# Formalizing Communication Protocols for Multiagent Systems*

**Munindar P. Singh**
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
singh@ncsu.edu

## Abstract

Protocols structure interactions among communicating agents. A *commitment machine* models a protocol in terms of how the commitments of the various parties evolve. Commitment machines thus support flexible behavior while providing a meaningful basis for compliance with a protocol. Unfortunately, current formulations of commitment machines are not sufficiently general or rigorous.

This paper develops *generalized commitment machines (GCMs)* whose elements are described generically in terms of inferences, and whose computations are infinite (thus supporting nonterminating protocols). This paper shows how a GCM can be understood as a nondeterministic Büchi automaton (BA), whose acceptance condition accommodates infinite as well as finite executions.

Deterministic BA are readily emulated by conventional software, e.g., a script running in a Web browser. In general, nondeterministic BA may have no equivalent deterministic BA. However, under well-motivated technical conditions, a GCM yields a deterministic Büchi automaton that, although *not* necessarily equivalent in automata theory terms, is *sound* (produces only computations allowed by the GCM) and *complete* (produces the effect of any computation allowed by the GCM).

## 1 Introduction

Protocols streamline communication and enable the development of interoperable agents by independent vendors. The participants in a protocol should be able to exploit opportunities offered by the environment, negotiate and exploit special relationships with others, and handle exceptions. Unfortunately, conventional engineering techniques for specifying and enacting protocols are tedious and result in interactions that are unnecessarily rigid [Yolum and Singh, 2002].

Work on *commitment machines (CMs)* shows how representing and inferencing on commitments can produce flexible protocols while providing a basis for compliance [Chopra and Singh, 2004; Winikoff *et al.*, 2005; Yolum and Singh, 2002]. To simplify comparison with previous work, we use NetBill as an example protocol. In NetBill, a customer requests a quote for some goods, followed by the merchant sending the quote. If the customer accepts the quote, the merchant delivers the goods. The customer then pays. After receiving the payment, the merchant sends a receipt to the customer. Figure 1($l$) (or "left") is an FSM representation of NetBill. Figures 1($c$) ("center") and ($r$) ("right") are discussed below.

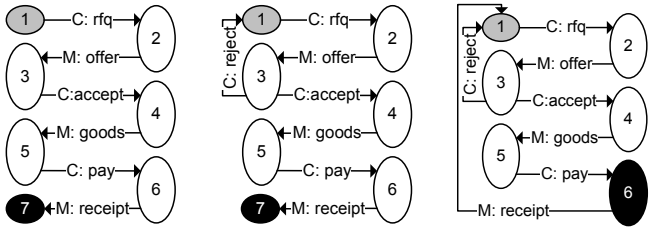


Figure 1: NetBill automata: original ($l$); terminating but unbounded ($c$); and nonterminating ($r$). States 1 are initial and 7 are final. Section 2.4 explains state 6 of ($r$). *C* and *S* refer to message senders customer and merchant, respectively

The above papers observe that the original NetBill is quite rigid. It cannot handle natural situations such as (1) a merchant proactively sending a quote, mimicking the idea of advertising a price; (2) a customer sending an "accept" message before requesting a quote (as in cases of trust or urgency); (3) a merchant sending the goods prior to the customer accepting (as in a trial offer). These papers show how to model protocols based on commitments, enabling each participant to perform a wider variety of actions yet ensure its own compliance and check others' compliance based on their actions.

Although promising, previous work on CMs is not sufficiently general, rigorous, or clear in its assumptions. Yolum and Singh model actions restrictively. Their approach only indirectly models the discharge of a commitment. Their strong point is a compilation of CMs to finite state machines. Winikoff *et al.* use the event calculus to reconstruct some of Yolum and Singh's work. They improve the methodology by which commitments are modeled, but don't study compilation as such. Chopra and Singh consider a richer, nonmonotonic theory of action based on causal logic. They don't discuss compilation except the general translation of causal logic into a transition system.

*This research was partially supported by the National Science Foundation under grant DST-0139037.

As a case in point, current and emerging financial protocols are not adequately formalized. For example, price discovery protocols, which are crucial for wholesale foreign exchange transactions, are conventionally described via scenarios and text descriptions [TWIST, 2006]. However, such descriptions are ambiguous and unwieldy. They treat each of several cases separately, losing opportunities for generalization and compactness. Our preliminary work with TWIST captures the key commitments and succinctly represents multiple scenarios using a few primitive patterns, that can be validated modularly. The value of commitments in modeling and analysis is clear; this paper strengthens their operational aspects.

Previous work assumes that a CM necessarily includes all states that are reachable through the actions modeled. However, the design of a protocol can involve considerations such as which states to include, e.g., to respect integrity and security constraints, and participants' preferences. This is especially so when subtle organizational requirements or complex roles are involved. Likewise, although nonmonotonic specifications can be valuable, monotonic representations that enumerate the modeled conditions facilitate verification of simplified, high-performance automaton-based implementations that nevertheless follow a specified protocol.

This paper introduces *generalized commitment machines (GCMs)*, which are a more general and rigorous version of CMs. It introduces a simple approach for modeling actions, so that we can clearly state the key requirements on them. The action models can be expressed in a variety of existing formalisms, including causal logic and event calculus. This paper handles the complete set of commitment operations.

Additionally, the semantics of GCMs is based on infinite computations, thus providing a proper treatment of nonterminating (and even terminating) protocols. To motivate nontermination, notice that protocols can readily have cycles. Figure 1(*c*) introduces a *reject* transition, which induces a cycle. Thus unbounded computations can be produced. However, if our model is a traditional finite automaton, each computation would terminate in a final state of the automaton. By contrast, if we automatically restart NetBill upon sending a *receipt* and interpret it as not having a final state but requiring that a *good state* (say, state 6) occurs infinitely often, then we would bring about infinite computations (Figure 1(*r*)). A good state is one in which some desirable condition, relative to design goals, is achieved. In traditional applications of automata, a good state is typically one where a stimulus has been responded to. For GCMs, a simple domain-independent basis may be that a good state is one where there are no pending commitments.

More natural nonterminating protocols involve maintaining a relationship (as in an organization) rather than completing a transaction. Participants of such protocols would continually send messages and respond as often as required. One can think of the underlying relationship as a standing agreement, analogous to those in real-life organizations or distributed systems. You don't want a multiagent system to terminate any more than you want your OS to terminate.

GCMs are specified modularly via sets of inference rules. We cleanly separate rules involving (1) generic expectations from action theories, (2) commitment operations, (3) how commitment operations are "constituted" in the setting being modeled, and (4) how a protocol should function, i.e., "regulative" rules in force. Here, (1) and (2) are fixed; (3) and (4) vary for each protocol. The above kinds of rules can be captured in a variety of existing action formalisms. Importantly, GCMs are compiled in a novel manner into *deterministic* Büchi automata, a well-known model of infinite computations [Thomas, 1990]. Automata facilitate verification; deterministic representations can be directly executed by a computer using a single thread, in constant space.

Briefly, our contributions are as follows. We seek a computational model of protocols that is natural for humans: thus it must reflect semantics. In general, producing an executable, deterministic representation can cause an exponential blow up and include unnatural states. Using semantics, we can map GCMs to deterministic automata with natural states.

## 2 Technical Framework

We introduce our action representation, and review commitments and Büchi automata. Below, $x$, $y$, $z$ are agents; $p$, $q$, etc. are formulas; and $\alpha$ is an atomic proposition or its negation: identify $\alpha$ with $\neg\neg\alpha$.

### 2.1 Generic Action Theory

The meaning of an action is specified in terms of the proposition it brings about (reflecting "causes" in causal logic or "initiates" in event calculus) under different conditions. Each action is described via an *action theory*, $\Delta$, i.e., a set of axioms that capture how the state of a protocol evolves when an action occurs. For simplicity, the performer of the action is kept implicit in the action label. Definition 1 describes action axiom schemas, which correspond to potential transitions (some of which might be allowed in a GCM).

**Definition 1** $p \overset{a}{\hookrightarrow} q$ means that $q$ is a consequence of performing action $a$ in a state where $p$ holds. ∎

If $p$ holds and $p \overset{a}{\hookrightarrow} q$, then performing $a$ yields $q$. When $q$ is $\neg p$, $a$ overturns $p$; when $q$ is $\mathsf{false}$, that means $a$ is impossible if $p$ holds. The relation $\overset{a}{\hookrightarrow}$ is closed under the inference rules below. An action theory $\Delta$ must respect these rules. Below $p \vdash q$ means that $q$ is a logical consequence of $p$.

$$\frac{p \vdash q \qquad q \overset{a}{\hookrightarrow} r \qquad r \vdash u}{p \overset{a}{\hookrightarrow} u} \quad \text{(CONSEQUENCE)}$$

$$\frac{r \overset{a}{\hookrightarrow} p \qquad r \overset{a}{\hookrightarrow} q}{r \overset{a}{\hookrightarrow} p \wedge q} \quad \text{(CONJUNCTION)}$$

The following is not an inference rule but a requirement that the $\overset{a}{\hookrightarrow}$ relation preserve consistency.

$$\frac{p \overset{a}{\hookrightarrow} q}{\neg(p \overset{a}{\hookrightarrow} \neg q)} \quad \text{(CONSISTENCY)}$$

$$\frac{p \overset{a}{\hookrightarrow} q \qquad q \not\vdash \neg\alpha}{p \wedge \alpha \overset{a}{\hookrightarrow} q \wedge \alpha} \quad \text{(INERTIA)}$$

INERTIA specifies that if $\alpha$ holds before $a$ and isn't overturned by $a$, then $\alpha$ continues to hold after $a$. INERTIA is not required here but is a convenient property of action theories.

**Example 1** Example of overturn and inertia: Consider a state $s$ where $goods \wedge \neg pay$ holds. Now if payment is made, the new state is $s'$ where $goods \wedge pay$ holds.

## 2.2 Commitments

A commitment is an explicitly manipulable directed obligation (in a context, ignored here for simplicity).

**Definition 2** A commitment $\mathsf{C}_{x,y}p$ relates a debtor $x$, a creditor $y$, and a condition $p$. Specifically, $x$ commits to $y$ to bring about $p$. ∎

**Commitment Actions**

The creation and manipulation of commitments can be described using the following operations, modeled as actions.

- $\mathsf{true} \overset{create(x,\mathsf{C}_{x,y}p)}{\hookrightarrow} \mathsf{C}_{x,y}p$: establish
- $p \wedge \mathsf{C}_{x,y}p \overset{discharge(x,\mathsf{C}_{x,y}p)}{\hookrightarrow} \neg\mathsf{C}_{x,y}p$: resolve and remove
- $\mathsf{C}_{x,y}p \overset{cancel(x,\mathsf{C}_{x,y}p)}{\hookrightarrow} \neg\mathsf{C}_{x,y}p$: remove (by debtor)
- $\mathsf{C}_{x,y}p \overset{release(y,\mathsf{C}_{x,y}p)}{\hookrightarrow} \neg\mathsf{C}_{x,y}p$: remove (by creditor)
- $\mathsf{C}_{x,y}p \overset{assign(y,z,\mathsf{C}_{x,y}p)}{\hookrightarrow} \mathsf{C}_{x,z}p \wedge \neg\mathsf{C}_{x,y}p$: change creditor
- $\mathsf{C}_{x,y}p \overset{delegate(x,z,\mathsf{C}_{x,y}p)}{\hookrightarrow} \mathsf{C}_{z,y}p \wedge \neg\mathsf{C}_{x,y}p$: change debtor

A debtor who cancels a commitment may be subject to penalties whereas a creditor may release a commitment (typically for organizational reasons) with no penalties.

**Definition 3** A conditional commitment $\mathsf{CC}_{x,y}(p,r)$ means that if antecedent $p$ becomes true, then the debtor $x$ will become committed to bringing about consequent $r$. ∎

When a conditional commitment's antecedent becomes true, it is discharged and a commitment for its consequent is created (typically, $z = y$):

- $\mathsf{CC}_{x,y}(q,r) \overset{detach(z,\mathsf{CC}_{x,y}(q,r))}{\hookrightarrow} \neg\mathsf{CC}_{x,y}(q,r) \wedge \mathsf{C}_{x,y}r$

When a conditional commitment's consequent becomes true, it is discharged and nothing else is needed (typically, $z = x$):

- $\mathsf{CC}_{x,y}(q,r) \overset{eliminate(z,\mathsf{CC}_{x,y}(q,r))}{\hookrightarrow} \neg\mathsf{CC}_{x,y}(q,r)$

**Constituting Commitment Actions**

The above rules provide the semantics of commitment actions. In each setting, the actions would be performed in some domain-specific manner. In protocols, commitment actions correspond to message types. In some cases, the messages may combine the effects of multiple commitment actions. Or, as in the case of partial returns and payments, multiple messages may be needed to implement one action. For example, paying \$3 would replace a commitment to pay \$9 by a commitment to pay \$6. The domain action theory would capture such requirements.

**Example 2** Consider the conditional commitment $\mathsf{CC}_{m,c}(pay, receipt)$ meaning that the merchant would commit to sending a receipt if the customer pays.

- The customer pays first, making *pay* true. The conditional commitment is terminated and a new commitment, $\mathsf{C}_{m,c}receipt$, is created in its stead (*detach*). When the merchant actually sends the receipt, i.e., when the proposition *receipt* becomes true, then the commitment $\mathsf{C}_{m,c}receipt$ is discharged (*discharge*).
- Before the customer pays, the merchant sends the receipt, making *receipt* true. The conditional commitment is terminated, but no other commitment is created: the customer did not commit to paying (*eliminate*).

## 2.3 Mapping to Existing Logics

The inference schemas on $\overset{a}{\hookrightarrow}$ defined above are generic and can be represented in more than one existing formalism. Hence, an interpreter for any of these formalisms can be plugged in to derive appropriate inferences. The following causal laws show how some commitment actions are captured in causal logic [Chopra and Singh, 2004, § 4.1].

1. *Discharge*: $p$ causes $\neg\mathsf{C}_{x,y}p$ if $\mathsf{C}_{x,y}p$
2. *Detach*: maps to two expressions
   $p$ causes $\neg\mathsf{CC}_{x,y}(p,r)$ if $\mathsf{CC}_{x,y}(p,r)$
   $p$ causes $\mathsf{C}_{x,y}r$ if $\mathsf{CC}_{x,y}(p,r)$
3. *Eliminate*: $r$ causes $\neg\mathsf{CC}_{x,y}(p,r)$ if $\mathsf{CC}_{x,y}(p,r)$

The following event calculus expressions capture the same actions as the above [Winikoff *et al.*, 2005, Fig. 7] (time is explicit here, but not specifically used i this formalization):

1. *Discharge*:
   $\mathsf{Terminates}(e, \mathsf{C}_{x,y}p, t) \leftarrow \mathsf{HoldsAt}(\mathsf{C}_{x,y}p, t) \wedge$
   $\mathsf{Happens}(e,t) \wedge \mathsf{Subsumes}(p', p) \wedge \mathsf{Initiates}(e, p', t)$
2. *Detach*: maps to two expressions
   $\mathsf{Initiates}(e, \mathsf{C}_{x,y}p, t) \leftarrow \mathsf{HoldsAt}(\mathsf{CC}_{x,y}(p,q), t) \wedge$
   $\mathsf{Happens}(e,t) \wedge \mathsf{Subsumes}(p', p) \wedge \mathsf{Initiates}(e, p', t)$
   $\mathsf{Terminates}(e, \mathsf{CC}_{x,y}(p,q), t) \leftarrow$
   $\mathsf{HoldsAt}(\mathsf{CC}_{x,y}(p,q), t) \wedge \mathsf{Happens}(e,t) \wedge$
   $\mathsf{Subsumes}(p', p) \wedge \mathsf{Initiates}(e, p', t)$
3. *Eliminate*: $\mathsf{Terminates}(e, \mathsf{CC}_{x,y}(p,q), t) \leftarrow$
   $\mathsf{HoldsAt}(\mathsf{CC}_{x,y}(p,q), t) \wedge \mathsf{Happens}(e,t) \wedge$
   $\mathsf{Subsumes}(q', q) \wedge \mathsf{Initiates}(e, q', t)$

We could just as easily use ECASL, the Extended Cognitive Agent Specification Language [Khan and Lespérance, 2006].

## 2.4 Büchi Automata

Finite state machines (FSMs) have long been used to specify protocols. The acceptance condition of FSMs requires termination in a final state. But, as Section 1 argues, many interesting protocols are nonterminating. To model such protocols requires that we consider automata over infinite words.

Büchi automata (BA) have finitely many states and the same transition rules as FSMs. However, their acceptance condition handles nonterminating computations by requiring that a good state (see Section 1) be visited infinitely often.

**Definition 4** A Büchi automaton is a five-tuple, $B = \langle \mathbf{S}, \mathbf{A}, s_0, \mathbf{G}, \delta \rangle$, where $\mathbf{S}$ is a set of states, $\mathbf{A}$ is the alphabet, $s_0 \in \mathbf{S}$ is the start state, $\mathbf{G} \subseteq \mathbf{S}$ is a set of good states, and $\delta \subseteq \mathbf{S} \times \mathbf{A} \times \mathbf{S}$ is the transition relation. ∎

Below, $\omega$ is the first infinite ordinal. A computation $\tau = \langle s_0, a_0, s_1, \ldots \rangle$ is an $\omega$-long alternating series of states and intervening actions. For a computation $\tau$, $\inf(\tau)$ is the set of states that occur infinitely often in $\tau$. Throughout this paper, $B$ is a BA and $\tau$ is a computation.

**Definition 5** $\tau$ is *realized* by $B$ if and only if $(\forall i \in \omega, \langle s_i, a_i, s_{i+1}\rangle \in \delta)$ and $\inf(\tau) \cap \mathbf{G} \neq \emptyset$. ∎

Figure 1(*r*) is an example BA with **S** = the set of vertices, **A** = the set of edge labels, $s_0 = 1$, **G**=$\{6\}$ shown in reverse video, and $\delta$ = the set of edges. The computation that cycles through states 1, 2, 3, 4, 5, 6, namely, $\langle 1, \mathsf{C} : \mathsf{rfq}, 2, \mathsf{M} : \mathsf{offer}, 3, \mathsf{C} : \mathsf{accept}, 4, \mathsf{M} : \mathsf{goods}, 5, \mathsf{C} : \mathsf{pay}, 6, \mathsf{M} : \mathsf{receipt}, \ldots\rangle$ is accepted by this BA, because it visits state 6 infinitely often. The computation that cycles through states 1, 2, 3 is not accepted because it does not visit state 6. Figure 2 shows BA for the languages $a(ba+b)^\omega$ and $a(a+b)^\omega$, respectively. Notice that Figure 2(*r*) accepts $a^\omega$, but Figure 2(*l*) doesn't. Thus, viewed in automata theoretic terms, the two are obviously *not equivalent*.


Figure 2: (*l*) nondeterministic BA; (*r*) deterministic BA

The move to infinite computations complicates some aspects of the automata. Importantly, whereas all nondeterministic FSMs have an equivalent deterministic FSM, some nondeterministic BA have *no* equivalent deterministic BA. Figure 2(*l*) is a famous example of a BA that cannot be determinized. A deterministic BA can be efficiently emulated by a script: if the BA is correct we know the script complies with the given protocol. This paper shows how to produce (if possible) a correct deterministic BA for a protocol.

## 3 Generalized Commitment Machines

A generalized commitment machine (GCM) is specified via states whose content is captured via logical expressions, actions and an action theory to transition between the states, and a set of good states. Terminating computations can be modeled by transitioning at their end to an artificial good state with a self-loop. Given a state, the new state reached by performing an action is logically inferred. In effect, a GCM specifies meanings of the states, and lets inferencing do the rest.

Since we construct BA from the same set of states and actions as GCMs, we use the same symbols **S**, $s_0$, **G**, and **A** in both definitions. Below, $p \equiv q$ means that $p$ and $q$ are logically equivalent, that is, $p \vdash q$ and $q \vdash p$.

**Definition 6** A GCM is a tuple $P = \langle \mathbf{S}, \mathbf{A}, s_0, \Delta, \mathbf{G}\rangle$, where **S** is a finite set of states, **A** is a finite set of actions, , $s_0 \in \mathbf{S}$ is the start state, $\Delta$ is an action theory, and $\mathbf{G} \subseteq \mathbf{S}$ is a set of good states. Members of **S** are logically distinct: $(\forall s_i, s_j \in \mathbf{S} : (s_i \equiv s_j) \Rightarrow (s_i = s_j))$; $\mathsf{false} \notin \mathbf{S}$; any state stronger than a good state is also good: $(\forall s_i \in \mathbf{G}, s_j \in \mathbf{S} : (s_j \vdash s_i) \Rightarrow (s_j \in \mathbf{G}))$. ∎

**Definition 7** $\tau$ is *generated* by a GCM $P$ if and only if $(\forall i \in \omega : s_i \in \mathbf{S}, (\exists a_i \in \mathbf{A} : s_i \overset{a_i}{\hookrightarrow} s_{i+1}))$, and $\inf(\tau) \cap \mathbf{G} \neq \emptyset$. ∎

Throughout this paper, $P$ (for "protocol") is a GCM. Given $q, r \in \mathbf{S}$, $a \in \mathbf{A}$, $P$ transitions from $q$ to $r$ under action $a$ if and only if $q \overset{a}{\hookrightarrow} r$. In effect, a GCM functions as an inference-driven (nondeterministic) BA.

**Example 3** Previous work formalizes NetBill and other terminating protocols as commitment machines. A similar approach applies here. Briefly, the modeling steps are:

- Model the atomic propositions, such as *request* or *pay*.
- Introduce the commitments needed. For example, *accept* abbreviates $\mathsf{CC}_{c,m}(goods, pay)$, i.e., if customer $c$ receives the goods, he would pay merchant $m$.
- Identify the states in **S**. For example, state 3 in Figure 1 means that the merchant has made an offer thus committing to delivering goods and sending a receipt upon payment: formally, *request* $\wedge$ *promiseReceipt* $\wedge$ *promiseGoods*.
- Identify the actions in **A**: these are the messages exchanged, e.g., *sendAccept* and *sendGoods*.
- Assert action theory axioms, $\Delta$: (1) sending an accept: $\mathsf{true} \overset{sendAccept}{\hookrightarrow} accept$; (2) delivering the goods and promising receipt: $\mathsf{true} \overset{sendGoods}{\hookrightarrow} goods \wedge promiseReceipt$.
- Identify the good states in **G**. For example, (1) the customer has made a request but no commitments have been made: *request*; (2) the merchant has delivered the goods and the receipt and the customer has paid: *request* $\wedge$ *goods* $\wedge$ *pay* $\wedge$ *receipt*.

**Example 4** A GCM analogous to Figure 2(*l*). State 1 is $p \vee q$; state 2 ($\in \mathbf{G}$) is $p \wedge q$, and $\Delta$ includes the following axioms: (1) $p \vee q \overset{a}{\hookrightarrow} p \wedge q$, (2) $p \wedge q \overset{b}{\hookrightarrow} p \wedge q$, and (3) $p \wedge q \overset{b}{\hookrightarrow} p \vee q$.

This example helps relate GCMs to the well-known BA of Figure 2(*l*) (revisited in Section 4). A scenario where such a GCM may occur is if $p$ and $q$ refer to some organizational relationships. Action $a$ strengthens the relationship from $p$ or $q$ to $p \wedge q$. Action $b$ confirms the relationship or can weaken it. (One can imagine that $b$ was originally specified for two organizational roles, now being played by the same party.)

## 4 Compiling GCMs into Büchi Automata

Reasoning with declarative representations at runtime can be expensive, may increase the code footprint of the agents. Many applications such as financial transactions [TWIST, 2006] call for flexibility when designing a protocol but rigid enactment to make performance guarantees. Also, the popularity of AJAX indicates the potential for simplified interacting agents. Compiling a GCM into a deterministic BA is thus highly desirable. A GCM is in essence a nondeterministic BA. In general, an equivalent deterministic BA may not exist. But we can exploit the semantics of the protocol to produce (if possible) a deterministic BA that for practical purposes is equivalent.

### 4.1 Determinism and Soundness

The states, initial state, and good states of a GCM map straight to those of a BA. More importantly, to ensure determinism, we can choose no more than one transition for a given state and action: this is what Algorithm 1 does.

To understand the $\min$ calculation in Algorithm 1, imagine that the GCM states are partially ordered according to logical consequence: $s_i$ is placed below $s_j$ iff $s_i \vdash s_j$. For $S \subseteq \mathbf{S}$,

$\min(S)$ is the unique minimum, if any, of $S$ according to this order. For example, assume that performing $a$ in $s_1$ could yield $s_2$ and $s_3$, where $s_2 \not\equiv s_3$. If $s_2 \vdash s_3$, Algorithm 1 would select $s_2$; if $s_3 \vdash s_2$, it would select $s_3$; if neither, Algorithm 1 would select nothing, thus *omitting* the $a$ transition from $s_1$. Theorem 1 shows that the result is deterministic.

**Theorem 1** Algorithm 1 yields a deterministic BA $B$.
**Proof.** Assume $B$ has two transitions for the same state and action, i.e., $\langle s_i, a, s_j\rangle$ and $\langle s_i, a, s_k\rangle$. Algorithm 1 ensures $s_j \vdash s_k$ and $s_k \vdash s_j$, i.e., $s_j \equiv s_k$. Definition 6 ensures $s_j = s_k$. Thus, $B$ is deterministic. ∎

1 $\delta = \{\ \}$;
2 **foreach** $s_i \in \boldsymbol{S}, a \in \boldsymbol{A}$ **do**
3 $\quad S_i = \{s_k \in \mathbf{S} : s_i \xrightarrow{a} s_k\}$;
4 $\quad$ **if** $s_j = \min(S_i)$ **then** $\delta := \delta \cup \langle s_i, a, s_j\rangle$

**Algorithm 1**: Compiling GCM $P$ into a BA $B$

**Example 5** On Example 4, Algorithm 1 would yield the BA of Figure 2(*r*) with states 1 and 2, state 2 in **G**, and transitions $\langle 1, a, 2\rangle$ (based on the axiom for $a$), $\langle 2, a, 2\rangle$ (based on applying CONSEQUENCE on the axiom for $a$), $\langle 2, b, 2\rangle$ ($\langle 2, b, 1\rangle$ is *not* included, based on line 4 of the algorithm). Thus, even though Figure 2(*l*) and Figure 2(*r*) are not equivalent, based on our knowledge of the meanings, they are practically equivalent.

Example 5 is fortunate. The compilation doesn't always work. The following develops additional steps to make compilation work, or to determine that it cannot work. Correctness has two main aspects. *Soundness* means that the compiled BA won't realize a computation that the GCM wouldn't have generated. *Completeness* means that if the GCM can generate a computation, the compiled BA can realize it. Because the BA's transitions are chosen from among those in the GCM, soundness is guaranteed. This is Theorem 2.

**Theorem 2** Let $P$ be compiled into $B$. Then any computation realized by $B$ is generated by $P$. ∎

## 4.2 Completeness via Coherence

In general, because the BA may not include all the transitions of the GCM, it would *not* be complete. First, the BA may have no transition for a state and action, because none is the (unique) strongest, as required by Algorithm 1. To address this, the basic idea is to restrict GCMs such that if there is a transition for a state and action, then there is a strongest transition for the same state and action: by exploiting the action rule CONJUNCTION. Definition 8 captures this as *coherence*. However, Figure 3(*l*) shows a situation where $a$ may occur in $p$ and produce $q_1$ and $q_2$. That is, $p \xrightarrow{a} q_1$ and $p \xrightarrow{a} q_2$ both hold. In this situation, $q_1 \wedge q_2 \notin \mathbf{S}$. That is, coherence is violated.

**Definition 8** For $s \in \mathbf{S}, a \in \mathbf{A}$, the projection of $a$ on $s$, $\pi_{a,s}$, is the set of states $s'$ in $\mathbf{S}$ such that $s \xrightarrow{a} s'$. A GCM is *coherent* if $(\forall s \in \mathbf{S}, a \in \mathbf{A} : \pi_{a,s} \neq \emptyset \Rightarrow (\exists n \in \mathbf{S} : s \xrightarrow{a} n$ and $(\forall s' \in \pi_{a,s} : n \vdash s')))$. Here, $n$ is the *supremal* state for $s$ and $a$, meaning it is the strongest resulting state. ∎

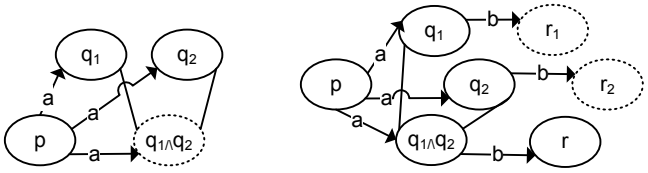


Figure 3: (*l*) Lack of coherence; (*r*) failure of coverage. Dashed ellipses represent potential states that are not in **S**

Second, a GCM computation may involve a transition that doesn't exist in the BA. To address this, the basic idea is that because of the action rule CONSEQUENCE, although a specific GCM computation may not be realized in the BA, a computation taking the strongest state at each step would be realized. Thus the BA would be complete for practical purposes. Definition 9 and Lemma 3 capture this intuition.

**Definition 9** A computation $\tau' = \langle s_0, a_0, s'_1, \ldots\rangle$ is the *semantic supremum* of a computation $\tau = \langle s_0, a_0, s_1, \ldots\rangle$ if and only if $\tau'$ begins at the same state, involves the same action sequence as $\tau$, and for each transition $\langle s_i, a_i, s_{i+1}\rangle$ exercised by $\tau$, $\tau'$ exercises a transition $\langle s'_i, a_i, s'_{i+1}\rangle$ where $s'_{i+1}$ is the strongest consequence of performing $a$ in $s'_i$. ∎

**Lemma 3** Any computation that begins from $s_0$ and is the semantic supremum of some computation in the GCM can be realized by the BA produced by Algorithm 1.
**Proof.** The proof is simply by transfinite induction on computations: consider prefixes of length from 0 to $\omega$. ∎

Theorem 4 establishes our main completeness result finessed for coherent GCMs and supremal computations.

**Theorem 4** Given a coherent GCM compiled into a BA, for any computation generated by the GCM that begins from $s_0$, there exists a semantic supremum computation realized by the BA.
**Proof.** By construction of a coherent GCM and Lemma 3. ∎

## 4.3 Producing Coherent GCMs

Let's now consider the question of whether and how an *arbitrary* GCM $P$ can be mapped into an equivalent coherent GCM $P'$. Producing a coherent GCM requires ensuring that supremal states exist for each action allowed in each state.

1 $U$ = minimal states from the above partial order of states;
2 **foreach** $u \in U, a \in \boldsymbol{A}$ **do**
3 $\quad V_a = \{v \in U : u \xrightarrow{a} v\}$;
4 $\quad$ **if** $|V_a| \geq 2$ **then**
5 $\quad\quad w = \bigwedge V_a$;
6 $\quad\quad U = U \setminus V_a \cup \{w\}$;
7 **foreach** $u \in U, a \in \boldsymbol{A}$ **do**
8 $\quad$ **if** $u \xrightarrow{a}$ true *and* $\neg(\exists s \in \boldsymbol{S} : u \vdash s$ *and* $s \xrightarrow{a}$ true$)$
$\quad$ **then** announce failure and exit;
9 $\mathbf{S}' = \mathbf{S} \cup U$;

**Algorithm 2**: Producing a coherent GCM with states $\mathbf{S}'$

Algorithm 2's first **foreach** block maintains a set of the known minimal states in the GCM $P'$ being constructed, and

examines each of them (including those newly added) to determine what new states are needed to ensure coherence. For Figure 3($l$), this would add $q_1 \wedge q_2$ to $U$ (and possibly to $\mathbf{S}'$).

Although this would yield a coherent GCM $P'$, $P'$ may generate computations that are unrelated to the computations in $P$. Consider Figure 3($r$). Here $q_1 \wedge q_2$ is added to $\mathbf{S}'$. Assume it turns out that our action theory supports $q_1 \wedge q_2 \stackrel{b}{\hookrightarrow} s$ but does not support that $b$ can occur at $q_1$ or at $q_2$ (i.e., neither $q_1 \stackrel{b}{\hookrightarrow} \mathsf{true}$ nor $q_1 \stackrel{b}{\hookrightarrow} \mathsf{true}$ hold). Thus introducing $q_1 \wedge q_2$ would generate a computation $\langle \ldots, p, a, q_1 \wedge q_2, b, \ldots \rangle$, which is *not generated* by $P$. Algorithm 2's second **foreach** block checks if the newly added states allow transitions that weren't allowed by any of the original states they entail. If so, the new GCM would generate computations not allowed by the original GCM, and so the algorithm terminates in failure. This establishes Theorem 5.

**Theorem 5** For an arbitrary GCM $P$, if Algorithm 2 succeeds, it produces a coherent GCM whose computations are generated by or are suprema of computations generated by $P$. ∎

## 5 Discussion

GCMs capture the intuitions of CMs and provide additional descriptions and expressiveness. They put the work on CMs on a sound footing and show how to incorporate them in conventional settings. GCMs can apply to a wider variety of commitment protocols, not just those relating to terminating transactions. In general, BA cannot be determinized, but taking advantage of the semantics we can compile a GCM in many cases.

Other classes of automata on infinite computations exist. For example, BA can be determinized into Müller automata. However, determinizing involves at least an exponential blow up in the number of states. More importantly, doing so yields states that are not intuitive. This paper exploits meaning to find *semantically equivalent* automata. In essence, it maps an $\omega$-language into a different $\omega$-language, but one whose computations convey the same meaning as the original.

Previous work on CMs has shown how CMs can enhance a protocol by naturally allowing a wider range of computations while ensuring compliance. In general, a GCM need not include all the states possible from a given set of states and the action theory, which would lead to large BA.

It is important to note that protocols or GCMs are merely engineering artifacts. Specifically, if Algorithm 2 ends in failure, one might consider altering the GCM so that an acceptable coherent GCM is produced. From the standpoint of engineering, the selection of states is an important decision: not all states that are possible are created equal. When states are selected, criteria such as the following ensure that the resulting GCM captures a meaningful protocol: (1) from each state where a commitment holds, we can reach a state where the commitment is discharged; (2) like above, but for other commitment operations; (3) eventually the commitment or a commitment resulting from an operation on it is discharged. We lack the space to discuss methodology further. Suffice it to say that such requirements can be readily verified via reachability analysis on the BA representation of a protocol.

Previous work on protocols either takes an operational stance or discusses semantics, but generally doesn't relate the two. Labrou and Finin [1997] describe a grammar for constructing conversations or protocols. They also give a belief and intention based semantics, but the grammar and semantics are unrelated. By contrast, GCMs provide a semantics for messages that is directly operationalized.

McBurney and Parsons' *posit spaces* protocol consists of five locutions: propose, accept, delete, suggest_revoke, and ratify_revoke [2003]. Propose and accept are similar to conditional commitments. Delete corresponds to release or discharge. Suggest_revoke and ratify_revoke enable canceling of posits. Unlike GCMs, posit spaces do not support compilation to an automata representation.

Endriss *et al.* [2003] study protocol conformance for an interaction protocol that is defined as a deterministic finite automaton (DFA). The DFA is completely specified directly, meaning that it would be rigid or unwieldy for humans to study. Endriss *et al.* do not provide formalisms to enable agents to reason about their interactions as we have done here.

## Acknowledgments

Thanks to Martin Purvis for hosting me at the University of Otago, Dunedin, New Zealand, where this paper was mostly written. Thanks also to Amit Chopra, Stephen Cranefield, and the anonymous reviewers for helpful comments.

## References

[Chopra and Singh, 2004] Amit Chopra and Munindar P. Singh. Nonmonotonic commitment machines. *AAMAS ACL Workshop*, *LNAI* 2922, pp. 183–200. Springer, 2004.

[Endriss *et al.*, 2003] Ulrich Endriss, Nicolas Maudet, Fariba Sadri, and Francesca Toni. Protocol conformance for logic-based agents. *IJCAI*, pp. 679–684, 2003.

[Khan and Lespérance, 2006] Shakil M. Khan and Yves Lespérance. On the semantics of conditional commitment. *AAMAS*, pp. 1337–1344, 2006.

[Labrou and Finin, 1997] Yannis Labrou and Tim Finin. Semantics and conversations for an agent communication language. *IJCAI*, pp. 584–591, 1997.

[McBurney and Parsons, 2003] Peter McBurney and Simon Parsons. Posit spaces: a performative model of e-commerce. *AAMAS*, pp. 624–631, 2003.

[Thomas, 1990] Wolfgang Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, volume B, pp. 133–192, North-Holland, 1990.

[TWIST, 2006] Transaction workflow innovation standards team, February 2006. http://www.twiststandards.org.

[Winikoff *et al.*, 2005] Michael Winikoff, Wei Liu, and James Harland. Enhancing commitment machines. *DALT Workshop*, *LNAI* 3476, pp. 198–220. Springer, 2005.

[Yolum and Singh, 2002] Pınar Yolum and Munindar P. Singh. Commitment machines. *ATAL Workshop*, *LNAI* 2333, pp. 235–247. Springer, 2002.