# Enhancing Commitment Machines

Michael Winikoff[1], Wei Liu[2], and James Harland[1]

[1] RMIT University, Melbourne, Australia
{winikoff, jah}@cs.rmit.edu.au
[2] University of Western Australia, Perth, Australia
wei@csse.uwa.edu.au

**Abstract.** Agent interaction protocols are usually specified in terms of permissible sequences of messages. This representation is, unfortunately, brittle and does not allow for flexibility and robustness. The *commitment machines* framework of Yolum and Singh aims to provide more flexibility and robustness by defining interactions in terms of the commitments of agents. In this paper we identify a number of areas where the commitment machines framework needs improvement and propose an improved version. In particular we improve the way in which commitments are discharged and the way in which pre-conditions are specified.

## 1    Introduction

Communications between software agents are typically regulated by interaction protocols. These include general communication protocols, such as the auction protocol and the contract net protocol, as well as more specific protocols such as the NetBill payment protocol [1, 2]. Traditional protocol representations such as Finite State Machines (FSM), Petri-Nets [3] and AUML sequence diagrams [4, 5] often specify protocols in terms of legal message sequences. Under such protocol specifications, agent interactions are pre-defined and predictable. However, the inevitable rigidity resulting from such protocols prevents agents from taking opportunities and handling exceptions in a highly dynamic and uncertain multi-agent environment.

Yolum and Singh's Commitment Machines [1] (CMs henceforth) define an interaction protocol in terms of actions that change the state of the system, which consists of not only the state of the world but also the *commitments* that agents have made to each other. It is the commitment made to an interaction partner that motivates an agent to perform its next action. In other words, an agent acts because it wants to comply with the protocol and provide the promised outcomes for another party. Actions in CMs not only change the values of state variables, but also may initiate new commitments and/or discharge existing commitments. In traditional protocol representations, agents are confined to perform some pre-defined sequence of actions, whereas in CMs, an agent is able to reason about the next action to be taken in accordance with the dynamics of the environment and the commitments. This fundamentally changes the process of protocol specification from a procedural approach (i.e., prescribing *how* an interaction is to be executed) to a declarative one (i.e., describing *what* interaction is to take place) [1].

Using commitments as the rationale for agent interactions allows protocols to be specified at a higher level, which then generates more flexible and robust interactions

than pre-defined sequences. For example, in the NetBill protocol (discussed in Section 2), a customer may wish to order goods without first receiving a quotation, or a merchant may be happy to send goods to a known reliable customer with less rigorous checking than normal.

In this paper we identify a number of areas where the Commitment Machine framework can be improved. Specifically, we show how the identification of undesirable states (such as omitting to provide a receipt, or receiving the goods before payment has been confirmed) can be incorporated into the design process in order to achieve acceptable outcomes for a wider variety of circumstances than is done in [1, 2]. We also show how certain anomalies in discharging commitments and in handling pre-conditions can be remedied.

We demonstrate the operation of the improved framework on some examples. This is necessarily limited to a small number of illustrative cases, but these suffice to demonstrate the generality of the improvements.

The paper is organized as follows: in Section 2 we introduce the commitment machine framework and a detailed example, both based on [1]. In Section 3 we identify a number of anomalies and issues with the commitment machines framework and in Section 4 we propose some improvements. In Section 5 we discuss further applications of the improved framework and in Section 6 we present our conclusions.

## 2    Background

We briefly introduce the commitment machines framework and the NetBill protocol. Both are based on the description in [1] and we refer the reader to [1, 2] for further details.

The key example used in [1] is the NetBill protocol [6]. In this protocol a customer buys a product from a merchant. To buy a desired product, the protocol begins with a customer (C) requesting a quote (message 1 in Figure 1) from the merchant (M), followed by the merchant sending the quote (message 2). If the customer accepts the quote (message 3), the merchant proceeds by sending the goods (message 4) and waits for the customer to pay by sending an electronic payment order (EPO). Note that it is assumed that the goods cannot be used until the merchant has sent the relevant decryption key, such as software downloaded from the internet, or sent on a CD. Once the customer has sent payment (via an EPO in message 5), the merchant will send the decryption key along with a receipt (message 6). This concludes the NetBill transaction.

As suggested by the name "commitment machine", a crucial concept is that of *commitment*. A (social) commitment is an undertaking by one agent (the *debtor*, $x$) to another agent (the *creditor*, $y$) to bring about a certain property $p$, written $C(x, y, p)$. A commitment of the form $C(x, y, p)$ is a *base-level* commitment. For example, in the NetBill protocol when the customer sends message 3 and then receives the goods, he or she has a commitment to pay the merchant, i.e., $C(C, M, pay)$.

When a party is willing to commit only if certain conditions hold (such as another party making a corresponding commitment), a *conditional commitment* can be used. A conditional commitment, denoted $CC(x, y, p, q)$, indicates that agent $x$ is committed to achieving $q$ for agent $y$ if $p$ becomes true. A conditional commitment is latent – it
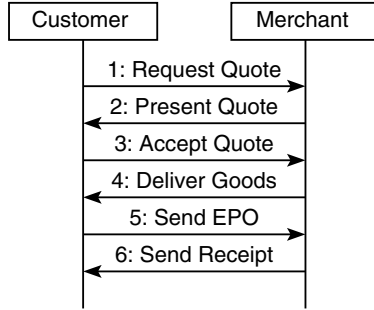
**Fig. 1.** Simplified Net Bill Protocol

doesn't commit $x$ to do anything until $p$ becomes true, at which point the conditional commitment is transformed to the base-level commitment $C(x, y, q)$. For example, in the NetBill protocol the customer may insist on his or her commitment to pay being conditional on the goods being sent, which would be represented as $CC(customer, merchant, goods, pay)$. Where the identity of the debtor and the creditor are obvious from the context we shall sometimes write $C(p)$ in place of $C(x, y, p)$ and $CC(p \rightsquigarrow q)$ in place of $CC(x, y, p, q)$.

Interactions are specified in the CM framework by defining the roles of the participants, the domain-specific fluents (i.e., boolean state variables), the (conditional) commitments that may arise during the interaction, and the rules for *initiating* and *terminating* commitments. Together, they define the preconditions and effects of (communicative) actions, and are used to regulate the choices of actions during protocol execution. The execution of a protocol is driven by the commitments that are in place: the desire to fulfil these commitments generates an action or actions to achieve them, which in turn may create new commitments or discharge existing ones. The NetBill protocol as a CM can be found in Figure 2.

A *state* in a CM is a triple $\langle F, CC, C \rangle$, where $F$ is a set of fluents, $CC$ is a set of conditional commitments and $C$ is a set of base-level commitments. A *final state* is a state that does not have undischarged base-level commitments. A final state may contain conditional commitments, since they are latent commitments that have not been activated. Formally, a state in a CM is a final state if $C = \emptyset$. Note that a final state in a CM is one where the interaction *may* end. However, it is also possible for interaction to continue from a final state. A *protocol run* consists of a sequence of actions that results in a final state.

A commitment machine places constraints on the sequence of agent actions that constitute the interaction. For example, if an agent has a commitment, then it must at some point fulfil its commitment[1]. However, commitment machines do not dictate or require that agents perform particular actions.

Each commitment machine implicitly defines a corresponding Finite State Machine[2] (FSM) where the states of the FSM correspond to the states of the CM and the transi-

---

[1] Commitments can also be discharged in ways other than being fulfilled [1].

[2] Actually, a variation of finite state machines, since there is no defined initial state.

**Roles:** $M$ (merchant), $C$ (customer)

**Fluents:**

- $request$ (the customer has requested a quote),
- $goods$ (the goods have been delivered to the customer),
- $pay$ (the customer has paid),
- $receipt$ (the merchant has sent the receipt)

**Commitments:**

- $accept = \mathsf{CC}(C, M, goods, pay)$: a commitment by the customer (to the merchant) to pay once the goods have been delivered.
- $promiseGoods = \mathsf{CC}(M, C, accept, goods)$: a commitment by the merchant to send the goods if the customer accepts. Since $accept$ is itself a commitment this is a nested commitment: $promiseGoods = \mathsf{CC}(M, C, \mathsf{CC}(C, M, goods, pay), goods)$.
- $promiseReceipt = \mathsf{CC}(M, C, pay, receipt)$: a commitment by the merchant to send a receipt once the customer has paid.
- $offer = promiseGoods \wedge promiseReceipt$: an offer is a commitment by the merchant (a) to send the goods if the customer accepts the offer, and (b) to send a receipt after payment has been made.

**Action Effects:** the following (communicative) actions are defined:

- $sendRequest$: this action by the customer makes the fluent $request$ true.
- $sendQuote$: this action by the merchant creates the two commitments $promiseGoods$ and $promiseReceipt$ (i.e., $offer$) and terminates (makes false) the fluent $request$.
- $sendAccept$: this action by the customer creates the commitment $accept$.
- $sendGoods$: this action by the merchant makes the fluent $goods$ true and also creates the commitment $promiseReceipt$.
- $sendEPO$: this action by the customer makes the fluent $pay$ true. This action is defined in [1] as having the pre-condition that the goods have been sent.
- $sendReceipt$: this action by the merchant makes the fluent $receipt$ true. This is defined in [1] as having the pre-condition that payment has been made.

**Fig. 2.** The NetBill Protocol as a Commitment Machine [1]

tions are defined by the effects of the actions. Figure 4 shows a (partial) view of the states and transitions corresponding to the CM defined in Figure 2. Final states (those with no undischarged base-level commitments) are shaded and dotted lines depict actions that are intended to be prevented by pre-conditions (but see Section 3.4). This figure is an extension of the figure given in [1, 2]. The table in Figure 3 gives the fluents and commitments that hold in each state.

## 3 Properties of CMs

In this section we discuss various properties of CMs as presented in [1, 2] and identify a number of areas where we propose improvements to the CM framework.

| No. | State |
|-----|-------|
| 1 | - |
| 2 | $request$ |
| 3 | M: $promiseReceipt \wedge promiseGoods$ |
| 4 | M: $promiseReceipt \wedge$ C$(goods)$, C: $accept$ |
| 5 | $goods$, M: $promiseReceipt$, C: C$(pay)$ |
| 6 | $goods, pay$, M: C$(receipt)$ |
| 7 | $goods, pay, receipt$ |
| 8 | $goods$, M: $promiseReceipt$ |
| 9 | C: $accept$ |
| 10 | $pay$, M: C$(receipt) \wedge promiseGoods$ |
| 11 | $pay, receipt$, M: $promiseGoods$ |
| 12 | $goods, receipt$ |
| 13 | $goods$, M: $promiseReceipt$, C: $accept$ |

**Fig. 3.** States and associated commitments and fluents

### 3.1    Explicit Labelling of Undesirable States

The presentation in [1, 2] presents protocols as defining states (in terms of the commitments of the agents and the fluents that hold). A query is then given and the reasoning module finds possible sequences of actions that lead to the requested state. For example, in [2] given the commitment machine defined in Figure 2, the reasoning module is asked to find sequences of actions that lead to a final state where goods have been received, payment has been made, and a receipt has been issued.

However, when designing interaction rules it is important to not only ensure that a desirable final state is possible, but also to ensure that undesirable states are not possible.

In this context when we talk about "desirable" and "undesirable" states we are talking from the perspective of the *designer* of the interaction, not from the perspective of an agent who will take part in the interaction. Roughly speaking, the designer should consider a state to be desirable if at least one agent desires it and no agents find it undesirable. A state should be considered undesirable if any agent finds it undesirable.

If an undesirable final state is determined to be possible then this can be fixed by either adding additional commitments so that the state is no longer final, or by adding pre-conditions so that the state can not be reached. It is *not* possible to fix undesirable final states by merely having the agents be aware of the undesirable state - if a state is undesirable to one agent, another agent may still perform an action that results in that state.

For example, in the NetBill protocol the desirable final states are those in which the goods have been delivered and paid for and a receipt has been given. Undesirable final states are those where only one or two of these three conditions hold; it is clearly undesirable to have the goods without payment, to have paid for the goods without getting a receipt, to have a receipt without payment, or to have paid without the goods being delivered. The final state where the goods have not been delivered, no payment has been made, and there is no receipt is acceptable, but not desirable (neutral). In Figure 4 state 7 is desirable, states 8,11,12 and 13 are undesirable, and states 1,2,3
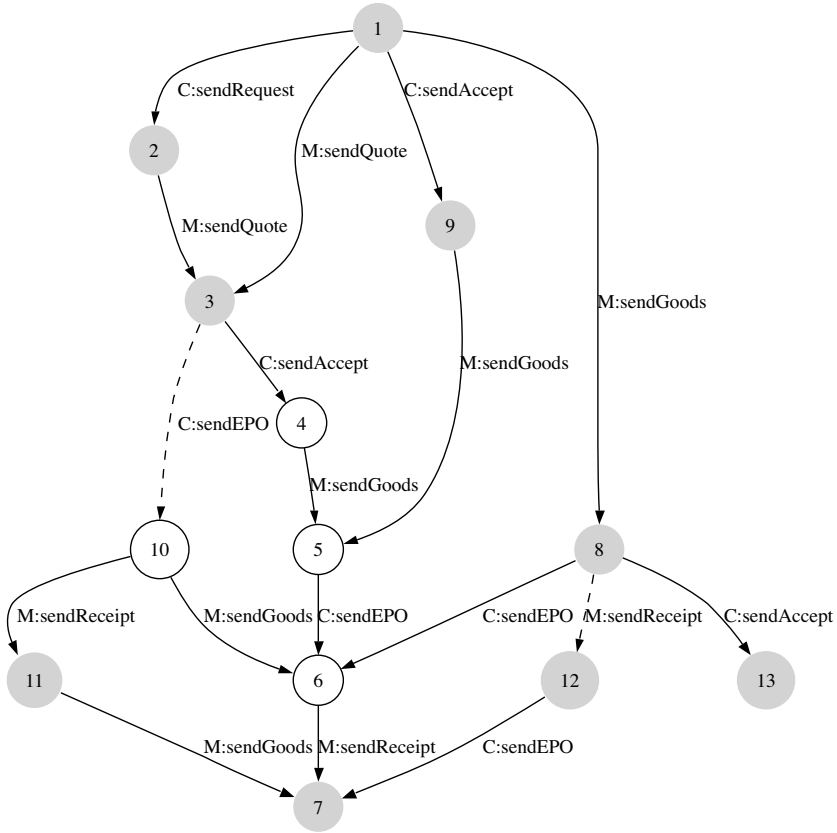
**Fig. 4.** Implied FSM for the NetBill CM (partial), final states are shaded and dotted lines depict actions that are intended to be prevented by pre-conditions

and 9 are neutral. Note that states 10, 11, 12 and 13 have been added to the machine discussed in [1, 2]. Note also that states 4,5,6 and 10 have undischarged commitments, and hence are not final states.

To illustrate why we need to identify and avoid undesirable states we consider an alternative protocol which seems quite reasonable. This protocol differs from the one presented in [1, 2] in that we remove the axiom:

$$Initiates(sendGoods, promiseReceipt, t)$$

This axiom is not needed in the "normal" expected sequence of actions (depicted in Figure 1) and it is quite possible that a naïve protocol designer would leave it out of an initial protocol specification.

Now suppose that the customer does not need a quote (perhaps the customer and merchant have interacted in the past), and begins the interaction with $sendAccept$. The merchant replies to the $sendAccept$ with $sendGoods$. At this point in the interaction the customer's acceptance commitment $\mathsf{CC}(good \rightsquigarrow pay)$ becomes a commitment to pay,

$C(pay)$, since the goods have been received. The customer then fulfils their obligation by paying. At this point we are in a final state — there are no remaining commitments — and goods have been received and payment made. However, this state is an undesirable one because the customer has not received a receipt.

The important point is that the omission of the *Initiates* rule is detected by checking whether undesirable (final) states are reachable, rather than by only checking whether desirable ones can be reached. If we had simply taken the variant protocol and asked for sequences which result in goods being delivered along with payment and a receipt then the problem would not have been noticed. In other words, the undesirable states can be used as a check on the interaction rules, which in this case results in the problem being easily found.

### 3.2    Failure to Discharge Conditional Commitments

There are anomalies in the rules that govern the discharge of conditional commitments. These anomalies can, in certain situations, result in conditional commitments not being discharged when, intuitively, they ought to be.

Consider the following sequence of steps:

1. The customer asks for a quote
2. The merchant replies with a quote. At this point the merchant has promised to send the goods if the customer accepts, and has promised to send a receipt if the customer pays.
3. The customer, misunderstanding the protocol perhaps, decides to accept but sends payment instead of an acceptance.

At this point the merchant becomes committed to sending a receipt, which it does, resulting in the following final state:

- fluents: $pay$, $receipt$
- commitments of merchant: $CC(CC(goods \leadsto pay) \leadsto goods)$

The crucial point here is that this is a final state and the merchant is not committed to sending the goods. The reason is that in order for $CC(CC(goods \leadsto pay) \leadsto goods)$ to become $C(goods)$ the commitment $CC(goods \leadsto pay)$ must hold: it is not enough according to the formal framework for $pay$ to hold. This is counter-intuitive because $pay$ is stronger than $CC(goods \leadsto pay)$ in that it discharges the commitment. The formal framework does recognise this, but only at the top level – the reasoning process that discharges $CC(goods \leadsto pay)$ when $pay$ becomes true is not applied to nested commitments.

### 3.3    Commitment Discharge Is Not Symmetrical

The axiom/postulate defining the conditions when a commitment (or conditional commitment) is discharged says that the commitment is discharged when it already exists and its condition is brought about by an event.

A problem with this is that it is possible to create a commitment $C(p)$ when $p$ already holds. This commitment will not be discharged unless an event takes place subsequently which re-initiates $p$.

For example, consider the following sequence:

1. The customer sends an accept. The customer has now committed to paying if the goods are received ($CC(goods \rightsquigarrow pay)$))
2. The merchant sends the goods. Since the goods have been sent, the customer now is committed to paying ($C(pay)$)).

However, lets consider what happens if the two steps occur in the reverse order:

1. The merchant sends the goods to the customer[3]
2. The customer sends an accept.

What is the resulting state? When sending the acceptance the customer initiates the conditional commitment to pay if the goods are received. This conditional commitment, however, does *not* become a commitment to pay even though the goods have already been sent. Consequently, the resulting state has no base-level commitments and so is an (undesirable) final state (state 13 in Figure 4).

### 3.4    Pre-condition Mechanism Does Not Prevent Action

A standard view of actions that goes back to STRIPS is that an action definition contains a pre-condition and a post-condition. The formalization of actions in the CM framework uses these, but the way in which pre-conditions are handled has a slight problem.

Pre-conditions in a CM are defined by putting conditions on the action effect definitions. For example, in [1] the effects of the $sendEPO$ action are defined using the clause[4]

$$Initiates(sendEPO, pay, t) \leftarrow HoldsAt(goods, t)$$

The intended reading in line with traditional pre-conditions is that "the goods must have been delivered in order for payment to be possible"[5] However, what this formalization actually does is limit the *effects* of $sendEPO$ rather than the action itself. In the event calculus the causality between $sendEPO$ and $pay$ is captured by the predicate $Intiates(sendEPO, pay, t)$, *not* by the implication. The implication only places a condition on when the *causality* holds, not on when the action may be performed. Thus, this does not prevent the event $sendEPO$ from occurring if $goods$ is false, it merely means that if the event $sendEPO$ occurs without $goods$ being true then the fluent $pay$ does not become true as a result of $sendEPO$.

From the perspective of the reasoning mechanism, the formalisation of pre-conditions introduces an additional link from a state back to that state that corresponds to performing an action whose pre-conditions are not satisfied (see Figure 5).

This is a fairly subtle difference but it does have one significant implication: if we consider agents that use an implementation of commitment machines to reason about

---

[3] As discussed in [2–example 2], this may be a sensible strategy if the goods are cheap to copy - e.g., software.

[4] Notation has been slightly changed. The actual clause in [1] is: $Initiates(sendEPO(i, m), pay(m), t) \leftarrow HoldsAt(goods(i), t)$.

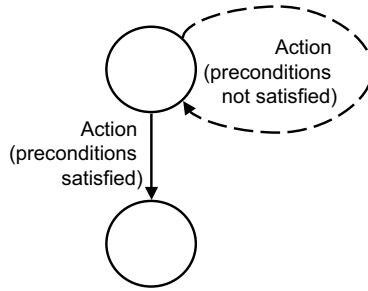[5] Note that this is the only $Initiates$ clause concerning $sendEPO$.

**Fig. 5.** Additional action link created by incorrect formalisation of pre-conditions

what actions to perform, then, for example, a customer agent who has not received the goods is not prevented from executing the $sendEPO$ action. Although the reasoning module will, in this case, believe that the effects of payment have not taken place, if the $sendEPO$ action is executed resulting in credit card details being sent, then in the real world the action's execution *will* have resulted in the undesired effect of payment.

### 3.5     Communication Mode Assumptions Not Clear

The state space defined by the available events (actions) includes sequences of events where an event representing an action by an agent (e.g., the merchant) is followed by an event representing another action by the same agent. This may not be desirable, if the intention is to define interactions where a message from $M$ to $C$ can only be followed by a response from $C$ to $M$.

The point here is that in the CM framework, there is no explicit specification of how the conversation should be carried out between the two parties, i.e., whether it should follow a synchronous mode or an asynchronous mode. Were the synchronous communication mode clearly specified, the action $sendReceipt$ by the Merchant would have been prevented in state 8 as the actors for the incoming and outgoing arc are the same.

However, there are situations where consecutive actions from the same agent *are* desirable. A typical CM state that may result in multiple actions from the same agent (or simultaneous actions from multiple agents) would have more than one base level commitment. See Section 4 for an example of a state with multiple base level commitments (state 10 in Figure 9).

A related issue is that the axioms allow an agent to perform actions that have no effect. For example, in the state where a request has been sent, sending another request has no effect on the state. In the FSMs that we show these arcs from a state $S$ to itself have been elided.

We do not address these issues in this paper; we will return to them in subsequent work.

## 4     Proposed Extended CM Model

In this section we propose an extended CM model which addresses some of the concerns discussed in the previous section.

### 4.1    Labelling Undesirable States

This isn't a change to the model so much as an extension and a change to how it is used (the methodology). As part of developing the commitment machine the designer indicates which states are undesirable (bad), which are desirable (good) and which are acceptable but not desirable (neutral). Indicating the desirability of states can be done by specifying conditions. For example, one could specify that all final states which satisfy $pay \wedge \neg receipt$ are undesirable.

The indication of good/bad states is specific to a particular interaction and the preferences of the parties involved. For example, in [2–example 2] where the goods are cheap to copy, the merchant may not consider state 8 in Figure 4, which has $goods$ but not $pay$ or $receipt$, to be a bad state.

The desirability of states, particularly of those states that are undesirable, is then used to perform safety checking.

### 4.2    Issues with Commitment Discharge

We now present a revised axiomatisation that remedies both anomalies associated with commitment discharge (Sections 3.2 and 3.3). We first consider the issue discussed in Section 3.2. Our proposed solution involves treating certain commitments as being "implied". For example, if $pay$ is true, then any commitment of the form $CC(X \rightsquigarrow pay)$ that occurs as a condition can be treated as having implicitly held (and been discharged).

We introduce predicates $Implied$ and $Subsumes$ which capture when a commitment (base or conditional) holds implicitly or is subsumed by a condition. These are used in the rules that govern commitment dynamics. When checking whether a condition $p$ holds, we also check whether it is implied or subsumed[6].

In [2], *fluents* are initiated directly by an action through the *initiates axioms* (e.g. $Initiates(sendGood, goods, t)$). On the other hand, *commitments*, both base-level and conditional, are created through the *Create* axioms. The *Create* axioms then initiate the commitments when the action happens according to the first axiom in Figure 6. However, this does not adequately distinguish commitments from other fluents. In particular, it makes no difference if we remove the first axiom in Figure 6 and replace all the *Create* axioms with *Initiates* axioms. In other words, according to the original commitment axioms and protocol specification, we can use *initiates* axioms for not only the fluents, but also all commitments. Such specifications dictate the action effects regardless of the current state of the world. However, an action should only initiate a base level commitment when the committed fluent is not already true, or when the premises but not the conclusion of a conditional commitment are true. An action should only initiate a conditional commitment when neither its premises nor conclusion are true. The axioms in Figure 6 fail to address this, which results in the asymmetrical discharge of commitments (Section 3.3).

In order to make commitment discharge symmetrical (Section 3.3) we de-couple intended causation from actual causation: instead of stating that an action initiates a

---

[6] $Implied(p, t)$ checks whether $p$ is implied at time $t$ and is used to check whether a condition (implicitly) holds at the current time. $Subsumes(p, p')$ checks whether $p$ subsumes $p'$ and is used to check whether an event would cause a condition to (implicitly) hold.

$Initiates(e, \mathsf{C}(x, y, p), t) \leftarrow Happens(e, t) \wedge Create(e, x, \mathsf{C}(x, y, p))$
$Initiates(e, \mathsf{CC}(x, y, p, q), t) \leftarrow Happens(e, t) \wedge Create(e, x, \mathsf{CC}(x, y, p, q))$

$Terminates(e, \mathsf{C}(x, y, p), t) \leftarrow Happens(e, t) \wedge Discharge(e, x, \mathsf{C}(x, y, p))$
$Discharge(e, x, \mathsf{C}(x, y, p)) \quad \leftarrow \quad HoldsAt(\mathsf{C}(x, y, p), t) \wedge Happens(e, t) \wedge$
$Initiates(e, p, t)$

$Initiates(e, \mathsf{C}(x, y, p), t) \quad \leftarrow \quad HoldsAt(\mathsf{CC}(x, y, p, q), t) \wedge Happens(e, t) \wedge$
$Initiates(e, p, t)$
$Terminates(e, \mathsf{CC}(x, y, p, q), t) \leftarrow HoldsAt(\mathsf{CC}(x, y, p, q), t) \wedge Happens(e, t) \wedge$
$\qquad Initiates(e, p, t)$
$Terminates(e, \mathsf{CC}(x, y, p, q), t) \leftarrow HoldsAt(\mathsf{CC}(x, y, p, q), t) \wedge Happens(e, t) \wedge$
$\qquad Initiates(e, q, t)$

**Fig. 6.** Commitment Machine Axiom 2,3,8,9,10 from [2]

commitment (e.g., $Initiates(sendGoods, promiseReceipt, t)$), we state that the action is *intended* to cause the initiation of the commitment (e.g., $Causes(sendGoods, promiseReceipt)$). We then link the two notions by defining $Initiates$ in terms of $Causes$. When $p$ is a fluent (not a commitment) then an event $Initiates$ the fluent $p$ exactly when it $Causes$ it. However, for a base level commitment $\mathsf{C}(p)$ even though $Causes(e, \mathsf{C}(p))$, the event $e$ will not make $\mathsf{C}(p)$ true if $p$ already holds. Similarly, for $Causes(e, \mathsf{CC}(p \rightsquigarrow q))$, if $p$ holds then $e$ will create $\mathsf{C}(q)$, not $\mathsf{CC}(p \rightsquigarrow q)$, and if $q$ holds then $e$ will have no effect. The rules in Figure 7 realise these cases and Figure 8 illustrates the additional commitment discharge and creation rules. Note that the axioms of figure 7 have been implemented and can be found in the appendix.

We then have the following action effect rules for the NetBill CM (the roles, fluents and commitments remain unchanged):

$Causes(sendRequest, request)$
$Causes(sendQuote, offer)$
$Causes(sendAccept, accept)$
$Causes(sendGoods, goods)$
$Causes(sendGoods, promiseReceipt)$
$Causes(sendEPO, pay)$
$Causes(sendReceipt, receipt)$
$Terminates(sendQuote, request, t)$

We now explain how the revised axiomatisation and rules address the two commitment discharge anomalies. Let us begin with the first anomaly (Section 3.2). Consider the following sequence of steps:

1. The customer asks for a quote
2. The merchant replies with a quote. At this point the merchant has promised to send the goods if the customer accepts, and has promised to send a receipt if the customer pays.

$Implied(p, t) \leftarrow HoldsAt(p, t)$
$Implied(\mathsf{C}(x, y, p), t) \leftarrow Implied(p, t)$
$Implied(\mathsf{CC}(x, y, p, q), t) \leftarrow Implied(q, t)$
$Implied(\mathsf{CC}(x, y, p, q), t) \leftarrow Implied(\mathsf{C}(x, y, q), t)$

$Subsumes(p, p)$
$Subsumes(p, \mathsf{C}(x, y, p')) \leftarrow Subsumes(p, p')$
$Subsumes(p, \mathsf{CC}(x, y, q, p')) \leftarrow Subsumes(p, p')$
$Subsumes(\mathsf{C}(x, y, p), \mathsf{CC}(x, y, q, p')) \leftarrow Subsumes(p, p')$

$Happens(e, t) \leftarrow AgentTry(a, e, t) \wedge Precond(e, p) \wedge HoldsAt(p, t)$

$Initiates(e, p, t) \leftarrow Happens(e, t) \wedge Causes(e, p) \wedge isFluent(p)$
$Initiates(e, \mathsf{C}(x, y, p), t) \leftarrow Causes(e, \mathsf{C}(x, y, p)) \wedge Happens(e, t) \wedge \neg Implied(p, t)$
$Initiates(e, \mathsf{C}(x, y, p), t) \quad \leftarrow \quad Causes(e, \mathsf{CC}(x, y, q, p)) \wedge Happens(e, t) \wedge Implied(q, t) \wedge$
$\qquad \neg Implied(p, t)$
$Initiates(e, \mathsf{CC}(x, y, p, q), t) \leftarrow Causes(e, \mathsf{CC}(x, y, p, q)) \wedge Happens(e, t) \wedge$
$\qquad \neg Implied(q, t) \wedge \neg Implied(p, t)$
$Initiates(e, \mathsf{C}(x, y, q), t) \leftarrow HoldsAt(\mathsf{CC}(x, y, p, q), t) \wedge Happens(e, t) \wedge$
$\qquad Subsumes(p', p) \wedge Initiates(e, p', t)$
$Terminates(e, \mathsf{C}(x, y, p), t) \quad \leftarrow \quad HoldsAt(\mathsf{C}(x, y, p), t) \wedge Happens(e, t) \wedge Subsumes(p', p)$
$\qquad \wedge Initiates(e, p', t)$
$Terminates(e, \mathsf{CC}(x, y, p, q), t) \leftarrow HoldsAt(\mathsf{CC}(x, y, p, q), t) \wedge Happens(e, t) \wedge$
$\qquad Subsumes(q', q) \wedge Initiates(e, q', t)$
$Terminates(e, \mathsf{CC}(x, y, p, q), t) \leftarrow HoldsAt(\mathsf{CC}(x, y, p, q), t) \wedge Happens(e, t) \wedge$
$\qquad Subsumes(p', p) \wedge Initiates(e, p', t)$

**Fig. 7.** Revised Commitment Machine Framework

3. The customer, misunderstanding the protocol perhaps, decides to accept but sends payment instead of an acceptance.

Unlike previously, the payment causes the merchant to become committed to sending the goods (as well as a receipt). Through the postulate
$\quad Implied(\mathsf{CC}(x, y, p, q), t) \leftarrow Implied(q, t),$
the fact that the *pay* fluent holds indicates that the conditional commitment $\mathsf{CC}(goods \rightsquigarrow pay)$ *implicitly* holds[7] at the same time. This implied conditional commitment discharges the *promiseGoods* ($\mathsf{CC}(\mathsf{CC}(goods \rightsquigarrow pay) \rightsquigarrow goods)$) conditional commitment and creates the base level commitment $\mathsf{C}(goods)$. Once the commitments $\mathsf{C}(goods)$ and $\mathsf{C}(receipt)$ are discharged we are in a desirable final state.

---

[7] More precisely, it could be considered to hold: there is no actual commitment, because it has been discharged, since *pay* is true.
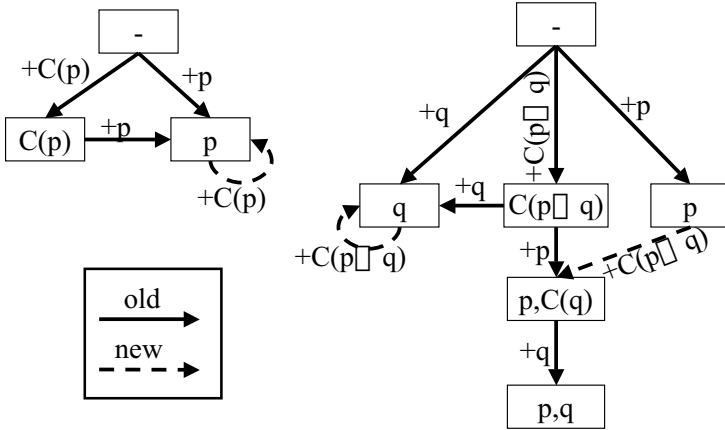
**Fig. 8.** Additional Commitment Transition Rules for base level commitment (left) and conditional commitment (right)

Consider now the second anomaly (non-symmetric commitment discharge, Section 3.3). Using the new predicate $Causes$, a conditional commitment is resolved to a base level commitment if the premise is already true using the clause

$Initiates(e, \mathsf{C}(x, y, p), t) \leftarrow$

$\qquad Causes(e, \mathsf{CC}(x, y, q, p)) \wedge Happens(e, t) \wedge Implied(q, t) \wedge \neg Implied(p, t)$

Consider the transition from state 8 to state 13, where the customer accepts after the goods have been sent. The customer's $sendAccept$ is meant to cause $accept$ ($\mathsf{CC}(goods \rightsquigarrow pay)$), but because $goods$ already holds, sending the acceptance actually creates the (base level) commitment $\mathsf{C}(pay)$.

Figure 9 shows (part of) the state machine implicitly defined by the revised Net-Bill protocol and CM axiomatisation. The differences are in states 10, 11 and 13. Whereas previously state 10 had $pay$, $\mathsf{C}(receipt)$ and $promiseGoods$, now it has $pay$, $\mathsf{C}(receipt)$ and $\mathsf{C}(goods)$. As a result state 11 now includes a commitment to send the goods and is no longer a final state. State 13, which previously had $goods$, $promiseReceipt$ and $accept$ now has $goods$, $\mathsf{C}(pay)$ and $promiseReceipt$ which is actually state 5, therefore state 13 no longer exists, and performing $sendAccept$ in state 8 leads to state 5. As before, final states are shaded. Also, dotted lines indicate actions that are affected by pre-conditions. Note that once pre-conditions are fixed (in the next sub-section) states 10, 11 and 12 will no longer be reachable. Note also that the resulting interaction space maintains the flexible interaction that is characteristic of the CM framework.

## 4.3 Issues with Pre-conditions

As discussed in Section 3.4 trying to capture pre-conditions by adding conditions to $Initiates$ clauses does not work.

Our proposed solution is to extend the agents with a proper notion of pre-condition that specifies when actions should not be performable (as opposed to preventing the
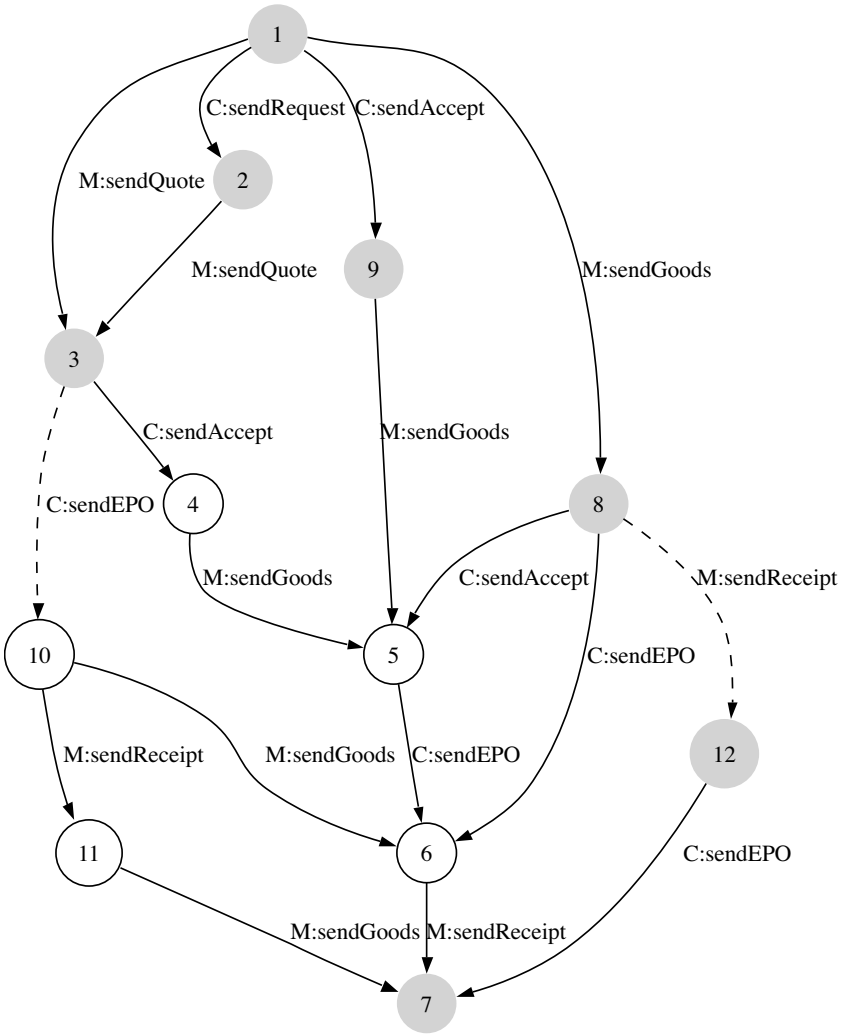
**Fig. 9.** Revised Transitions in example (partial), final states are shaded and dotted lines depict actions that are affected by pre-conditions

effects of the action from being caused). In the NetBill example we have the pre-conditions $Precond(sendEPO, goods)$ and $Precond(sendReceipt, pay)$.

We then need to de-couple an agent wanting to perform an action from the action actually occurring. This can be done by using a new predicate $AgentTry(a, e, t)$ to indicate that an agent $a$ wants to perform an action $e$ at time $t$. If the pre-conditions of the action $e$ hold[8] at time $t$ then this will imply that the event $e$ happens.

---

[8] This assumes that $p$ does not involve commitments. If it does then replace $HoldsAt(p, t)$ with $Implied(p, t)$.

$$Happens(e, t) \leftarrow AgentTry(a, e, t) \land Precond(e, p) \land HoldsAt(p, t)$$

Note that the definition of the interaction cannot prevent an agent from performing an action (any more than it can force an agent to honour its commitments). However, it can specify when an action should not be performed, and detect violations, in the same way that violations of commitments are detected.

## 5    Applications

Having proposed an improvement to the commitment machine framework, let us see how it works on another example of mutual commitment. This simple (and unrealistic) example is intended to show how the improved framework sharpens the interactions between the agents.

The example involves two roles, called "me" and "you". The two roles are friends who would like to negotiate with the outcome that both of them get an outrageous haircut for the last day of classes. It is highly undesirable that only one person have the haircut. There are two fluents (*yourscut* and *minecut*) representing who has had the haircut. We define two commitments: $Dare$ which is the conditional commitment $CC(you, me, Ok, yourscut)$, i.e., the commitment from you to me that if I agree ($Ok$) then you will get your hair cut;[9] and $Ok$, which is the conditional commitment $CC(me, you, yourscut, minecut)$, i.e., the commitment from me to you that if you get a haircut then I will get a haircut. There are four actions: *cutme* (which makes the fluent *minecut* true), *cutyou* (which makes the fluent *yourscut* true), *Offer* (which you can use to make $Dare$ true), and *Accept* (which I can use to make $Ok$ true). The actions of me cutting my hair ($cutme$) and you cutting your hair ($cutyou$) have the precondition that I (respectively you) have a commitment to do so ($C(me, you, minecut)$, respectively $C(you, me, yourscut)$).

Figure 10 shows the complete finite state machine corresponding to this CM, derived using the old axioms, whereas Figure 11 shows the complete finite state machine derived using the new axioms. It should be immediately clear that the new axioms yield a much simpler behaviour that correctly reflects the intentions of this very simple example.

Note also the symmetry of Figure 11, reflecting that the order in which the *Offer* and *Accept* messages are sent is immaterial. Hence changing $Dare$ to $CC(me, you, Ok, minecut)$ and $Ok$ to $CC(you, me, minecut, yourscut)$ will result in the same machine, except for the relabelling of me to you and vice-versa throughout.

On the other hand, using the old axioms (Figure 10), even this very simple example has anomalies. For example, consider the following sequence:

– *accept* from state 1 to state 25: $CC(me, you, yourscut, minecut)$

---

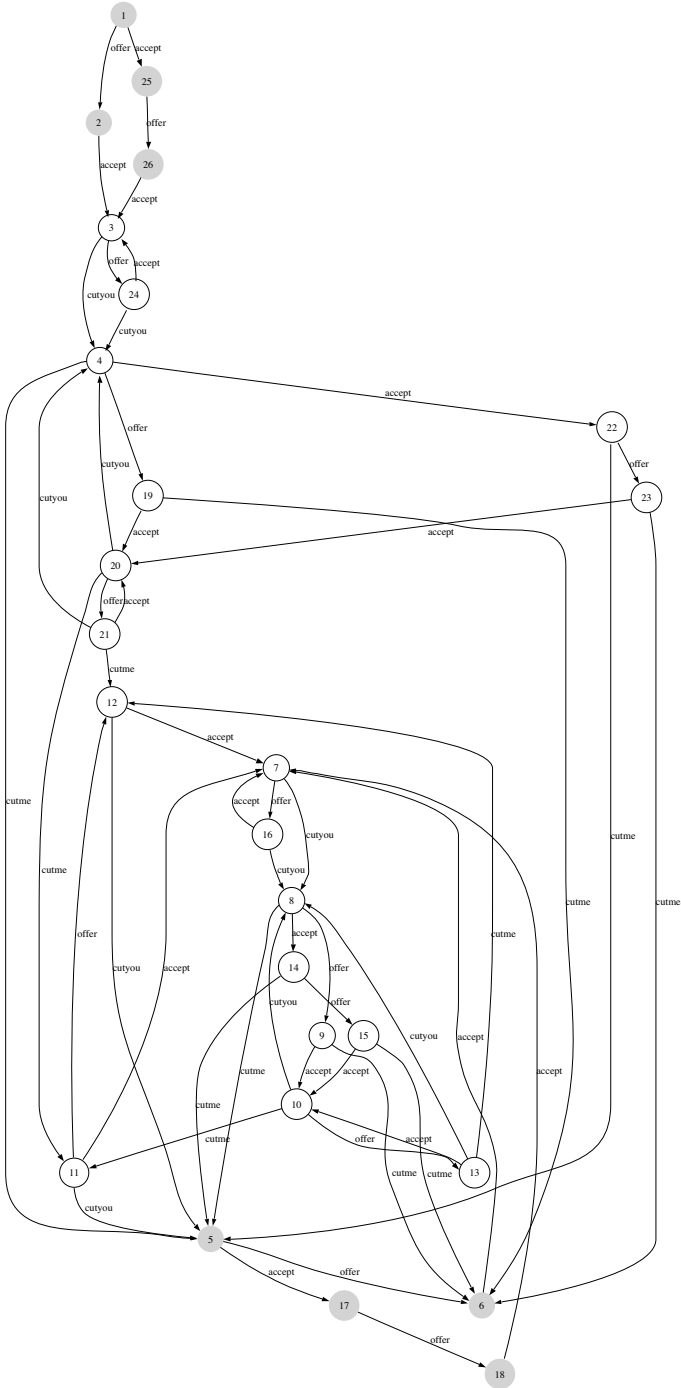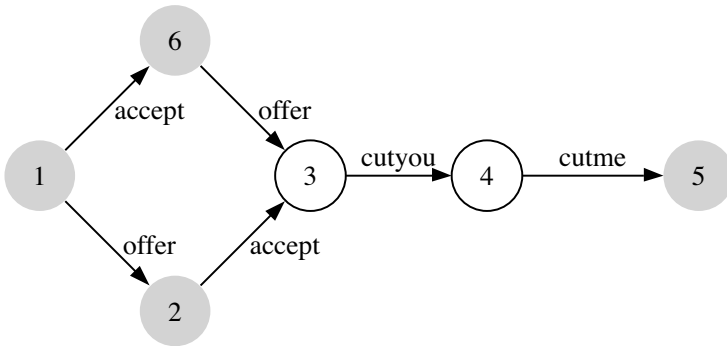[9] This is the agent equivalent of "I will if you will".

**Fig. 10.** Complete Finite State Machine for Haircut Example (old axioms)

1 -
2 $Dare = \mathsf{CC}(you, me, \mathsf{CC}(me, you, yourscut, minecut), yourscut)$
3 $\mathsf{C}(you, me, yourscut), Ok = \mathsf{CC}(me, you, yourscut, minecut)$
4 $yourscut, \mathsf{C}(me, you, minecut)$
5 $yourscut, minecut$
6 $Ok = \mathsf{CC}(me, you, yourscut, minecut)$

**Fig. 11.** Complete Finite State Machine for Haircut Example (new axioms)

– *offer* to state 26: $\mathsf{CC}(me, you, yourscut, minecut)$,
  $\mathsf{CC}(you, me, \mathsf{CC}(me, you, yourscut, minecut), yourscut)$

– *accept* (again) to state 3: $\mathsf{CC}(me, you, yourscut, minecut)$,
  $\mathsf{C}(you, me, yourscut)$

– *cutyou* to state 4: $yourscut, \mathsf{C}(me, you, minecut)$

– *accept* (again) to state 22: $yourscut, \mathsf{C}(me, you, minecut)$,
  $\mathsf{CC}(me, you, yourscut, minecut)$

– *offer* (again) to state 23: $yourscut, \mathsf{C}(me, you, minecut), \mathsf{CC}(me, you, yourscut,$
  $minecut), \mathsf{CC}(you, me, \mathsf{CC}(me, you, yourscut, minecut), yourscut)$

– *accept* (again) to state 20: $yourscut, \mathsf{C}(me, you, minecut)$,
  $\mathsf{CC}(me, you, yourscut, minecut), \mathsf{C}(you, me, yourscut)$

– *cutme* to state 11: $yourscut, \mathsf{C}(you, me, yourscut), minecut$

– *cutyou* (again) to state 5, which is a final state.

Because of anomalies with commitment creation and discharge, the action of accepting can be performed four times along this path, leading each time to a distinct state. Further, in order to fulfil the commitments, you have to cut you hair twice, which is counter-intuitive.

Note also the lack of symmetry in Figure 10. This reflects the fact that the order in which the commitments are expressed is not immaterial (i.e. that *Dare* is defined in terms of *Ok* means that the *Offer* and *Accept* actions are not entirely symmetric).

Let us now turn to the Net Bill example. Figure 12 shows the complete state machine for the interaction using the old axioms. This figure is not meant to be readable, but illustrates the size of the space and its complexity. By contrast, Figure 13 shows the same interaction, but using the new axioms[10]. The old axioms generate many more states (166 compared with 16) because of the discharge anomalies. For example, consider the following sequence of actions and resulting states:

1. sendQuote: results in the state with the two commitments *promiseGoods* and *promiseReceipt*.

   $\mathsf{CC}(accept \rightsquigarrow goods), \mathsf{CC}(pay \rightsquigarrow receipt)$

2. sendAccept: creates the commitment *accept*, which results in *promiseGoods* being transformed into the commitment to send the goods.

   $\mathsf{CC}(goods \rightsquigarrow pay), \mathsf{C}(goods), \mathsf{CC}(pay \rightsquigarrow receipt)$

3. sendQuote: creates the two commitments *promiseGoods* and *promiseReceipts*. The latter makes no difference (since it already holds), but the former is re-introduced, even though it is redundant.

   $\mathsf{CC}(goods \rightsquigarrow pay), \mathsf{C}(goods), \mathsf{CC}(pay \rightsquigarrow receipt), \mathsf{CC}(accept \rightsquigarrow goods)$

## 6    Conclusion

We analyzed the reasoning process of commitment machines and identified several anomalies in the current reasoning mechanism. We then indicated how these anomalies could be remedied, giving detailed rules for fixing the anomalies involving commitment discharge and pre-conditions.

The aim of this work is to make agent interactions more flexible and robust. There is a range of other work that has similar aims.

The work of Fornara and Colombetti [7] also uses commitments, but they use them to define the meanings of speech acts, rather than defining protocols in terms of their effects on commitments.

Kumar et al. [8] model interaction in terms of *landmarks* that need to be reached. They use the framework of joint intention theory to formalise both landmarks and speech acts. Given the complexity of the multi-modal logics used, implementation would seem to be a challenge, and no details of an implementation appear to be available. Baldoni et. al. [9] also use a multi-modal logic to formalise interaction, in their case within the DCaseLP environment using the DyLOG language.

---

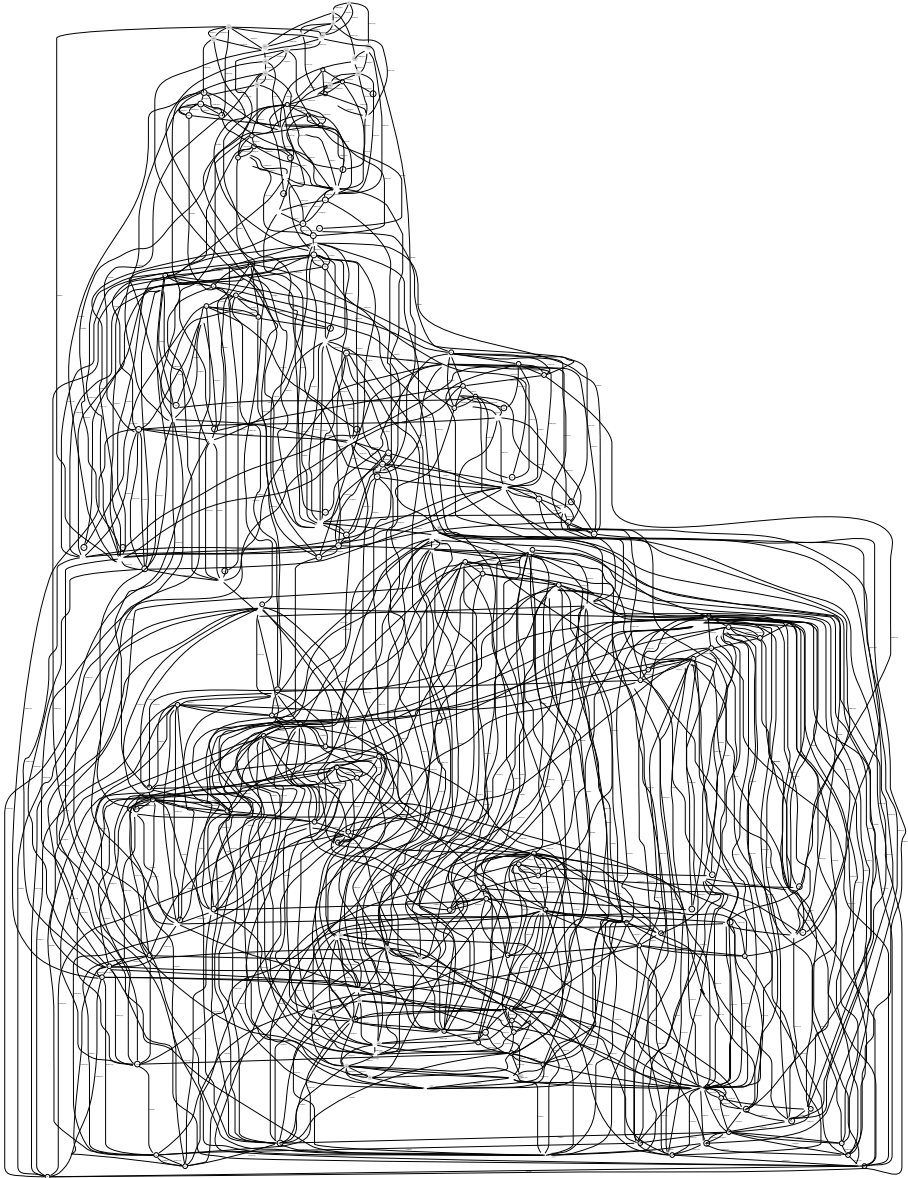[10] Note that the numbering of states has changed, since this figure was automatically generated.

**Fig. 12.** Complete Finite State Machine for Net Bill using Old Axioms. This figure is not intended to be readable, but to illustrate the size and complexity of the interaction space

Hutchison and Winikoff [10] use belief-desire-intention agents and realise inter-action using goal-triggered plans, where a given goal may have many plans that can achieve it. This work is more implementation-oriented, but is not particularly well
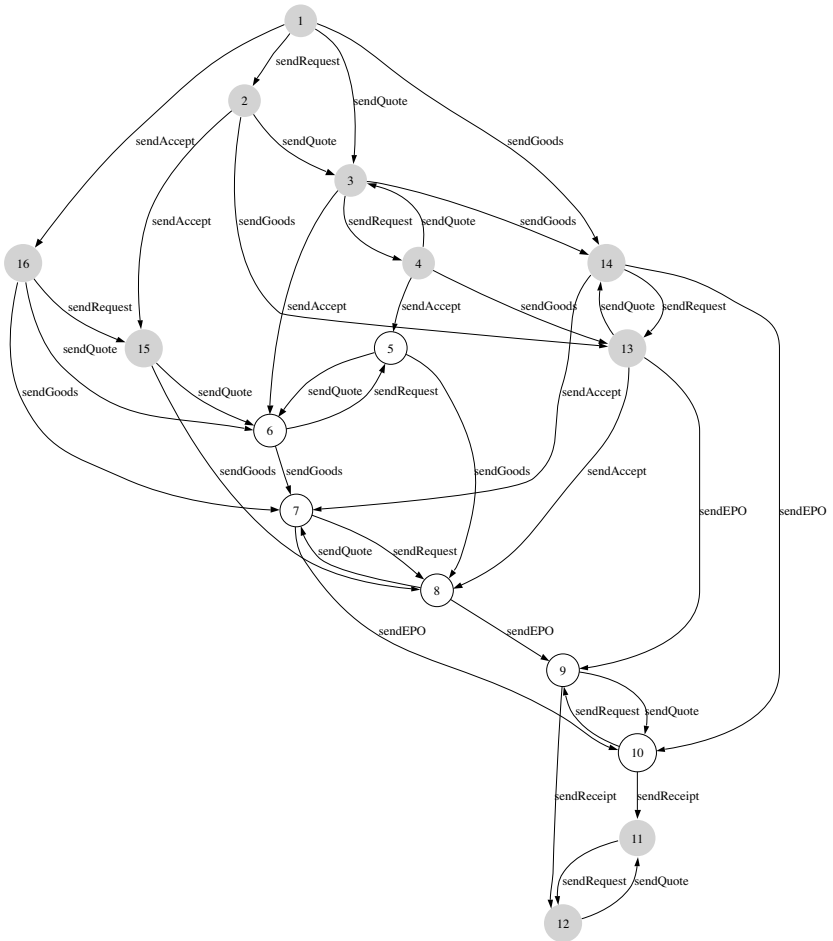
**Fig. 13.** Complete Finite State Machine for Net Bill using New Axioms, final states are shaded

developed. It provides a set of guidelines for a human to follow when designing and implementing agent interaction, rather than a formally specified interaction mechanism.

Küngas and Matskin [11] use linear logic to formalise negotiation. Their approach has been implemented, but it is not clear whether it generalises to types of interaction other than negotiation.

Chopra et. al. [12] formalise the commitment machine framework using the $\pi$ calculus. However, their formalisation is based on the axioms of [13] and suffers from the shortcomings discussed in this paper.

There are a number of papers in these proceedings that are concerned with verifying agent interactions in various ways [14, 15].

There are a number of areas for future work including extending the CM framework to deal with protocols involving open numbers of participants $(1 - N)$ such as auction protocols.

One area where we believe that commitment machines could be simplified concerns pre-conditions. In a sense pre-conditions and commitments are dual: the former state that a certain action must not be performed (under the prescribed conditions) whereas the latter state that a certain state must be brought about. It may be that the commitment machines framework could be simplified by merging the two concepts into a more generalised form of commitment. Specifically, pre-conditions could be replaced by commitments to *avoid* certain actions. These avoidance commitments, might be better termed *prohibitions*. A prohibition of the form $P(x, a)$ would state that agent $x$ is prohibited from performing action $a$. A *conditional* prohibition of the form $CP(x, a, p)$ would state that agent $x$ is *prohibited* from performing action $a$ if $p$ holds. For example, a merchant could have a conditional prohibition against sending a receipt if payment has not been made: $CP(M, sendReceipt, \neg pay)$. Prohibitions are more flexible than pre-conditions in that they can vary over time.

An additional issue in the CM framework concerns the termination of interactions. If an interaction reaches a state where there are no base-level commitments, but there are conditional commitments, then an agent, $A$, might decide that it wants to consider the interaction finished, and delete any record of it from its memory. However, after $A$ drops the interaction, agent $B$ might act in a way that changes a conditional commitment to a base-level commitment. The underlying issue is that a final state is only final in the sense that the interaction *could* end in that state, a final state does not require that the interaction *must* end there. As a result, an agent cannot consider the interaction to be completed if it could be continued.

Another area for future work would be applying our changes to the presentation of commitment machines in [13, 16]. Whereas the presentation of commitment machines in [1, 2] uses the event calculus to formalise commitment machines, the presentation of [13, 16] defines a process for compiling a commitment machine to a finite state machine.

Finally, the reasoning that each agent performs when deciding which action to do needs to be specified in more detail. The reasoning could resemble a form of game playing where an agent wants to ensure that states that it considers undesirable cannot be reached by other agents' actions while trying to achieve states that it considers desirable.

## Acknowledgements

## References

1. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: Applying event calculus planning using commitments. In: Proceedings of the 1st Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS). (2002) 527–534
2. Yolum, P., Singh, M.P.: Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. Annals of Mathematics and Artificial Intelligence (AMAI), Special Issue on Computational Logic in Multi-Agent Systems **42** (2004) 227–253

3. Reisig, W.: Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science. Springer-Verlag (1985) ISBN 0-387-13723-8.

4. Huget, M.P., Odell, J., Haugen, Ø., Nodine, M.M., Cranefield, S., Levy, R., Padgham., L.: Fipa modeling: Interaction diagrams. On http://www.auml.org under "Working Documents" (2003) FIPA Working Draft (version 2003-07-02).

5. Odell, J., Parunak, H., Bauer, B.: Extending UML for agents. In: Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence. (2000)

6. Sirbu, M.A.: Credits and debits on the internet. In Huhns, M.N., Singh, M.P., eds.: Readings in Agents. Morgan Kaufman (1998) 299–305 (Reprinted from *IEEE Spectrum*, 1997).

7. Fornara, N., Colombetti, M.: Operational specification of a commitment-based agent communication language. In: Proceeding of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, ACM Press (2002) 535 – 542

8. Kumar, S., Huber, M.J., Cohen, P.R.: Representing and executing protocols as joint actions. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, ACM Press (2002) 543 – 550

9. Baldoni, M., Baroglio, C., Gungui, I., Martelli, A., Martelli, M., Mascardi, V., Patti, V., Schifanella, C.: Reasoning about agents' interaction protocols inside DCaseLP. In *Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT'04)*, LNCS 3476, Springer-Verlag (2005). In this volume.

10. Hutchison, J., Winikoff, M.: Flexibility and robustness in agent interaction protocols. In: Workshop on Challenges in Open Agent Systems at the First International Joint Conference on Autonomous Agents and Multi-Agents Systems, Bologna, Italy, ACM Press (2002)

11. Küngas, P., Matskin, M.: Partial deduction for linear logic — the symbolic negotiation perspective. In *Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT'04)*, LNCS 3476, Springer-Verlag (2005). In this volume.

12. Chopra, A.K., Mallya, A.U., Desai, N.V., Singh, M.P.: Modeling flexible business processes. In Leite, J., Omicini, A., Torroni, P., Yolum, P., eds.: Preproceedings of Declarative Agent Languages and Technologies. (2004) 93–108

13. Yolum, P., Singh, M.: Commitment machines. In Meyer, J.J.C., Tambe, M., eds.: Agent Theories, Architectures, and Languages (ATAL). Volume 2333 of Lecture Notes in Computer Science., Springer (2002) 235–247

14. Vasconcelos, W.W.: Norm verification and analysis of electronic institutions. In *Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT'04)*, LNCS 3476, Springer-Verlag (2005). In this volume.

15. Walton, C.D.: Model checking agent dialogues. In *Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT'04)*, LNCS 3476, Springer-Verlag (2005). In this volume.

16. Yolum, P., Singh, M.: Synthesizing finite state machines for communication protocols. Technical Report TR-2001-06, North Carolina State University (2001) Available from http://www.csc.ncsu.edu/research/tech-reports/README.html.

# A    Source Code for the Implemented Axioms

The complete source code is available from http://www.cs.rmit.edu.au/~winikoff/CM

### New Axioms

```
implied(P,T) :- holdsAt(P,T).
implied(c(_,_,P),T) :- implied(P,T).
implied(cc(_,_,_,Q),T) :- implied(Q,T).
implied(cc(X,Y,_P,Q),T) :- implied(c(X,Y,Q),T).

subsumes(P,P).
subsumes(P,c(_,_,PP)) :- subsumes(P,PP).
subsumes(P,cc(_,_,_,PP)) :- subsumes(P,PP).
subsumes(c(X,Y,P),cc(X,Y,_Q,PP)) :- subsumes(P,PP).

happens(E,T) :- isAction(E), precond(E,P), holdsAt(P,T).

initiates(E,P,T) :- happens(E,T), isFluent(P), causes(E,P).
initiates(E,c(X,Y,P),T) :- causes(E,c(X,Y,P)),
  happens(E,T), \+(implied(P,T)).
initiates(E,c(X,Y,P),T) :- causes(E,cc(X,Y,Q,P)),
  happens(E,T), implied(Q,T), \+(implied(P,T)).
initiates(E,cc(X,Y,P,Q),T) :- causes(E,cc(X,Y,P,Q)),
  happens(E,T), \+(implied(Q,T)), \+(implied(P,T)).
initiates(E,c(X,Y,Q),T) :- holdsAt(cc(X,Y,P,Q),T), happens(E,T),
  subsumes(PP,P), initiates(E,PP,T).

terminates(E,c(X,Y,P),T) :- holdsAt(c(X,Y,P),T), happens(E,T),
  subsumes(PP,P), initiates(E,PP,T).
terminates(E,cc(X,Y,P,Q), T) :- holdsAt(cc(X,Y,P,Q),T),
  happens(E,T), subsumes(QP,Q), initiates(E,QP,T).
terminates(E,cc(X,Y,P,Q), T) :- holdsAt(cc(X,Y,P,Q),T),
  happens(E,T), subsumes(PP,P), initiates(E,PP,T).
```

### Old Axioms

```
initiates(E,P,T) :- happens(E,T), causes(E,P).
initiates(E,c(X,Y,Q),T) :- holdsAt(cc(X,Y,P,Q),T), happens(E,T),
  initiates(E,P,T).

terminates(E,c(X,Y,P),T) :- holdsAt(c(X,Y,P),T), happens(E,T),
  initiates(E,P,T).
terminates(E,cc(X,Y,P,Q), T) :- holdsAt(cc(X,Y,P,Q),T),
  happens(E,T), initiates(E,Q,T).
terminates(E,cc(X,Y,P,Q), T) :- holdsAt(cc(X,Y,P,Q),T),
  happens(E,T), initiates(E,P,T).
```