# Cupid: Commitments in Relational Algebra

**Amit K. Chopra**
Lancaster University
Lancaster LA1 4WA, United Kingdom
a.chopra1@lancaster.ac.uk

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

**Abstract**

We propose Cupid, a language for specifying commitments that supports their information-centric aspects, and offers crucial benefits. One, Cupid is first-order, enabling a systematic treatment of commitment instances. Two, Cupid supports features needed for real-world scenarios such as deadlines, nested commitments, and complex event expressions for capturing the lifecycle of commitment instances. Three, Cupid maps to relational database queries and thus provides a set-based semantics for retrieving commitment instances in states such as being violated, discharged, and so on. We prove that Cupid queries are safe. Four, to aid commitment modelers, we propose the notion of well-identified commitments, and finitely violable and finitely expirable commitments. We give syntactic restrictions for obtaining such commitments.

## 1 Introduction

(Social) commitments provide a natural basis for modeling and enacting interactions in multiagent systems (Yolum and Singh 2002a). An important application is in realizing secure collaboration among autonomous parties by providing a formal high-level model of what each party may expect from another and to what it may hold the other accountable (Young and Antón 2010). Despite advances in representing and reasoning about commitments, current approaches suffer from technical shortcomings that limit their applicability in real-world settings.

**Information-based commitments.** First, we distinguish between a schema (what occurs in a specification) and its instances (what transpires and is represented in a database). Below, we reserve the word *commitment* to mean a schema and refer to an *instance* explicitly.
Current approaches are not information-based. Most current approaches represent commitments based on propositions, thereby losing the schema-instance distinction. Existing first-order approaches deal inadequately with commitment instances: they rely upon commitment identifiers, which interferes with reasoning about commitments (Chopra and Singh 2011). For example, two commitments would appear unrelated from their conjunction because each of the three commitments would have a unique identifier. A correct approach would rely upon identifiers defined on the underlying information structures. However, there is currently no support for basing commitments on information, and naïve approaches prove inadequate.

**Expressiveness.** Effective secure collaboration presumes features such as deadlines, complex expressions, standing commitments, and nested commitments. Current approaches do not provide these in a unified framework.

**Tracking.** To interact effectively, an agent should be able to track the states of the commitments in which it is involved. Doing so is essential for accountability. For example, a resource owner would track which resources it offered to share with whom and until when. A patient would track which records a hospital should share, archive, or destroy. Current approaches provide inadequate support for tracking commitment instances.

**Modeling guidance.** First-order commitments may fail to capture what their modelers intended. For instance, variables may be inappropriately quantified, which could mean that commitment instances are impossible to discharge. Or, specification of deadlines may be such that commitment instances are never violated or they never expire, both of which would be undesirable in many settings. Current approaches provide little support to modelers in detecting and avoiding such anomalous specifications.

Cupid provides a generic application-independent solution that avoids these shortcomings. Specifically, we contribute

- Cupid, an expressive language for specifying commitments that supports identifying commitment instances.
- A semantics for Cupid using relational algebra, which enables straightforward implementation over information systems.
- Formulating safety and proving all Cupid queries are safe.
- Identifying syntactic restrictions such that Cupid commitments are well-identified. We show that commitments that are not well-identified are vacuous.
- Formulating finite violability and expiration and identifying syntactic restrictions ensuring these properties.

*Organization of this paper.* Section 2 presents some challenges in commitments with the help of examples. Section 3 presents the Cupid syntax and semantics, which Section 4 uses to address the challenges of Section 2. Section 5 formalizes properties of interest and how they may be guaran-

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

teed. Section 6 concludes with a discussion of the literature.

## 2 Challenges in Specifying Commitments

*Background.* The expression $\mathsf{C}(x, y, r, u)$ specifies a commitment from an agent $x$ to an agent $y$ that if its antecedent $r$ holds, then its consequent $u$ will hold too (Singh 2008). A commitment follows the following lifecycle. Assuming a commitment has been created, a commitment is ($a$) detached if its antecedent holds (i.e., the commitment is then unconditional); ($b$) discharged if its consequent holds; ($c$) violated if it is detached but it is not discharged and it will never be discharged; ($d$) expired if it is not detached and it will never be detached.

*Running example.* For ease of exposition, we adopt this familiar example to demonstrate our contributions:

**Ex. 1** A merchant initiates the transaction by sending a customer a *Quote*, which gives the unit price for some item. The customer responds by sending an *Order* for some number of those items. The parties then exchange the appropriate *Payment* and *Shipment*. The merchant sends a *Refund* to the customer if it fails to ship the items.

### 2.1 Expressiveness Desiderata for Cupid

We derive our desiderata for Cupid from challenges that motivate a sound treatment of information for commitments.

We consider a series of examples of commitments based on Ex. 1. To illustrate the challenges, we use a variation of Singh's (2008) notation augmented with variables and quantifiers. We treat $r$ and $u$ as expressions over events. Where the principals are not specified, we assume the commitments are from the merchant to the customer.

**Ex. 2** The merchant commits to the customer if payment occurs, then delivery will occur, that is, $\mathsf{C}(\mathsf{paid}, \mathsf{delivered})$.

Ex. 2 is not expressive: it does not say which items were committed to be delivered for what price.

**Ex. 3** $\forall price \forall item(\mathsf{C}(\mathsf{paid}(price), \mathsf{delivered}(item)))$

Ex. 3 is erroneous: once a price is paid, discharging the commitment would require delivering all possible items, which is impossible in general. Even when the domain of $item$ is finite, the commitment has an anomalous meaning. We say that such a commitment is not *safe*.

**Ex. 4** $\forall price \exists item(\mathsf{C}(\mathsf{paid}(price), \mathsf{delivered}(item)))$

Ex. 4 says that each commitment is discharged by delivering some item, but doesn't relate the item to the price paid.

**Ex. 5** $\forall price \forall item(\mathsf{C}(\mathsf{paid}(price, item), \mathsf{delivered}(item)))$

Ex. 5 is safe: both $item$ and $price$ are bound when paid occurs; delivery is expected for those items.

**Ex. 6** $\forall pID \forall price \forall item \exists dID(\mathsf{C}(\mathsf{paid}(pID, price, item), \mathsf{delivered}(dID, pID, item)))$

Ex. 6 is safe and more expressive than Ex. 5. Here $pID$ and $dID$ uniquely identify each payment and delivery. The key $pID$ enables expressing distinct commitments for the same price-item pair. Further, every delivered event *refers* to a paid event via $pID$, which identifies the payment the delivery is for: $pID$ is a foreign key in delivered. Thus, $pID$ identifies the commitment; an occurrence of delivered with that $pID$ discharges that commitment. Ex. 6 demonstrates two desirable properties: proper use of quantifiers (for safety), and use of keys to identify commitments.

A commitment may be discharged even if the antecedent has not been brought about (Yolum and Singh 2002b). For example, a merchant can discharge its commitment by delivering before payment occurs. The commitment in Ex. 6 rules out this possibility as the consequent is dependent on the antecedent for information (bindings for $pID$ and $item$). Does a commitment being safe entail that a commitment instance must not be discharged before it is detached? Such inflexibility would make commitments ill-suited for describing collaboration between autonomous parties.

Ex. 6 suffers from an additional problem. Its antecedent corresponds to the customer's payment. The customer binds the variables $price$, $item$. Then, the merchant is committed to delivering: whatever the customer wants at whatever price. This is counterintuitive and violates the autonomy of the merchant. Informally, we would expect that the merchant would supply the bindings as part of its offer, and the customer would pay accordingly to take up the offer. We should make this apparent in the commitment itself.

Ex. 7 takes care of the above problems related to autonomy. The merchant commits that for any offer of an item for a price, if the price is paid, the item will be delivered. This commitment can be discharged, that is, item can be delivered, after offer occurs but before payment occurs, because delivery does not depend on payment any more. Further, the payment must equal the price indicated in the offer.

**Ex. 7** $\forall oID \forall price \forall item \forall pID \exists dID(\mathsf{C}(\mathsf{offered}(oID, item, price) \wedge \mathsf{paid}(pID, oID, price), \mathsf{delivered}(dID, oID, item)))$

Ex. 8 models a *standing offer*: delivered depends upon payment. Every payment creates a new commitment instance to be discharged by the corresponding delivery.

**Ex. 8** $\forall oID \forall price \forall item \forall pID \exists dID(\mathsf{C}(\mathsf{offered}(oID, price, item) \wedge \mathsf{paid}(pID, oID, price), \mathsf{delivered}(dID, pID, item)))$

Ex. 9 expresses that the merchant commits to deliver the item only if the payment were at least 90% of the quoted price. Further, Ex. 9 also uses expressions for deadline: the commitment expires within ten days of making the offer and is violated if more than five days pass since payment ($oDate$ and $pDate$ are the timestamps of offered and paid).

**Ex. 9** $\forall oID \forall price \forall item \forall oDate \forall pID \forall pDate \exists dID$ $(\mathsf{C}(\mathsf{offered}(oID, price, item, oDate) \wedge$ $\mathsf{paid}(pID, oID, pPrice, pDate) \wedge \geq (pPrice, price * 0.9) \wedge$ $\mathsf{expires}(oDate + 10), \mathsf{delivered}(dID, oID, item, dDate) \wedge$ $\mathsf{violated}(pDate + 5)))$

We also need to express nested commitments, e.g., if a commitment is violated, a compensation will be provided.

## 3 The Cupid Language

We capture the antecedent and consequent of a commitment as event expressions, which proves perspicuous as we show.

Reasoning about commitments means reasoning about social events, such as their creation, discharge, violation, and so on. The social events are constructed on top of relevant lower-level events, which therefore *count as* the social events in appropriate settings (Searle 1995). The lower-level events are what are often recorded in the databases of business partners enacting a protocol with each other. For example, one party may record a message with specified contents being received, e.g., a purchase order, which may correspond to the social event of a commitment being created.

### 3.1 Syntax

Table 1 defines the syntax of Cupid. Below, $\mathcal{A}$ and $\mathcal{T}$ are the sets of agent names and time instants, respectively; in particular, $\mathcal{T} = \mathbb{N} \cup \{\infty\}$, where $\mathbb{N}$ is the set of natural numbers and $\infty$ is an infinitely distant time instant.

Table 1: Syntax of Cupid.

| | | |
|---|---|---|
| Event | $\longrightarrow$ | Base \| LifeEvent |
| LifeEvent | $\longrightarrow$ | created($\mathcal{A}$, $\mathcal{A}$, Expr, Expr, Expr) \| |
| | | detached($\mathcal{A}$, $\mathcal{A}$, Expr, Expr, Expr) \| |
| | | discharged($\mathcal{A}$, $\mathcal{A}$, Expr, Expr, Expr) \| |
| | | expired($\mathcal{A}$, $\mathcal{A}$, Expr, Expr, Expr) \| |
| | | violated($\mathcal{A}$, $\mathcal{A}$, Expr, Expr, Expr) |
| Expr | $\longrightarrow$ | Event[Time, Time] \| Expr $\sqcap$Expr \| Expr $\sqcup$Expr \| |
| | | Expr $\ominus$ Expr \| Expr where $\varphi$ |
| Time | $\longrightarrow$ | Event + $\mathcal{T}$ \| $\mathcal{T}$ |
| ComSpec | $\longrightarrow$ | commitment($\mathcal{A}$, $\mathcal{A}$, Expr, Expr, Expr) |

Conceptually, all expressions of type Expr represent composite events. This enables us to talk uniformly about the occurrence of $X \sqcap Y$, $X \sqcup Y$, and so on. Our semantics computes a timestamp for composite events. Time intervals for an event ([Time, Time] in Table 1) are interpreted strictly: the event is required to occur after (including at) the left timepoint but before the right timepoint of the interval.

When the same debtor and creditor apply uniformly in a formulation, we omit them for brevity. That is, we write $\mathsf{commitment}(c, r, u)$ instead of $\mathsf{commitment}(x, y, c, r, u)$.

### 3.2 Semantics

An *information schema* is a nonempty set of events, each modeled as a relation with a superkey and a distinguished timestamp column. Base events are materialized relations. Lifecycle events are computed via the semantics below.

A model $M$ of an information schema specifies, for each event schema $E$, an extension of that schema $[\![E]\!]$ as the set of instances of that event schema, respecting the event schema's key. The intuition behind the key is that any two event instances that agree on the key attributes must agree on every attribute (i.e., they are the same instance). Below $|$ indicates projection to the specified set of attributes.

**Definition 1** *Let $\mathcal{D} = \{\mathcal{D}_1 \ldots \mathcal{D}_n\}$ be a set of domains where $\mathcal{T} \in \mathcal{D}$ is the domain of time instants. For convenience, we identify a domain with its set of possible values.*

*An* event schema over $\mathcal{D}$ *pairs a nonempty set of* attributes *and a* key. *That is, $E = \langle A, K\rangle$, where $A \subseteq \mathcal{D}$, $\mathcal{T} \in A$, and $K \subseteq A$. (Treating each attribute as unique with its own domain simplifies the notation without loss of generality.) $\mathcal{E}_\mathcal{D}$ is the set of all possible event schemas over $\mathcal{D}$.*

*The* universe over $E$, *is the set of all possible instances of $E$. That is, if $A = \{A_1 \ldots A_m\}$, $\mathcal{U}_E = A_1 \times \ldots \times A_m$.*

*The* intension of $E$ *is the powerset of its universe restricted to sets that satisfy $E$'s key. That is, $(\!|E|\!) = \{Y | Y \in \mathcal{U}_E$ and $(\forall u_i, u_j \in Y$ if $u_i|K = u_j|K$ then $u_i = u_j)\}$.*

*An* information schema $I$ *over $\mathcal{D}$ is a nonempty set of event schemas over $\mathcal{D}$. That is, $I \subseteq \mathcal{E}_\mathcal{D}$.*

**Definition 2** *A* model *of an information schema is a function that maps each of its (Base) event schemas to its extension, i.e., a member of its intension. Specifically, $M : \mathcal{E}_\mathcal{D} \mapsto (\!|E|\!)$. We term $M(E)$ the* extension of $E$ *and write it as $[\![E]\!]_M$, omitting the subscript when $M$ is understood.*

The model defines $[\![E]\!]$ for Base $E$. The semantic postulates below lift the $[\![\ ]\!]$ to all expressions in Cupid, using relational algebra operators . Select ($\sigma$), project ($\pi$), natural join ($\bowtie$), rename ($\rho$), union ($\cup$), intersection ($\cap$), Cartesian product ($\times$), and complement ($\setminus$) retain their usual meanings (Elmasri and Navathe 1994). We reproduce definitions for some of the less common ones. Below, $Singleton_{A,B} = \{(null, \ldots, null)\}$ is the singleton null relation whose attributes are those attributes of $A$ that are not attributes of $B$.

- Left semijoin ($\ltimes$). $R \ltimes S = \pi_R(R \bowtie S)$.
- Antijoin ($\triangleright$). $R \triangleright S = R \setminus (R \ltimes S)$.
- Left outer join ($\rtimes$). $R \rtimes S = R \bowtie S \cup ((R \setminus \pi_R(R \bowtie S)) \times Singleton_{S,R})$.
- Right outer join ($\ltimes$). $R \ltimes S = S \bowtie R \cup (Singleton_{R,S} \times (S \setminus \pi_S(S \bowtie R)))$.

Below, $t$ is the distinguished timestamp attribute of all event schemas and $\{c, d\} \subseteq \mathcal{T}$. In each postulate below, $t'$ refers to a fresh (previously unused) timestamp attribute name. Below, $E, F, \ldots$ are events; $X, Y, \ldots$ are expressions.

$D_1$. $[\![E[c, d]]\!] = \sigma_{c \leqslant t < d}([\![E]\!])$. Select all events in $E$ that occur after (including at) $c$ but before $d$.

$D_2$. $[\![E[F + c, d]]\!] = \sigma_{t'+c \leqslant t < d}([\![E]\!] \ltimes \rho_{t/t'}[\![F]\!])$. Select $E$ if $F$ occurs and $E$ occurs after $c$ moments of $F$'s occurrence but before $d$.

$D_3$. $[\![E[c, F + d]]\!] = \sigma_{c \leqslant t < t'+d}([\![E]\!] \ltimes \rho_{t/t'}[\![F]\!])$. Select $E$ if $F$ occurs and $E$ occurs after $c$ but before $d$ moments of $F$'s occurrence.

$D_4$. $[\![E[F + c, G + d]]\!] = [\![E[F + c, \infty]]\!] \bowtie [\![E[0, G + d]]\!]$. Select $E$ if it occurs after $c$ moments of $F$'s occurrence and before $d$ moments of $G$'s occurrence.

$D_5$. $[\![X \sqcap Y]\!] = \sigma_{t \geq t'}([\![X]\!] \bowtie \rho_{t/t'}[\![Y]\!]) \cup \sigma_{t' < t}(\rho_{t/t'}[\![X]\!] \bowtie [\![Y]\!])$. Select $(X, Y)$ pairs where both have occurred; the timestamp of this composite event is the greater of the two.

$D_6$. $[\![X \sqcup Y]\!] = \sigma_{t \leq t' \text{ OR } t' \text{ is } null}([\![X]\!] \rtimes \rho_{t/t'}[\![Y]\!]) \cup \sigma_{t < t' \text{ OR } t' \text{ is } null}([\![X]\!] \rtimes \rho_{t/t'}[\![Y]\!])$. Select $(X, Y)$ pairs where at least one has occurred. The timestamp of this composite event is the smaller of the two, if both have occurred, or equal to the timestamp of the one that has occurred.

$D_7$. $[\![X \text{ where } \varphi]\!] = \sigma_\varphi([\![X]\!])$. Select $X$ if $\varphi$.

The interpretation of $X \ominus Y$ is that $X$ should have occurred but the (corresponding) $Y$ should not have occurred. But what is the time of nonoccurrence of an event? Consider $X \ominus E[c, d]$. Here, $E[c, d]$ (corresponding to $X$) has not occurred if $E$ (corresponding to $X$) hasn't occurred between $c$ and $d$. Thus if $E$ occurs before $c$, say at $b$, then the time of the nonoccurrence of $E[c, d]$ is $b$; if $E$ doesn't occur before $d$, then the time of nonoccurrence of $E[c, d]$ is $d$. Notice that $d$ could be $\infty$. The time of occurrence of the $X \ominus E[c, d]$ is the maximum of the timestamps of $X$ and $E[c, d]$.

$D_8$. $[\![X \ominus E[c, d]]\!] = R \cup \pi_S((\sigma_{t' \leqslant t}(\rho_{t/t'} S \times T)) \cup (\sigma_{t' < t}(S \times \rho_{t/t'} T)))$, where

- $R = \pi_X [\![X \sqcap E[0, c]]\!]$. $R$ is $X$ such that $E$ occurred too soon, that is, before $c$.
- $S = [\![X]\!] \rhd \rho_{t/t'} [\![E[0, d]]\!]$. $S$ is $X$ such that $E$ did not occur in time, that is, not before $d$.
- $T = \{(d)\}$ is a singleton relation with a single attribute $t$.

$D_9$. $[\![X \ominus E[F + c, d]]\!] = R \cup \pi_S((\sigma_{t' \leqslant t}(\rho_{t/t'} S \times T)) \cup (\sigma_{t' < t}(S \times \rho_{t/t'} T)))$, where

- $R = \pi_X [\![X \sqcap E[0, F + c]]\!]$. $R$ is $X$ such that $E$ occurred too soon, that is, before $f + c$, where $f$ is the value of $F$'s timestamp.
- $S = [\![X]\!] \rhd \rho_{t/t'} [\![E[0, d]]\!]$.
- $T = \{(d)\}$ is a singleton relation with a single attribute $t$.

The definition of $[\![X \ominus E[c, F + d]]\!]$ follows along the same lines except to account for the difference that the right timepoint refers to an event ($F$) instead of being a constant. As before, we want $X$ if $E$ occurs too soon (before $c$). We also want $X$ if $E$ occurs too late, in this case, after $f + d$, where $f$ is the value of $F$'s timestamp. We will give this nonoccurrence of $E$ the timestamp $f + d$. But what if $F$ itself hasn't occurred? Then, we won't have a value for $f$. But in this case, we would not want $X$ anyway because without the occurrence of $F$, it is not possible to determine the appropriateness of the occurrence of $E$.

$D_{10}$. $[\![X \ominus E[c, F + d]]\!] = R \cup \pi_S((\sigma_{t' \leqslant t}(\rho_{t/t'} S \bowtie T)) \cup (\sigma_{t' < t}(S \bowtie \rho_{t/t'} T)))$, where

- $R = \pi_X [\![X \sqcap E[0, c]]\!]$.
- $S = [\![X]\!] \rhd \rho_{t/t'} [\![E[0, F + d]]\!]$.
- $T$ is identical to $F$ except that each value in the timestamp column has been incremented by $d$.

$D_{11}$. $[\![X \ominus E[F + c, G + d]]\!] = R \cup \pi_S((\sigma_{t' \leqslant t}(\rho_{t/t'} S \bowtie T)) \cup (\sigma_{t' < t}(S \bowtie \rho_{t/t'} T)))$, where

- $R = \pi_X [\![X \sqcap E[0, F + c]]\!]$.
- $S = [\![X]\!] \rhd \rho_{t/t'} [\![E[0, G + d]]\!]$.
- $T$ is identical to $G$ except that each value in the timestamp column has been incremented by $d$.

$D_{12}$–$D_{14}$ reduce complex expressions involving $\ominus$.

$D_{12}$. $[\![X \ominus (Y \sqcap Z)]\!] = [\![(X \ominus Y) \sqcup (X \ominus Z)]\!]$.

$D_{13}$. $[\![X \ominus (Y \sqcup Z)]\!] = [\![(X \ominus Y) \sqcap (X \ominus Z)]\!]$.

$D_{14}$. $[\![X \ominus (Y \ominus Z)]\!] = [\![(X \ominus Y) \sqcup (X \sqcap Z)]\!]$.

$D_{15}$. $[\![created(c, r, u)]\!] = [\![c]\!]$. A commitment is created when its create event occurs.

$D_{16}$. $[\![detached(c, r, u)]\!] = [\![c \sqcap r]\!]$. A commitment is detached when its create and detach events both occur.

$D_{17}$. $[\![discharged(c, r, u)]\!] = [\![(c \sqcap u) \sqcup (r \sqcap u)]\!]$. A commitment is discharged when its discharge event has occurred along with either its create or detach event.

$D_{18}$. $[\![expired(c, r, u)]\!] = [\![c \ominus r]\!]$. A commitment is expired when its create event has occurred but its detach fails to occur within the specified interval.

$D_{19}$. $[\![violated(c, r, u)]\!] = [\![(c \sqcap r) \ominus u]\!]$. A commitment is violated when it has been created and detached but not discharged within the specified interval.

We define satisfaction with respect to a model and a time instant. That is, given a model and an instant, we restrict the extension selected by the model to events that have occurred prior to the instant, i.e., are known at that instant.

$D_{20}$. $M_{t'} \models X$ iff $\sigma_{0 \leqslant t < t'} [\![X]\!]_M$ is not empty

A surface syntax for Cupid helps improve readability. We write and, or, and except for $\sqcap$, $\sqcup$, and $\ominus$ respectively. In time intervals, we omit lower and upper instants when they are 0 and $\infty$, respectively. An omitted detach clause means the commitment is unconditional. We label commitments to simplify writing nested commitments, as in Listings 2 and 7.

## 4 Specifying Realistic Commitments

Listing 1 shows an information schema capturing the scenario of Ex. 1. Each event corresponding to a message includes participant IDs indicating its sender and receiver. As earlier, when the key of an event occurs in another event, we treat it as a foreign key in the latter.

Listing 1: Example schema identifying timestamp columns.

```
schema
 Quote(mID, cID, qID, itemID, uPrice, t)
  key qID
 Order(cID, mID, oID, qID, qty, addr, t)
  key oID
 Payment(cID, mID, pID, oID, pPrice, t)
  key pID
 Shipment(mID, cID, sID, oID, addr, t)
  key sID
 Refund(mID, cID, rID, pID, rAmount, t)
  key rID
 Coupon(cID, mID, uID, oID, rebate, t)
  key uID
```

**Ex. 10** The merchant commits to the customer via Quote that if the payment for 90% of the amount due for the quantity of items ordered by the customer occurs within ten days of the quote, then shipment will occur within five days of payment. Listing 2 is the corresponding Cupid specification.

Listing 2: A specification in Cupid surface syntax that captures the commitment in Ex. 10.

```
commitment DiscountQuote mID to cID
 create Quote
```

```
detach Order and Payment[,Quote + 10]
 where pPrice >= 0.9 * uPrice * qty
discharge Shipment[,Payment + 5]
```

**Ex. 11** Listing 2 expresses a standing commitment: the occurrence of each Order and its correlated Payment corresponds to the creation of a new commitment instance, each of which would require its own Shipment. In contrast, Listing 3, where Order is moved from detach to create, is not a standing commitment.

Listing 3: A specification in Cupid surface syntax that captures the commitment in Ex. 11.

```
commitment DiscountQuote mID to cID
 create Quote and Order
 detach Payment[,Quote + 10]
  where pPrice >= 0.9 * uPrice * qty
 discharge Shipment[,Payment + 5]
```

In contrast with Ex. 10, Ex. 12 expresses a customer's commitment. The same schema underlies both commitments. We have the choice here because Payment and Shipment do not depend upon each other. Which commitment is modeled is the modeler's choice.

**Ex. 12** Listing 4 specifies the customer's commitment to the merchant that if the ordered items are shipped within five days of order placement, then payment for at least 90% of the amount due will occur within ten days of the shipment.

Listing 4: The discounted order commitment of Ex. 12.

```
commitment DiscOrder cID to mID
create Order(,) and Quote(,)
detach Shipment(,Order+5)
discharge Payment(,Shipment + 10)
  where pPrice >= 0.9 * uPrice * qty
```

Another alternative for this schema is a nested commitment, as illustrated in Ex. 13.

**Ex. 13** The merchant's Quote creates the commitment that if the customer creates an unconditional commitment within two days of Quote to pay 90% of the Order amount within ten days of placing the Order, then the merchant will ship within five days of Payment. Listing 5 specifies the customer's unconditional commitment and Listing 6 specifies the merchant's commitment.

Listing 5: The Order commitment of Ex. 13.

```
commitment UncondPayment cID to mID
 create Order and Quote
 discharge Payment[,Order + 10]
  where pPrice >= 0.9 * uPrice * qty;
```

Listing 6: The Quote commitment of Ex. 13.

```
commitment NestedQuote mID to cID
 create Quote
 detach created(UncondPayment)[,Quote + 2]
 discharge Shipment[,Payment + 5]
```

Ex. 14 features compensation for violated commitments.

**Ex. 14** Let commitment DiscountQuote be as in Ex. 10. Listing 7 shows a compensation commitment that says that if DiscountQuote is violated, the merchant will refund 110% of the payment within nine days of the violation.

Listing 7: The compensation commitment of Ex. 14.

```
commitment Compensation mID to cID
create Quote
detach violated(DiscountQuote)
discharge Refund[,violated(DiscountQuote) + 9]
 where rAmount = 1.1 * pPrice
```

Ex. 15 illustrates the use of except.

**Ex. 15** Listing 8 is similar to Listing 3, the only difference being that the commitment is created only if the customer does not introduce a coupon in the transaction (presumably because coupons cannot be combined with other discounts, as is often the case). Listing 8 specifies this commitment.

Listing 8: A specification in Cupid surface syntax that captures the commitment in Ex. 15.

```
commitment DiscountQuote mID to cID
 create Quote and Order except Coupon
 detach Payment[,Quote + 10]
  where pPrice >= 0.9 * uPrice * qty
 discharge Shipment[,Payment + 5]
```

# 5 Desirable Properties

## 5.1 Safety

Safety is a well-known correctness criterion for database queries (Elmasri and Navathe 1994).

**Definition 3** A query $Q$ is *safe* if and only if given any possible model $M$ with finite extensions for base events, the extension of $Q$ relative to $M$, $[\![Q]\!]$, is finite.

**Theorem 1** *All Cupid queries are safe.*

*Proof sketch.* Any Cupid query maps to finitely many applications of the semantic postulates. Each postulates produces a finite output if its inputs are finite.

## 5.2 Well-Identified Commitments

Events in a commitment expression have to correlate in a certain manner. In addition, to enable a uniform treatment of events, commitment lifecycle events must have keys just the same as base events. Pragmatically, such keys would also help us refer to particular commitment instances.

We introduce the standard notion of *functional determination* from database theory (Elmasri and Navathe 1994). Let $A$ and $B$ be subsets of the attributes of a relation $R$. Then $A$ functionally determines $B$ in $R$, denoted by $A \rightarrow B$, iff each $A$ value is associated with a unique $B$ value. Definition 4 uses this notion to correlate events.

**Definition 4** *Let $E$ and $F$ be events.* $E$ determines $F$ *iff*

- *the key of $E$ functionally determines the key of $F$ in $E$, or*
- *the key of $E$ functionally determines the key of $G$ in $E$, and $G$ determines $E$.*

Definition 5 characterizes well-identified expressions in terms of the correlation among events that occur in it.

**Definition 5** *Let $X$ be an expression of type Expr in Table 1. Let $E_0, E_1, \ldots, E_n$ be the events that appear in $X$. $X$ is* well-identified *iff there exists an $E_i$ such that for all $E_j$, $E_j$ determines $E_i$. We refer to the key of $E_i$ as the key of $X$.*

*A* commitment$(X, X', X'')$ *is* well-identified *if and only if* $detached(X, X', X'')$ *determines* $created(X, X', X'')$*, and* $discharged(X, X', X'')$ *determines either* $created(X, X', X'')$ *or* $detached(X, X', X'')$*.*

*The key of* $created(X, X', X'')$ *is the key of* $X$*; that of* $detached(X, X', X'')$ *is the key of* $X'$*; that of* $discharged(X, X', X'')$ *is* $X''$*; that of* $expired(X, X', X'')$ *is* $X$*; and that of* $violated(X, X', X'')$ *is* $X'$*.*

Well-identified commitments are crucial to correlating the events involved a single commitment instance. Each of the commitments in the listings above is well-identified. For an example of a commitment that is not well-identified, consider an alternative schema where Payment does not refer to Order, that is oID is not a foreign key in Payment. Now consider the commitment of Ex. 10. There would be no way to relate any payment instance to an order instance; the detach query for this commitment would always return empty. Definition 6 characterizes such commitments as *vacuous*.

**Definition 6** *A* commitment$(X, X', X'')$ *is* vacuous *iff there is no* $M$ *such that* $M \models detached(X, X', X'')$ *and there is no* $M$ *such that* $M \models discharged(X, X', X'')$*.*

Theorem 2 gives a syntactic criterion for identifying vacuous commitments, e.g., to assist a commitment modeler.

**Theorem 2** *A commitment that is not well-identified is vacuous.*

*Proof Sketch.* Follows from $D_{16}$ and $D_{17}$.

### 5.3 Time-Oriented Properties

In general, modelers would normally prefer to write commitments that once created are either detached or discharged or are set to expire within a finite amount of time.

Definition 9 formalizes this property.

**Definition 7** *A* commitment$(X, X', X'')$ *is* finitely expirable *if and only if the following condition holds:* $\forall t \in \mathbb{N}$*, if* $M_t \models created(X, X', X'')$*, then* $\exists t' \in \mathbb{N}$ *such that* $M_{t'} \models detached(X, X', X'')$ *or* $M_{t'} \models discharged(X, X', X'')$ *or* $M_{t'} \models expired(X, X', X'')$*.*

Finite expirability of a commitment specification is not automatically guaranteed; it would depend on the specification of the time intervals in the detach clause. For instance, if the detach clause of a commitment expression is $E[0, \infty]$, then none of its instances will ever expire—$E$ could happen sometime in the future thereby detaching the instance. (Alternatively, if the detach clause is $E[0, 30]$, then if $E$ doesn't happen by 30, we can be certain it is expired.)

The situation is made more complicated if the detach clause were $E[0, F + d]$ because now the deadline for $E$ depends on when $F$ occurs. But $F$ may never occur. Consider that the create clause of this commitment may be $F[0, 10] \sqcup G[0, 10]$. Say, $G$ occurs at 8, creating the commitment. If $F$ never occurs, then the reference to $F$ in the interval of $E$ is a dangling pointer. The only way to avoid such situations is if the right timepoint of all intervals in the detach clause refers only to events that are guaranteed to have occurred once the commitment is created (we don't care about the left timepoint). The only such event is the commitment creation. Definition 8 captures this property.

**Definition 8** *A* commitment$(X, X', X'')$ *is* expirably specified *iff (1) any event that appears in the right timepoint of any interval specification in* $X'$ *is* $created(X, X', X'')$*, and (2)* $\infty$ *does not appear in the right timepoint of any interval specification in* $X'$*.*

Theorem 3 connects Definitions 7 and 8.

**Theorem 3** *A* commitment$(X, X', X'')$ *that is expirably specified is finitely expirable.*

*Proof Sketch.* From $D_{18}$, those commitments are determined expired that have been created but whose detach event is determined to have not occurred. Using $created(X, X', X'')$ and avoiding $\infty$ in the detach clause guarantees an upper time bound for the nonoccurrence of the detach.

Modelers may also want to ensure that their specifications are such that if a commitment instance is detached but not discharged within a finite amount of time, then it is violated. This motivates the analogous property of finite violability.

**Definition 9** *A* commitment$(X, X', X'')$ *is* finitely violable *if and only if the following condition holds:* $\forall t \in \mathbb{N}$*, if* $M_t \models detached(X, X', X'')$*, then* $\exists t' \in \mathbb{N}$ *such that if* $M_{t'} \models discharged(X, X', X'')$ *or* $M_{t'} \models violated(X, X', X'')$*.*

Definition 10 is analogous to Definition 8.

**Definition 10** *A* commitment$(X, X', X'')$ *is* violably specified *iff (1) any event that appears in the right timepoint of any interval specification in* $X''$ *is either* $created(X, X', X'')$ *or* $detached(X, X', X'')$*, and (2)* $\infty$ *does not appear in the right timepoint of any interval specification in* $X''$*.*

Theorem 4 connects Definitions 9 and 10.

**Theorem 4** *A* commitment$(X, X', X'')$ *that is violably specified is finitely violable.*

*Proof Sketch.* From $D_{19}$, those commitments are determined violated that have been detached but whose discharge event is determined to have not occurred. From $D_{16}$, we also know that those commitments have been created. Using $created(X, X', X'')$ or $detached(X, X', X'')$ and avoiding $\infty$ in the discharge clause guarantees an upper time bound for the nonoccurrence of the discharge.

Listing 9 shows a finitely violable and finitely expirable commitment.

Listing 9: A finitely violable and expirable commitment.

```
commitment DiscountQuote mID to cID
 create Quote
 detach Order(,created(DiscountQuote)+5) and
      Payment[,created(DiscountQuote)+10]
  where pPrice >= 0.9 * uPrice * qty
 discharge
      Shipment[,detached(DiscountQuote)+5]
```

## 6 Discussion

Broadly, our contribution consists of an elegant, expressive, event and information-centric treatment of commitments with time. We guarantee safety and show how to

achieve finite expirability and violability. Our semantics provides a compositional mapping from Cupid syntax to relational algebra queries, which are evaluated executable on commercial database products, e.g., using Yang's [2014] tool. Below, we discuss some of the key literature.

*Information in commitments.* Our treatment of information in commitments is novel. Some existing formalizations of commitment are explicitly propositional, e.g., (Singh 2008). Other work uses first-order languages such as the event calculus but has not considered information systematically (Yolum and Singh 2002b; Chesani et al. 2013). Montali et al. (2014) notice this shortcoming and present a formalization of commitment-based multiagent systems that focuses on information. Their conception of the system is a centralized one. In their conception, an "institutional" agent keeps track of all agents' commitments; commitment specifications are over the databases of multiple agents; and in the verification of system properties they consider all agents' specifications. By contrast, Cupid queries run on a single agent's database.

Montali et al. do not consider deadlines. Thus they cannot handle lifecycle events such as expiry and violation. They also do not address the identification of commitments nor a clear formulation of the safety of commitments. Further, they do not handle nested commitments. A technical difference is that Montali et al. insert commitments in databases and delete them. Therefore, it is unclear how queries about commitment lifecycle events would be supported in their approach. By contrast, we do not materialize commitments; in Cupid, commitments lifecycle events are stable and our language semantics is in terms of queries for lifecycle events. Montali et al. formalize commitment operations as well. For brevity, we do not include cancellation or release in this paper: they are straightforward in our framework.

A way of identifying commitments is by introducing a place called *commitment identifier* in the commitment relation, as in $\mathsf{C}(id, x, y, r, u)$ (Fornara and Colombetti 2002; Rovatsos 2007; Flores, Pasquier, and Chaib-Draa 2007). As discussed in Section 1, these approaches block commitment reasoning because the identifiers would get in the way. Cupid, by contrast, is compatible with commitment reasoning because in Cupid, commitment instances are identified from underlying domain information. The notions of well-identified and finitely violable and finitely expirable commitments complements earlier work on providing guidance to protocol designers (Yolum 2007).

Meneguzzi et al. (2013) specify commitments with variables to address problems in planning. However, Meneguzzi et al. don't systematically address the information-based modeling of commitments. They don't mention the concept of a (database) key. They also lack a notion of events and a notion of time. Further, Meneguzzi et al. employ a syntactic notion of commitment types and instances of a type. We distinguish commitment instances semantically, based on the events that affect the state of a commitment instance. Meneguzzi et al.'s quantification is also more limited than ours (only existentially quantified formulas in the antecedent and consequent). Further, they don't handle nested commitments. However, Meneguzzi et al.'s approach is largely complementary to ours since it concerns planning with goals rather than computing commitment lifecycle states. Of the patterns of behavior they describe, piecemeal progress, consolidation, and compensation can be handled through a suitable design of event schemas (and their keys); concession requires a more complex arrangement of commitments, which we will consider in the future.

Cupid is broadly motivated by concerns of modeling information via artifacts (e.g., Order and Payment) in business processes (Belardinelli, Lomuscio, and Patrizi 2011). A key distinction is that we model commitments on top of the artifacts, thus enabling a higher level of abstraction at which to model the business processes.

*Commitment lifecycles and time.* Cupid is event-based in the sense that commitments are affected by the occurrence of events. Each event instance can occur at most once and if it occurs it cannot be undone. Note that the effect of an event may be undone through some other event, as refund can undo payment. In Cupid, temporal intervals qualify the occurrence of an event. Our event-based approach is similar in spirit to Marengo et al.'s (2011) and Chesani et al.'s (2013). A key difference is that we treat information, including keys systematically, and formalize a commitment lifecycle via queries on databases.

Fornara and Colombetti's (2002) commitment lifecycle includes a precommitment phase. A precommitment is established when a principal requests a commitment from another. There is no analog of precommitment in our notion of commitment. However, it can be modeled in Cupid if appropriate for some application. In Cupid, there is no analog of partial violation either (Chesani et al. 2009).

Mallya et al. (2004) specify the validity conditions for time intervals that could potentially be nested. Torroni et al. (2009) and Chesani et al. (2009) show how Mallya et al.'s interval expressions can be encoded in the reactive event calculus (REC) to enable reasoning about commitment satisfaction and violation at runtime. We expect this body of work to inform work on more expressive interval specifications. Recently, Kafalı et al. (2014) propose an approach, GOSU, based on the REC that enables monitoring goals in reference to the lifecycles of commitments that support them. Cupid could help support GOSU by enabling more complex representations of commitments and through set queries.

Immediate directions of future work include (1) combining with distributed systems aspects of tracking commitments, as in Chopra and Singh (2009); (2) enhancing the language to aggregation and event-related abstractions; (3) producing a compiler that generates relational algebra from Cupid specifications (ongoing) and (4) studying the performance of Cupid queries.

## Acknowledgments

Munindar Singh was partially supported by U.S. Department of Defense under the Science of Security Lablet grant.

## References

Belardinelli, F.; Lomuscio, A.; and Patrizi, F. 2011. Verification of deployed artifact systems via data abstraction. In

*Proceedings of the Ninth International Conference on Service Oriented Computing*, number 7084 in LNCS, 142–156. Springer.

Chesani, F.; Mello, P.; Montali, M.; and Torroni, P. 2009. Commitment tracking via the reactive event calculus. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 91–96.

Chesani, F.; Mello, P.; Montali, M.; and Torroni, P. 2013. Representing and monitoring social commitments using the event calculus. *Autonomous Agents and Multi-Agent Systems* 27(1):85–130.

Chopra, A. K., and Singh, M. P. 2009. Multiagent commitment alignment. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, 937–944. IFAAMAS.

Chopra, A. K., and Singh, M. P. 2011. Specifying and applying commitment-based business patterns. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 475–482. Taipei: IFAAMAS.

Elmasri, R., and Navathe, S. 1994. *Fundamentals of Database Systems*. Redwood City, CA: Benjamin Cummings, second edition.

Flores, R. A.; Pasquier, P.; and Chaib-Draa, B. 2007. Conversational semantics sustained by commitments. *Autonomous Agents and Multi-Agent Systems* 14(2):165–186.

Fornara, N., and Colombetti, M. 2002. Operational specification of a commitment-based agent communication language. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 535–542. ACM Press.

Kafalı, Ö.; Günay, A.; and Yolum, P. 2014. GOSU: Computing goal support with commitments in multiagent systems. In *Proceedings of 21st European Conference on Artificial Intelligence*, 477–482.

Mallya, A. U.; Yolum, P.; and Singh, M. P. 2004. Resolving commitments among autonomous agents. In Dignum, F., ed., *Advances in Agent Communication, International Workshop on Agent Communication Languages, ACL 2003*, volume 2922 of *Lecture Notes in Computer Science*, 166–182. Springer.

Marengo, E.; Baldoni, M.; Baroglio, C.; Chopra, A. K.; Patti, V.; and Singh, M. P. 2011. Commitment with regulations: Reasoning about safety and control in REGULA. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 467–474.

Meneguzzi, F.; Telang, P. R.; and Singh, M. P. 2013. A first-order formalization of commitments and goals for planning. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*, 697–703. Bellevue, Washington: AAAI Press.

Montali, M.; Calvanese, D.; and Giacomo, G. D. 2014. Verification of data-aware commitment-based multiagent system. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*, 157–164. Paris: IFAAMAS.

Rovatsos, M. 2007. Dynamic semantics for agent communication languages. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 100–107. Honolulu: IFAAMAS.

Searle, J. R. 1995. *The Construction of Social Reality*. New York: Free Press.

Singh, M. P. 2008. Semantical considerations on dialectical and practical commitments. In *Proceedings of the 23rd Conference on Artificial Intelligence*, 176–181.

Torroni, P.; Chesani, F.; Mello, P.; and Montali, M. 2009. Social commitments in time: Satisfied or compensated. In *Proceedings of the 7th International Workshop on Declarative Agent Languages and Technologies (DALT)*, volume 5948 of *LNCS*, 228–243. Springer.

Yang, J. 2014. RA: A relational algebra interpreter. http://www.cs.duke.edu/~junyang/ra/. Version 2.2b.

Yolum, P., and Singh, M. P. 2002a. Commitment machines. In *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL 2001)*, volume 2333 of *LNAI*, 235–247. Seattle: Springer.

Yolum, P., and Singh, M. P. 2002b. 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*, 527–534. ACM Press.

Yolum, P. 2007. Design time analysis of multiagent protocols. *Data and Knowledge Engineering Journal* 63(1):137–154.

Young, J. D., and Antón, A. I. 2010. A method for identifying software requirements based on policy commitments. In *Proceedings of the 18th IEEE International Requirements Engineering Conference*, 47–56.