook of Defeasible Reasoning and Uncertainty Management Systems, Kluwer (2002) 357–380

27. Guerin, F., Pitt, J.: Denotational semantics for agent communication languages. In: Proceedings of the Fifth International Conference on Autonomous Agents, ACM Press (2001) 497–504
28. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice-Hall, London (1991)
29. W3C: Business process execution language for web services, version 1.1. (2003) URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.
30. Huhns, M.N., Singh, M.P., eds.: Readings in Agents. Morgan Kaufmann, San Francisco (1998)