Custard: Computing Norm States over Information Stores

Amit K. Chopra
School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
a.chopra1@lancaster.ac.uk

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

ABSTRACT

Norms provide a way to model the social architecture of a sociotechnical system (STS) and are thus crucial for understanding how such a system supports secure collaboration between principals, that is, autonomous parties such as humans and organizations. Accordingly, an important challenge is to compute the state of a norm instance at runtime in a sociotechnical system.

Custard addresses this challenge by providing a relational syntax for schemas of important norm types along with their canonical lifecycles and providing a mapping from each schema to queries that compute instances of the schema in different lifecycle stages. In essence, Custard supports a norm-based abstraction layer over underlying information stores such as databases and event logs. Specifically, it supports deadlines; complex events, including those based on aggregation; and norms that reference other norms.

We prove important correctness properties for Custard, including stability (once an event has occurred, it has occurred forever) and safety (a query returns a finite set of tuples). Our compiler generates SQL queries from Custard specifications. Writing out such SQL queries by hand is tedious and error-prone even for simple norms, thus demonstrating Custard's practical benefits.

1. INTRODUCTION

A sociotechnical system (STS) involves social elements or *principals*, such as autonomous humans or organizations, and technical elements such as IT resources. We understand an agent as a software entity that acts on behalf of a principal in an STS.

Norms provide a standard of correctness for interactions among the principals, thereby capturing the social architecture of an STS [37]. Specifically, a norm captures how the principals ought to interact: it provides a social-level, yet computational, encoding of an integrity or security (for simplicity, including privacy) requirement regarding their collaboration. An example norm would be that a physician is prohibited by the hospital from disclosing identifying information about a patient. Such a norm helps characterize security at the social level independently of the implementation. Therefore, representing norms is crucial for an agent to determine how to act; and how to evaluate compliance and accountability of others.

We distinguish norm schemas from instances. A norm schema or specification describes a norm in general terms, such as a prohibition against disclosing information about a patient. A norm instance would specify the specific patient whose information has been received. Some instances of a norm may be violated and some satisfied. We consider important norm types from the literature, namely, commitment, authorization, prohibition, and power [4, 22, 26, 36]. Each of these norm types involves a canonical lifecycle [36], discussed in Figure 3, in which a norm instance may be created, expired, detached, discharged, or violated.

Norms as *institutional facts* are elements of social reality in the sense of Searle [31] and as such are realized through and reflected in *brute facts* [3], that is, low-level information. In our setting, brute facts are recorded in databases and event logs; often, these events correspond to messages sent and received. However, Searle's claims about mental representation are inapplicable here [7, 16, 30, 32].

An important challenge in realizing norms, therefore, is how to compute norm instances from brute facts. To this end, we treat (1) norms on par with information schemas and (2) database relations as stores of norm instances in various lifecycle stages. For example, an information store may indicate which instances of a prohibition are expired and which are violated. In general, we would like to specify nested norms—for example, a commitment to inform patients of the violation of a prohibition on disclosing their private information. Figure 1 illustrates our approach in conceptual terms.

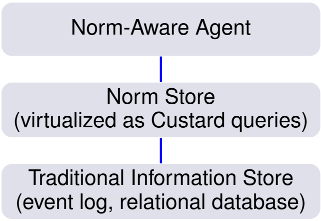

Norm-Aware Agent
Norm Store
(virtualized as Custard queries)
Traditional Information Store
(event log, relational database)

Figure 1: Custard realizes norms over information stores. An agent can query a virtual norm store for norm instances; Custard computes and retrieves such instances based on events in the underlying information store.

Our contributions are as below. First, we propose *Custard*, a language for specifying information-based norms, including *commitments*, *authorizations*, *prohibitions*, and *powers*. Custard is event-based: important stages in the lifecycle of a norm instance, specifically, its creation, detachment, expiration, discharge, and violation, are event instances and inferred from event instances recorded in the underlying information store. Custard supports complex event expressions involving logic operators, aggregation operators, relative time intervals within which events should occur, and nested norms. We give the semantics of Custard via queries in the tuple relational calculus (TRC) [15]. Effectively, for every norm specified in Custard, we define a query (expression) for each stage in the norm's canonical lifecycle, which yields all instances of the norm in that stage. The benefit of using the TRC is that it maps well to underlying representations and paves the way for easy implementation in widely used query languages such as SQL.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016),
J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.),
May 9–13, 2016, Singapore.*

Second, we formulate and prove two desirable properties for Custard. *Stability* is monotonicity over time: once an event instance has occurred, it stays occurred forever. For example, a prohibition instance violated at one moment remains violated at all future moments. Ensuring stability requires a correct treatment of time. *Safety* captures the idea that queries map to finite sets.

Third, because of Custard's support for specifying the nonoccurrence of an event, aggregation, and expressive time intervals, the TRC queries for a norm turn out to be nontrivial. As Section 4 shows, even a simple Custard specification yields SQL queries that are complex and an order of magnitude longer. Writing such queries by hand would be highly tedious, time-consuming, and error-prone. To demonstrate the practical benefits of Custard, we implemented a compiler that generates SQL queries from Custard specifications.

2. SAMPLING CUSTARD IN PRIVACY

We demonstrate the effectiveness of Custard by modeling a real-world privacy consent scenario being considered by Health Level Seven (HL7) [18], which is a leading standardization body for health information systems. A patient signs up with a cloud-based health vault provider to store and manage access to its private health information (PHI). This information may include records of the patient's vital signs such as blood pressure and blood sugar, for example, as monitored by wearable devices and uploaded to the vault. The patient may authorize third parties, such as a health coach, to receive the PHI from the vault by indicating consent. A patient may revoke an authorization. The overseeing jurisdictional authority empowers the patient to grant or revoke such authorizations. In general, third parties authorized by the patient to access information are prohibited from forwarding the information they receive to yet other parties. Parties may be sanctioned for violating this prohibition.

Listing 1 shows an information schema for this healthcare setting. It describes a number of event specifications as relations, each annotated with its key and timestamp attributes. No two instances of an event (specification) may have identical bindings for the key; for every instance, the timestamp attribute records the time of occurrence of the instance. The key of one event may occur in another. For example, accID occurs in Allowed. Such foreign keys enable correlation: every Allowed instance can be correlated by a Signedup instance via the binding for accID in the former. In general, correlations may be effected via chaining of foreign keys. For example, a Revoked instance is correlated with an Allowed instance by discID, and, therefore, with Signedup via accID.

Notice that there can be at most one SentCred instance for an Allowed instance as their keys are identical. For every disclosure to a third party, there can be zero or more requests for data from that party to the vault provider (ReqData). For every request, there can be at most one access (Accessed). A third party may forward data that it has accessed via a request to other parties zero or more times (Forwarded). Every Forwarded instance is correlated with a Signedup instance via a chain of correlations (forID to reqID to discID to accID).

We exclude methodologies for designing the appropriate information schemas from our present scope and expect such methodologies can build on known data and ontology modeling techniques.

Listing 1: Example schema for the healthcare scenario.

```
schema
 //Patient pID registers in jurisdiction jID
  Registered(pID, jID, resID, council)
 key resID time t

 //pID signs up with health vault provider hID
 Signedup(pID, hID, accID)
 key accID time t

 //pID allows disclosure to third party tpID
 Allowed(pID, hID, discID, accID, tpID, info)
 key discID time t

 //pID revokes disclosure for tpID
 Revoked(pID, hID, discID)
 key discID time t

 //hID sends creds to tpID if disclosure allowed
 SentCred(hID, tpID, discID, credentials)
 key discID time t

 //tpID requests patient data from hID
 ReqData(tpID, hID, reqID, discID, request)
 key reqID time t

 //tpID gets access to the requested data
 Accessed(tpID, hID, reqID, response)
 key reqID time t

 //tpID forwards data to party otherID
 Forwarded(tpID, otherID, forID, reqID, response)
 key forID time t

 //hID sanctions tpID for mishandling information
 Sanction(hID, tpID, discID, details)
 key discID time t
```

We build on recent work that understands norm types such as commitment, authorization, prohibition, and power as directed *social expectations* between agents [36,37]. Figure 2 shows important elements of our conceptual model.

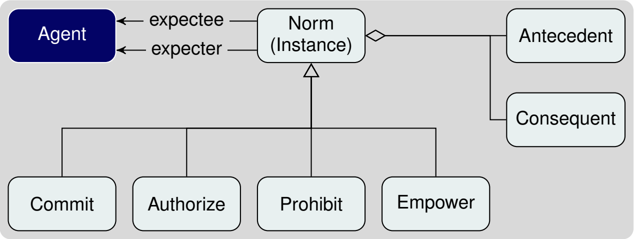
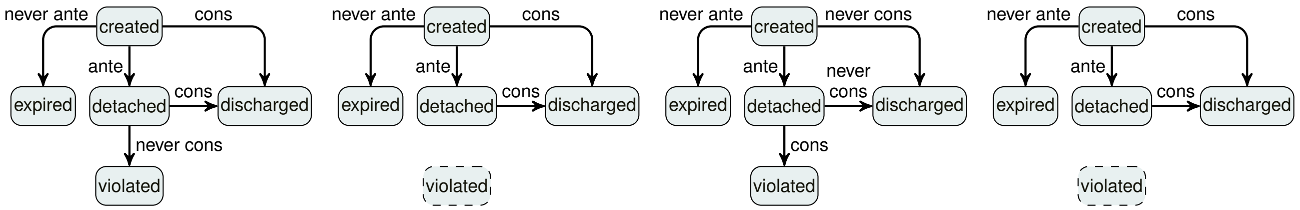


Figure 2: Simplified metamodel for norms (based on [36]).

In Figure 2, each (created) norm instance is a conditional expectation, whose *antecedent* states the condition under which the force of the norm, given by the *consequent*, applies. The expector and expectee represent the *privileged* and *liable* parties, respectively. Crucially, this formulation yields a basis for accountability in STSs: the *expectee* is accountable to the *expector* for the satisfaction of the expectation. Conversely, the expector has *standing* and may legitimately demand that the expectee give an account of the status of the expectation. As Example 1 illustrates, doing so helps understand accountability independently of implementation.

EXAMPLE 1. A commitment from a hospital to encrypt sensitive private health information (PHI) represents the patient's expectation that the hospital will do just that—and that the patient has a basis for demanding an account from the hospital about whether his or her PHI has been encrypted. A failure to encrypt PHI would be a violation, for which the hospital may be sanctioned.

Viewing norms as expectations leads to interesting questions that have not received adequate attention in the literature. For example, when is an authorization violated? And, who is accountable to whom for the violation? We now discuss possible lifecycles for the norm types, as in Figure 3, and conventions about accountability.



(a) Commitment (b) Authorization (c) Prohibition (d) Power

Figure 3: Lifecycles for the various norms. For each norm type, each box refers to a lifecycle event (not a state). The transitions refer to events as well—the occurrence or the impossibility of occurrence of the antecedent and consequent events.

2.1 Commitment

Our notion of commitment corresponds to the standard one in the literature [33]: the debtor commits to the creditor to bring about a condition (the consequent) if some condition (the antecedent) holds. We give an example from the foregoing scenario.

EXAMPLE 2. When a patient signs up, the health vault provider commits to the patient that if the patient grants access to a third party, the vault provider will authorize that party within a day.

Listing 2 shows a commitment listing in Custard. DisclosureCom is a name for the schema. The commitment is from hID to pID; it is created upon the occurrence of a Signedup instance; detached upon the occurrence of a correlated Allowed disclosure instance, provided that disclosure has not been Revoked (this allows modeling situations where patients may change their minds); it is discharged when a correlated instance of DisclosureAuth (an authorization, see Listing 3) is created. The detach and discharge clauses represent the antecedent and the consequent, respectively. In an expression of the form $E[l, r]$, $[l, r]$ is a time interval; the expression says that E should occur within this interval. To reduce clutter, we often omit l and r when they are 0 and ∞, respectively.

Listing 2: Captures Example 2.

```
commitment DisclosureCom hID to pID
 create Signedup
 detach Allowed except Revoked
 discharge created DisclosureAuth[0, Allowed+1]
```

Figure 3a shows the lifecycle of a created commitment instance. The instance is violated if the antecedent occurs but the consequent cannot; it expires if the antecedent cannot occur; and it is discharged if the consequent occurs. For simplicity, a commitment instance may be both expired and discharged: when its antecedent can no longer occur (as after a time out) but the consequent occurs.

2.2 Authorization

An agent authorizes another to bring about its consequent provided its antecedent holds.

EXAMPLE 3. The health vault provider authorizes a party by sending it credentials assuring it that patient data will be available for access any time between one and ten days after its request.

Listing 3: An authorization in Custard.

```
authorization DisclosureAuth tpID by hID
 create SentCred
 detach ReqData
 discharge Accessed[ReqData+1, ReqData+10]
```

In authorizations, we treat the authorizing party as the expectee and the authorized party as the expector. Thus, in Listing 3, the expector is tpID and the expectee is the vault. Specifically, tpID *expects* access to patient information if it has been authorized to obtain it. This agrees with the intuition that an authorization is the authorized party's privilege, not a liability. The authorization instance would be violated if the patient had authorized disclosure to a specific party, but the vault provider blocks that party from accessing the patient's information.

Although, in principle, an authorization may be violated as above, we make an architectural assumption and adopt a convention that the violation of an authorization is impossible [36]. Specifically, we assume that authorizations are regimented [20] by the agent, for example, via authentication and access control mechanisms. That is, access to a patient's record is controlled via a mechanism that allows access by all who are authorized. Figure 3b captures the authorization lifecycle. It makes the *violated* event unreachable to capture the doctrine that authorizations cannot be violated.

2.3 Prohibition

An agent prohibits another from bringing about the consequent if its antecedent has occurred. In contrast with authorizations, we apply prohibitions to cases where events are either not easily regimented or not desired to be regimented. For example, it is in general impossible to regiment a system to make disclosure of confidential information impossible. We capture the undesirability of such behaviors by placing prohibitions on them. Example 4 demonstrates a prohibition followed by its formal encoding in Listing 4.

EXAMPLE 4. The health vault provider prohibits any party who accesses patient information from forwarding that information to other parties.

Listing 4: A prohibition in Custard.

```
prohibition DisclosureProh tpID by hID
 create discharged DisclosureAuth
 violate Forwarded
```

Figure 3c captures the prohibition lifecycle. A prohibition is violated if the antecedent and the consequent have both occurred. In a prohibition, the expectee and expector are the prohibited and prohibiting parties, respectively.

2.4 Power

An agent empowers another to bring about certain states of affairs by simply "saying so" provided some conditions hold [5, 19]. We adopt Jones and Sergot's distinction between power and authorization [20]. In our setting, power is the ability of an agent to modify norms among other agents whereas authorization is the ability to access resources [36]. In addition, as discussed above, we treat authorizations as regimented via technical mechanisms.

EXAMPLE 5. The jurisdictional authority empowers the patient to authorize any party to receive the patient's information from the vault provider by simply filling out the appropriate form.

Listing 5 captures Example 5. It says that a patient who is registered in a jurisdiction and who has signed up with a vault provider after becoming a resident has the power to authorize disclosure to other parties—and to revoke such authorizations.

Listing 5: A power in Custard.

```
power ConsentPower pID by jID
 create Registered
 detach Signedup[Registered ,]
 discharge Allowed[Signedup ,] and Revoke[Signedup ,]
```

In a power instance, the empowered agent is the privileged party, that is, the expector. The empowering agent is the liable party, that is, the expectee. For simplicity, we adopt the view that a power cannot be violated for the simple reason that "saying so" under the right conditions is enough to fully exercise the power. The effects of a power may be realized through other norms. In Example 5, the effect of the power is realized through the vault provider becoming committed to authorizing parties who are allowed by the patient to receive information. Of course, the vault provider may refuse to comply, thus violating the above-mentioned commitment. If the antecedent would never not hold, then the power expires. Figure 3d captures the lifecycle of power.

2.5 Aggregation

Norms can naturally involve aggregation, as Example 6 illustrates. Such norms capture key performance indicators in business settings as well as levels of infraction by various agents.

EXAMPLE 6. The vault provider commits to the patient to declare a third party as "out of compliance" if in the year following the patient authorizing the party to receive information, it violates the prohibition to forward information to others more than twice.

Listing 6 captures Example 6. The aggregation syntax is based on the standard one in databases: specify the attribute to aggregate over (forID), the expression in which the attribute appears (violated DisclosureProh), how to group the tuples of the expression (by discID), and the attribute that holds the aggregated value (numViol). The aggregate event occurs when the count of forID over the specified interval is greater than two.

Listing 6: A norm involving aggregation in Custard.

```
commitment SanctionCom hID to pID
 create Signedup
 detach count forID of violated DisclosureProh as
      numViol group by discID > 2
      [Signedup,Signedup+365]
 discharge Sanction[,detached SanctionCom+10] where
      details="Out of compliance"
```

3. TECHNICAL FRAMEWORK

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

Expr yields complex events. The foregoing Custard listings use a surface syntax in which we (1) write and, or, and except for $\sqcap$, $\sqcup$, and $\ominus$ respectively; (2) omit lower and upper instants in time intervals when they are, respectively, 0 and ∞; (3) omit the detach clause for unconditional norms; and (4) label norms to simplify writing nested commitments. For instance, Listing 2 equals the following Spec expression: commitment(hID, pID, SignedUp[0,∞], Allowed[0,∞] $\ominus$ Revoked[0,∞], created authorization(tpID, hID, SentCred[0,∞], ReqData[0,∞], Accessed[ReqData+1, ReqData+10])[0,Allowed+1]).

Spec	$\longrightarrow$	Norm($\mathcal{A}$, $\mathcal{A}$, Expr, Expr, Expr)
Norm	$\longrightarrow$	commitment $\mid$ prohibition $\mid$ authorization $\mid$ power
Expr	$\longrightarrow$	Event[Time, Time] $\mid$ Expr where φ $\mid$
		Expr EvOp Expr
Event	$\longrightarrow$	Base $\mid$ Life $\mid$ Aggr
Life	$\longrightarrow$	created Spec $\mid$ detached Spec $\mid$ discharged Spec
		expired Spec $\mid$ violated Spec
EvOp	$\longrightarrow$	$\sqcap \mid \sqcup \mid \ominus$
Aggr	$\longrightarrow$	Func $\mathcal{D}$ of Event as $\mathcal{D}$ group by GSpec Comp $\mathbb{R}$
GSpec	$\longrightarrow$	$\mathcal{D}$ $\mid$ GSpec,$\mathcal{D}$
Time	$\longrightarrow$	Event–$\mathcal{T}$ $\mid$ Event + $\mathcal{T}$ $\mid$ $\mathcal{T}$
Func	$\longrightarrow$	sum $\mid$ count $\mid$ min $\mid$ max $\mid$ avg
Comp	$\longrightarrow$	> $\mid$ >= $\mid$ < $\mid$ <= $\mid$ = $\mid$!=

Table 1: Syntax of Custard. Spec is the start symbol.

3.1 Semantics

As Definition 1 describes, an *information schema* is a nonempty set of event schemas, each modeled as a relation with a key and a distinguished timestamp column. Informally, these correspond to Base events in our syntax (Table 1). The relation for each event schema records (positive) events. That is, no event that has occurred ever goes away.

Below, we assume that the timestamp does not feature in a key since it has no semantic force as such. $\mathcal{N}$ is a set of event names.

DEFINITION 1. *For convenience, we identify a domain with its set of possible values. (Treating each attribute as unique with its own domain simplifies the notation without loss of generality.)*

An information schema *I over $\mathcal{D}$ is a partial mapping from event names to attributes and keys; it includes precisely the events of interest. That is, $I : \mathcal{N} \to \mathcal{D} \times \mathcal{D}$. Specifically, $I(E) = \langle A, K\rangle$, where $A \subseteq \mathcal{D}$, $\mathcal{T} \in A \setminus K$, and $K \subseteq A$. For brevity, we write $E = \langle A, K\rangle$ below.*

Definition 2 gives the intension, the set of possible extensions, of an event schema with respect to a universe. A universe captures the possible combinations of attribute values under consideration.

DEFINITION 2. *Let $E = \langle\{A_1 \ldots A_m\}, K\rangle$. The* universe over *$E$, $U_E = A_1 \times \ldots \times A_m$.*

The intension *of E is the powerset of U_E restricted to sets that satisfy the key constraint of the event schema. That is, any two E instances that agree on the key attributes must agree on every attribute (that is, they are the same instance). That is, $\langle\!| E |\!\rangle = \{Y | Y \subseteq U_E$ and $(\forall u_i, u_j \in Y$: if $u_i || K = u_j || K$, then $u_i = u_j)\}$, where $||$ indicates projection to the specified set of attributes.*

An extension *of E is any member of its intension.*

Let I_E be the set of event schemas defined in an information schema I. Definition 3 states that a model of an information schema determines an extension for each of its event schemas.

DEFINITION 3. *A model M of an information schema, I, maps each of I's event schemas to its extension. Let $E = \langle A, K\rangle \in I_E$. Then the extension of E in M is any member of E's intension: that is, $[\![E]\!]^M \in \langle\!| E |\!\rangle$. (We omit M when it is understood.)*

The model defines $\llbracket Ev \rrbracket$ for Base event Ev. The semantic postulates below lift $\llbracket\ \rrbracket$ to all expressions in Custard via the TRC. In the TRC, quantification is over tuples; for a tuple τ, $\tau.a$ gives the value of attribute a of τ. Below, t is the distinguished timestamp attribute of all event schemas; $\{c, d\} \subseteq \mathcal{T}$; $E, F, \ldots$ are expressions of type Event; $X, Y, \ldots$ are expressions of type Expr; l and r are Time expressions; $\oplus$ is either '+' or '–'; $\rhd$ is any Comp operator; N is a Norm expression; g is a GSpec expression.

We use the following auxiliary definitions. The function att produces the nontimestamp attributes of an event schema; and cmn the common nontimestamp attributes of two event schemas. The predicate eq takes two tuples and a set of attributes and returns true if and only if the tuples are equal for each of those attributes; nul takes a tuple and a set of attributes and returns true if and only if each attribute's value is null in that tuple; and $holds$ takes a constraint and a tuple and returns true iff the tuple satisfies the constraint. The function $sumf$ works like the conventional sum operator in database theory: it takes four inputs, a relation S, an attribute col (in S) which needs to be summed, a set of columns g to group S by, and an attribute colsum whose value will be the sum and produces a set of tuples with the attributes g and colsum. That is, it produces a relation whose attributes are g and colsum and whose values are what it computes. We would need analogous functions $maxf$, $minf$, $countf$, and $avgf$ for tackling the corresponding constructs in Custard. The functions $maxt$ and $mint$ compute max and min of timestamps.

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

D_2. $\llbracket E[F \oplus c, d] \rrbracket = \{\tau | \exists \tau'\ \tau \in \llbracket E \rrbracket \wedge \tau' \in \llbracket F \rrbracket \wedge eq(\tau, \tau', cmn(E, F)) \wedge \tau'.t \oplus c \leqslant \tau.t < d\}$. Select E if F occurs and E occurs after (or before, depending upon what $\oplus$ is) c moments of F's occurrence but before d.

D_3. $\llbracket E[c, F \oplus d] \rrbracket = \{\tau | \exists \tau'\ \tau \in \llbracket E \rrbracket \wedge \tau' \in \llbracket F \rrbracket \wedge eq(\tau, \tau', cmn(E, F)) \wedge c \leqslant \tau.t < \tau'.t \oplus d\}$. Select E if F occurs and E occurs after c but before d moments have passed since F's occurrence (or at least d moments before F's occurrence, depending upon what $\oplus$ is).

D_4. $\llbracket E[F \oplus c, G \oplus d] \rrbracket = \{\tau | \tau \in \llbracket E[F \oplus c, \infty] \rrbracket \wedge \tau \in \llbracket E[0, G \oplus d] \rrbracket\}$. Combines D_2 and D_3.

We give definitions for aggregate events involving sum. We skip the analogous definitions for the other Func expressions (min, max, count, avg) for brevity.

D_5. $\llbracket \textsf{sum}\ col\ \textsf{of}\ E\ \textsf{as}\ colsum\ \textsf{group by}\ g \rhd n\ [l, d] \rrbracket = \{\tau | \exists \tau' \in sumf(\llbracket E[l, d] \rrbracket, col, colsum, g) \wedge \tau'.colsum \rhd n \wedge eq(\tau, \tau', g \cup \{colsum\}) \wedge \tau.t = d\}$. Compute all E instances between l and d and sum them up grouped by g over column col. If the sum for some g is $\rhd\ n$, then the event has occurred with a timestamp of d.

D_6. $\llbracket \textsf{sum}\ col\ \textsf{of}\ E\ \textsf{as}\ colsum\ \textsf{group by}\ g \rhd n\ [l, F \oplus d] \rrbracket = \{\tau | \exists \tau' \in sumf(\llbracket E[l, F \oplus d] \rrbracket, col, colsum, g) \wedge \tau'.colsum \rhd n \wedge eq(\tau, \tau', g \cup \{colsum\}) \wedge (\exists \tau'' \in \llbracket F \rrbracket \wedge eq(\tau', \tau'', att(F)) \wedge \tau.t = \tau'' \oplus d)\}$. Analogous to D_5 except that the timestamp of the sum event is relative to a corresponding F event.

D_7. $\llbracket X \sqcap Y \rrbracket = \{\tau | \exists \tau' \in \llbracket X \rrbracket \wedge (\exists \tau'' \in \llbracket Y \rrbracket \wedge eq(\tau, \tau', att(X)) \wedge eq(\tau, \tau'', att(Y)) \wedge \tau.t = maxt(\tau'.t, \tau''.t))\}$. Select ($X$, Y) pairs where both have occurred; the timestamp of this composite event is the greater of the two.

D_8. $\llbracket X \sqcup Y \rrbracket = \{\tau | (\exists \tau' \in \llbracket X \rrbracket \wedge (\exists \tau'' \in \llbracket Y \rrbracket \wedge eq(\tau, \tau', att(X)) \wedge eq(\tau, \tau'', att(Y)) \wedge \tau.t = mint(\tau'.t, \tau''.t))) \vee$

$(\exists \tau' \in \llbracket X \rrbracket \wedge eq(\tau, \tau', att(X)) \wedge \tau.t = \tau'.t \wedge nul(\tau, att(Y)) \wedge (\forall \tau'' \in \llbracket Y \rrbracket \rightarrow \neg eq(\tau, \tau'', att(Y)))) \vee$

$(\exists \tau' \in \llbracket Y \rrbracket \wedge eq(\tau, \tau', att(Y)) \wedge \tau.t = \tau'.t \wedge nul(\tau, att(X)) \wedge (\forall \tau'' \in \llbracket X \rrbracket \rightarrow \neg eq(\tau, \tau'', att(X))))\}$.

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.

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 $X \ominus E[l, d]$. Here, $E[l, d]$ (corresponding to X) has not occurred if E (corresponding to X) has not occurred between l and d. Thus if E occurs before l, say at b, then the time of the nonoccurrence of $E[l, d]$ is b; if E does not occur before d, then the time of nonoccurrence of $E[l, d]$ is d, which could be ∞. The time of occurrence of the $X \ominus E[l, d]$ is the maximum of the timestamps of X and $E[l, d]$.

D_9. $\llbracket X \ominus E[l, d] \rrbracket = \{\tau | (\exists \tau' \in \llbracket X \rrbracket \wedge eq(\tau, \tau', att(X)) \wedge \tau.t = maxt(\tau'.t, d) \wedge (\forall \tau'' \in \llbracket E[0, d] \rrbracket \rightarrow \neg eq(\tau', \tau'', cmn(X, E)))) \vee$

$(\exists \tau' \in \llbracket X \rrbracket \wedge eq(\tau, \tau', att(X)) \wedge (\exists \tau'' \in \llbracket E[0, l] \rrbracket \wedge eq(\tau', \tau'', cmn(X, E)) \wedge \tau.t = maxt(\tau'.t, \tau''.t)))\}$.

The definition of $\llbracket X \ominus E[c, F \oplus d] \rrbracket$ 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). In addition, we want X if E occurs too late, in this case, after $f \oplus d$, where f is the value of F's timestamp. We will give this nonoccurrence of E the timestamp $f \oplus d$. But what if F itself has not occurred? Then, we would not 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}. $\llbracket X \ominus E[l, F \oplus d] \rrbracket = \{\tau | (\exists \tau' \in \llbracket X \rrbracket \wedge (\exists \tau'' \in \llbracket F \rrbracket \wedge eq(\tau, \tau', att(X)) \wedge eq(\tau', \tau'', cmn(X, F)) \wedge \tau.t = maxt(\tau'.t, \tau''.t \oplus d) \wedge (\forall \tau''' \in \llbracket E[0, F \oplus d] \rrbracket \rightarrow \neg eq(\tau, \tau''', cmn(X, E))))) \vee$

$(\exists \tau' \in \llbracket X \rrbracket \wedge eq(\tau, \tau', att(X)) \wedge (\exists \tau'' \in \llbracket E[0, l] \rrbracket \wedge eq(\tau', \tau'', cmn(X, E)) \wedge \tau.t = maxt(\tau'.t, \tau''.t)))\}$.

D_{11}. $\llbracket X\ \textsf{where}\ \varphi \rrbracket = \{\tau | \tau \in \llbracket X \rrbracket \wedge holds(\tau, \varphi)\}$.

D_{12}–D_{15} transform complex expressions involving $\ominus$ to those where $\ominus$ has an event of the form $E[l, r]$ as its right hand side operand.

D_{12}. $\llbracket X \ominus (Y \sqcap Z) \rrbracket = \llbracket (X \ominus Y) \sqcup (X \ominus Z) \rrbracket$.

D_{13}. $\llbracket X \ominus (Y \sqcup Z) \rrbracket = \llbracket (X \ominus Y) \sqcap (X \ominus Z) \rrbracket$.

D_{14}. $\llbracket X \ominus (Y \ominus Z) \rrbracket = \llbracket (X \ominus Y) \sqcup (X \sqcap Z) \rrbracket$.

D_{15}. $\llbracket X \ominus (Y\ \textsf{where}\ \varphi) \rrbracket = \llbracket (X \ominus Y) \sqcup T \rrbracket$, where $T = \{\tau | \exists \tau' \in \llbracket X \sqcap Y\ \textsf{where}\ \neg\varphi \rrbracket \wedge eq(\tau, \tau', att(X)) \wedge \tau.t = \tau'.t\}$.

Below, we write $N(c, r, u)$ where the definition applies uniformly to all kinds of norms. For brevity, we omit the expectee and expecter agents.

D_{16}. $\llbracket \mathsf{created}\ N(c,r,u) \rrbracket = \llbracket c \rrbracket$. An norm instance is created when its create event occurs.

D_{17}. $\llbracket \mathsf{detached}\ N(c,r,u) \rrbracket = \llbracket c \sqcap r \rrbracket$. A norm instance is detached when its create and detach events both occur.

D_{18}. $\llbracket \mathsf{expired}\ N(c,r,u) \rrbracket = \llbracket c \ominus r \rrbracket$. A norm instance is expired when its create event has occurred but its detach fails to occur within the specified interval.

D_{19}. $\llbracket \mathsf{discharged\ commitment}(c,r,u) \rrbracket = \llbracket (c \sqcap u) \sqcup (r \sqcap u) \rrbracket$. A commitment is discharged when its discharge event has occurred along with either its create or detach event.

D_{20}. $\llbracket \mathsf{discharged\ authorization}(c,r,u) \rrbracket = \llbracket c \sqcap r \sqcap u \rrbracket$. An authorization is discharged when its discharge event has occurred along with its create and detach event.

D_{21}. $\llbracket \mathsf{discharged\ power}(c,r,u) \rrbracket = \llbracket c \sqcap r \sqcap u \rrbracket$. A power is discharged when its discharge event has occurred along with its create and detach event.

D_{22}. $\llbracket \mathsf{discharged\ prohibition}(c,r,u) \rrbracket = \llbracket (c \sqcap r) \ominus u \rrbracket$. A prohibition is discharged when its create and detach events occur but the violate event fails to occur.

D_{23}. $\llbracket \mathsf{violated\ commitment}(c,r,u) \rrbracket = \llbracket (c \sqcap r) \ominus u \rrbracket$. A commitment is violated when its create and detach events occur but the discharge event fails to occur.

D_{24}. $\llbracket \mathsf{violated\ authorization}(c,r,u) \rrbracket = \emptyset$ (the empty set). No authorization can be violated.

D_{25}. $\llbracket \mathsf{violated\ power}(c,r,u) \rrbracket = \emptyset$. No power can be violated.

D_{26}. $\llbracket \mathsf{violated\ prohibition}(c,r,u) \rrbracket = \llbracket c \sqcap r \sqcap u \rrbracket$. A prohibition is violated when its create, detach, and violate events all occur.

3.2 Properties

Stability is the idea that once an event is determined to have occurred, then at all future time instants, it should continue to be determined to have occurred. In other words, an event that has occurred cannot later unoccur. For example, a message that been sent cannot be unsent. Stability of events is a fundamental assumption in reasoning about distributed systems.

We would like to extend this notion of stability to norm lifecycle event instances. Thus, for example, if a prohibition instance is determined to have been violated at a time instant, then at all future instants, it should be determined violated. Stability would be highly desirable in business settings as it would give stakeholders confidence in the status of things.

A Base event is by definition stable, since the model defines its extension. However, stability for complex events, including lifecycle events, does not automatically follow from the stability of Base events. Two features, in particular, demand careful consideration.

First, let us consider $\ominus$, the except operator. Imagine an event specification of the form $X \ominus E[0,100]$. Let us say an expression is evaluated at time 50, before which some X instance has occurred but the corresponding E instance has not occurred. Then one may assume that the corresponding $X \ominus E[0,100]$ instance has occurred. However, doing so would be premature: the E instance could yet occur, say at time 55, and a later query would determine the $X \ominus E[0,100]$ instance to have not occurred. In essence, we would have switched the status of the event from occurred to not occurred.

Second, let us consider aggregation operators. A sum event determined to have occurred at an instant may at future instants be determined to have not occurred as additional events occur. For example, if the sum over a number of events was required to be greater than some value, it may hit that value after observing, say, five events. However, future events may lower the sum (if the attribute which is being summed can take negative values) and cause it to dip below the required value.

We have built the semantics so that stability is guaranteed for all events and the above-described scenarios do not occur. We can treat all observations up to a time as forming a model. Thus as a computation progresses and additional events occur, a model corresponding to additional observations would extend a prior model.

DEFINITION 4. *Let L and M be two models for information schema I. Then M* expands on *L if and only if for each event in I_E, $\llbracket E \rrbracket^L \subseteq \llbracket E \rrbracket^M$.*

Theorem 1 states that the result obtained by evaluating an expression would persist through expansions of models, thereby ensuring stability of Custard.

THEOREM 1. *Let L and M be models where M expands on L. Let X be any Expr expression. Then, $\llbracket X \rrbracket^L \subseteq \llbracket X \rrbracket^M$.*

Proof Sketch. The proof is by induction on the syntax. Specifically, an expression X maps to a tree of height h where the leaf nodes are Base events and the root is the expression itself.

The expressions at the leaves represent the base case for the induction. The model defines $\llbracket Ev \rrbracket$ if Ev is a Base event. We obtain $\llbracket Ev \rrbracket^L \subseteq \llbracket Ev \rrbracket^M$ immediately from our assumption regarding Base events. Consider an expression at height k $(0 < k \leqslant h)$. Assume that the property holds for its children. The expression must correspond to one of the postulates in D_1–D_{26}. We must show that the stability property holds for them.

D_1. From the inductive hypothesis, we know that $\llbracket E \rrbracket^L \subseteq \llbracket E \rrbracket^M$. It follows from D_1 that $\llbracket E[c,d] \rrbracket^L \subseteq \llbracket E[c,d] \rrbracket^M$. Reasoning for D_2–D_4 is analogous.

D_5. If $\tau \in \llbracket \mathsf{sum}\ col\ \mathsf{of}\ E\ \mathsf{as}\ colsum\ \mathsf{group\ by}\ \gamma \rhd n[l,d] \rrbracket^L$, then from the fact that $\tau.t = d$, we know all relevant E instances, that is, those that happen in $\llbracket E[l,d] \rrbracket$ have been considered. Further, $\llbracket E[l,d] \rrbracket^L = \llbracket E[l,d] \rrbracket^M$. Therefore, stability holds. D_6 is analogous.

D_7. Follows from $\llbracket X \rrbracket^L \subseteq \llbracket X \rrbracket^M$ and $\llbracket Y \rrbracket^L \subseteq \llbracket Y \rrbracket^M$. D_8 is analogous.

D_9. Two subcases. One, $\tau \in \llbracket X \ominus E[l,d] \rrbracket^L$ and τ occurs at some time k but E does not occur in $[0,d]$. In this case, stability of the sum event follows from the fact that $\llbracket E[0,d] \rrbracket^L = \llbracket E[0,d] \rrbracket^M$ and $\llbracket X \rrbracket^L \subseteq \llbracket X \rrbracket^M$. Two, $\tau \in \llbracket X \ominus E[l,d] \rrbracket^L$ because X and $E[0,l]$ have both occurred. In this case, stability of the sum event follows from the fact that $\llbracket E[0,l] \rrbracket^L = \llbracket E[0,l] \rrbracket^M$ and $\llbracket X \rrbracket^L \subseteq \llbracket X \rrbracket^M$. D_{10} is analogous.

D_{11}. Follows from the fact that $\llbracket X \rrbracket$ is stable.

D_{12}–D_{15}. An expression of these forms reduces to an expression where the right hand side of every $\ominus$ operand is a base event, lifecycle event, or aggregation event qualified by a time interval. Such expressions are stable. Their combinations with other expressions is also stable.

D_{16} Follows from the fact that $\llbracket c \rrbracket$ is stable. D_{17}–D_{26} are analogous.

Safety is a well-known correctness criterion for database queries [15]. Definition 6 describes safety as the idea that the result of any query evaluated over a finite database is finite. Here, a query is an expression that we evaluate using a database.

DEFINITION 5. *Let M be a model over information schema I. Then M is a* finite model *if and only if for each event E occurring in I_E, $\llbracket E \rrbracket^M$, is finite.*

DEFINITION 6. *Let Q be an Expr expression. Then, Q is* safe *if and only if given any finite model M of I, the extension of Q relative to M, $\llbracket Q \rrbracket^M$, is finite.*

Negation-like operators such as $\ominus$ have the potential to compromise safety if their usage is not restricted adequately. For example, imagine that we had a unary negation operator $\ominus_u$ and the create clause for some commitment were simply $\ominus_u E$ (assume E is Base). This would amount to considering created infinitely many commitment instances, one for each E instance that is *not* present in $\llbracket E \rrbracket$. A technique that is commonly employed to avoid such conclusions is to guard such negation-like operators, as we do in Custard: $\ominus$ is a binary operator, the extension of whose left operand circumscribes the extension of its right operand. Theorem 2 and its proof sketch below capture the foregoing discussion.

THEOREM 2. *Let Q be any Expr expression in Custard. Then Q is safe.*

Proof sketch. The proof is by induction on the height of the syntax tree. The expressions at the leaves represent the base case. We know that they have finite extensions because a finite model defines finite extensions for Base events. Assume finiteness for every expression at height k and show finiteness for every expression at height $k+1$. For brevity, we illustrate only the crucial cases, which involve $\ominus$, as motivated above. Suppose an expression at $k+1$ is $X \ominus E[l, d]$. By the inductive hypothesis, we know that both X and $E[l, d]$ have finite extensions. According to the definition of $\llbracket X \ominus E[l, d] \rrbracket$ (D_9), there are two subcases to consider, corresponding to the disjunction. In both cases, though, we are selecting tuples from finite extensions of X and E (specifically, from $E[0, d]$ and $E[0, l]$). Hence, $\llbracket X \ominus E[l, d] \rrbracket$ is finite. The other cases involving $\ominus$ are analogous.

4. IMPLEMENTATION

We implemented a Custard compiler in Java using the Eclipse XText language definition and parsing library (version 2.8.3). The compiler reads in one or norm schemas along with an information schema, such as the one in Listing 1, and outputs (1) SQL table creation statements corresponding to the information schema and (2) SQL queries, one for each lifecycle event for each specified norm schema. We adopt the widely used MySQL dialect of SQL. Listing 7 shows some of the table creation statements generated for the information schema in Listing 1.

Listing 7: Generated SQL Create Table statements.

```
CREATE TABLE SentCred (
  hID  VARCHAR(10), tpID  VARCHAR(10), discID
      VARCHAR(10), credentials  VARCHAR(10),
  t  DATETIME,
  PRIMARY KEY(discID)
);

CREATE TABLE ReqData (
  hID  VARCHAR(10), tpID  VARCHAR(10), discID
      VARCHAR(10), reqID  VARCHAR(10), request
      VARCHAR(10),
  t  DATETIME,
  PRIMARY KEY(reqID)
);

CREATE TABLE Accessed (
  hID  VARCHAR(10), tpID  VARCHAR(10), reqID
      VARCHAR(10), response  VARCHAR(10),
  t  DATETIME,
  PRIMARY KEY(reqID)
);
```

For the authorization specification in Listing 3, the compiler generates four SQL queries corresponding to the created, expired, detached, and discharged instances (recall that in our model, authorizations cannot be violated). Listing 8 shows the SQL query that returns the created instances of the authorization at time NOW (the current time). That is, it shows $\llbracket \texttt{created}\ DisclosureAuth \rrbracket^{NOW}$—rendered into SQL. Although, in the current implementation all queries are automatically evaluated for NOW, we are working on an extension where the user could input a time value. This would allow the user to run retrospective queries such as *How many instances of this authorization were created two months ago?* and hypothetical queries such as *Given the current state of the database, augmented with some hypothetical events, how many instances of DisclosureCom commitments will be violated?*

Listing 8: Generated SQL for created instances of DisclosureAuth.

```
SELECT hID, tpID, discID, credentials, t
FROM (SELECT hID, tpID, discID, credentials, t
     FROM SentCred) AS Query0
WHERE t < NOW();
```

Listing 8 contains a nested SQL query. The query is simple and could be easily rewritten without nesting. Listing 9, which shows the SQL query for the discharged instances of the authorization, is far more complex, and contains several levels of unavoidable nesting. Such a query would be practically impossible to write by hand—which demonstrates the significant practical benefits of Custard.

Listing 9: Generated SQL for the discharged instances. The SQL DATETIME values '1000-01-01 00:00:00' and '9999-12-31 23:59:59' correspond to the 0th and the infinitely distant time instants, respectively, in our implementation. The unit of time is day.

```
SELECT
  hID, tpID, discID, credentials, reqID,
      response, t
FROM
  (SELECT
      hID, tpID, discID, credentials, reqID,
          response,
          GREATEST(Query21.t, Query28.t3) AS t
  FROM
      (SELECT
      hID, tpID, discID, credentials, t
  FROM
      SentCred) AS Query21
  NATURAL JOIN (SELECT
      hID, tpID, reqID, response, t AS t3
  FROM
      (SELECT
      hID, tpID, reqID, response,
         GREATEST(Query30.t, Query32.t4) AS t
  FROM
      (SELECT
      hID, tpID, reqID, response, discID,
          request,
          GREATEST(Query22.t, Query34.t5) AS t
  FROM
      (SELECT
      hID, tpID, reqID, response, t
  FROM
      Accessed) AS Query22
  NATURAL JOIN (SELECT
      hID, tpID, discID, reqID, request, t AS t5
  FROM
```

```
      (SELECT
      hID, tpID, discID, reqID, request, t
  FROM
      ReqData) AS Query23) AS Query34
  WHERE
      Query34.t5 + INTERVAL 0 DAY <= Query22.t
          AND Query22.t < '9999-12-31 23:59:59')
              AS Query30
  NATURAL JOIN (SELECT
      hID, tpID, reqID, response, t AS t4
  FROM
      (SELECT
      hID, tpID, reqID, response, discID,
           request,
          GREATEST(Query22.t, Query36.t6) AS t
  FROM
      (SELECT
      hID, tpID, reqID, response, t
  FROM
      Accessed) AS Query22
  NATURAL JOIN (SELECT
      hID, tpID, discID, reqID, request, t AS t6
  FROM
      (SELECT
      hID, tpID, discID, reqID, request, t
  FROM
      ReqData) AS Query24) AS Query36
  WHERE
      '1000-01-01 00:00:00' <= Query22.t
          AND Query22.t < Query36.t6 + INTERVAL
              10 DAY) AS Query31) AS Query32) AS
              Query25) AS Query28) AS Query26
WHERE
  t < NOW();
```

5. DISCUSSION

Custard is a language for specifying norms over low-level information schemas and evaluating norm instances over information stores. Custard supports subtle features that are important in real-world settings, such as nonoccurrence of events, nesting, and aggregation. Custard's novelty lies not only in that it raises norm specifications to the level of information schemas but also in how it describes a general approach to formalizing norm lifecycles. Specifically, our formalization can readily support alternative semantics for norm lifecycles—for example, considering an authorization as violated when its consequent occurs without the antecedent having occurred. The implementation of Custard to generate SQL queries is evidence of its practical value in combating complexity. SQL queries corresponding to even a simple norm turn out to complex, running in tens of lines. Thus Custard offer benefits in dealing with complexity by saving a modeler significant effort.

Custard follows a recent trend toward increasingly explicit information modeling in commitments [10, 27]. It is specifically informed by advances reported in Cupid [10], which presents an information-based language for commitments. Whereas Custard adopts Cupid's basic style and approach, it goes significantly beyond Cupid in expressiveness. Cupid supports only commitments and does not support aggregation. Cupid formulates the query extensions in terms of relational algebra whereas Custard defines them in terms of the TRC, which yields cleaner and more direct set-based formulations. The formulation and proof of stability (over model expansions) is novel to Custard.

Norms are widely studied in multiagent systems from different perspectives. Custard leverages work on first-order event-based representations of commitments and, more generally, norms [4,25, 39,40,43]. These representations often emphasize different aspects, for example, reasoning about operations on norms, richer content, and deadlines. Through being first order, these approaches naturally support distinguishing norm schemas and instances. Where Custard goes beyond existing languages is in bringing together important modeling concerns in a single expressive language with a clear information-based semantics. Features such as nesting and aggregation are not supported in any of the existing languages.

Custard may be applied toward specifying security policies. There is a significant conceptual difference from traditional approaches for specifying security policies, for instance, in languages such as XACML [28]. In such languages, one specifies the actions to be taken by the computer system upon a security-pertinent event ("obligations") or the actions that must be blocked by the system ("prohibitions"). In other words, the policies are implemented and executed by the computer system. These approaches are not sociotechnical in that they do not represent or reason about any of the social aspects. Specifically, they talk about computer systems and not sociotechnical systems. By contrast, with Custard, one may capture requirements via norms among autonomous agents. Custard leaves it to each agent, in light of its autonomy, to decide whether to satisfy or violate a norm. Singh emphasizes these distinctions, crucial for achieving secure collaboration, in greater depth in recent work on norms and cybersecurity [36,37].

In the literature on normative multiagent system, important overlapping themes relevant to Custard concern (1) modeling institutions and contracts [4, 22, 24]; (2) how to develop sociotechnical system specifications given stakeholder requirements [8, 12] and a low-level information schema; (3) protocol specifications specified in information-based languages such as BSPL [34, 35]; (4) formulating alignment of commitment states across asynchronously communicating autonomous agents [9,11]; (5) norm reasoning and conflicts [17,29,42]; (6) monitoring and reasoning about agent compliance with norms [1, 26, 41]; and (7) programming norm-aware intelligent agents that reason about norms in deciding upon a course of action [2, 6, 13, 21, 23, 25, 41].

Future Work. Custard represents an initial effort to represent norms in an information-oriented framework. It opens up many interesting directions of work. These include improvements to Custard to accommodate valuable ideas from the literature as well as enhancements to address particular needs of real-life problems, such as in healthcare or business.

The following are some particularly important and interesting directions for future research. One, formalizing norms in higher-level database logics, such as 4QL, which are beginning to be applied in settings of collaborative agents [14] as well as defeasible reasoning. Two, extending Custard to enable capturing settings where there is uncertainty associated with the occurrence of events [38] and therefore the states of the norm instances. Three, developing an enhanced agent-oriented API to support runtime monitoring of norms and their incorporation in the agent deliberation cycle. Four, creating a catalog of norm patterns that arise in real-life settings and attempting to encode them in Custard. This exercise would point to directions in which Custard would need to be extended. Five, developing a tool-supported methodology for writing Custard specifications, including well-formedness criteria for specifications, for example, relating to the specification of time intervals and keys. Six, relating Custard specifications to alignment for ensuring that multiple Custard stores remain adequately synchronized.

Acknowledgments

We thank the anonymous reviewers for their helpful comments on a previous version of this paper. Munindar Singh thanks the US Department of Defense for partial support under the Science of Security Lablet.

REFERENCES

[1] N. Alechina, N. Bulling, M. Dastani, and B. Logan. Practical run-time norm enforcement with bounded lookahead. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, pages 443–451. IFAAMAS, 2015.

[2] N. Alechina, M. Dastani, and B. Logan. Programming norm-aware agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1057–1064, Valencia, 2012. IFAAMAS.

[3] G. E. M. Anscombe. On brute facts. *Analysis*, 18(3):69–72, Jan. 1958.

[4] A. Artikis, M. J. Sergot, and J. V. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1):1:1–1:42, Jan. 2009.

[5] J. L. Austin. *How to Do Things with Words*. Clarendon Press, Oxford, 1962.

[6] M. Baldoni, C. Baroglio, and F. Capuzzimati. A commitment-based infrastructure for programming socio-technical systems. *ACM Transactions on Internet Technologies*, 14(4):23:1–23:23, Dec. 2014.

[7] A. K. Chopra, A. Artikis, J. Bentahar, M. Colombetti, F. Dignum, N. Fornara, A. J. I. Jones, M. P. Singh, and P. Yolum. Research directions in agent communication. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 42(2):20:1–20:23, Mar. 2013.

[8] A. K. Chopra, F. Dalpiaz, F. B. Aydemir, P. Giorgini, J. Mylopoulos, and M. P. Singh. Protos: Foundations for engineering innovative sociotechnical systems. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, pages 53–62, Karlskrona, Sweden, Aug. 2014. IEEE Computer Society.

[9] A. K. Chopra and M. P. Singh. Multiagent commitment alignment. In *Proceedings of the 8th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 937–944, Budapest, May 2009. IFAAMAS.

[10] A. K. Chopra and M. P. Singh. Cupid: Commitments in relational algebra. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI)*, pages 2052–2059, Austin, Texas, Jan. 2015. AAAI Press.

[11] A. K. Chopra and M. P. Singh. Generalized commitment alignment. In *Proceedings of the 14th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 453–461, Istanbul, May 2015. IFAAMAS.

[12] A. K. Chopra and M. P. Singh. From social machines to social protocols: Software engineering foundations for sociotechnical systems. In *Proceedings of the 25th International World Wide Web Conference*, pages 1–12, Montréal, Apr. 2016. ACM.

[13] F. Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79, Mar. 1999.

[14] B. Dunin-Kęplicz and A. Strachocka. Tractable inquiry in information-rich environments. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 281–300, Buenos Aires, Aug. 2015.

[15] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin Cummings, 2nd edition, 1994.

[16] D. Fitzpatrick. Searle and collective intentionality: The self-defeating nature of internalism with respect to social facts. *American Journal of Economics and Sociology*, 62(1):45–66, Jan. 2003.

[17] K. V. Hindriks and M. B. V. Riemsdijk. A real-time semantics for norms with deadlines. In *Proceedings of the 12th International Conference on Autonomous agents and Multiagent Systems*, pages 507–514, St. Paul, Minnesota, 2013. IFAAMAS.

[18] HL7. Consent directive use cases. http://wiki.hl7.org/index.php?title=Consent_Directive_Use_Cases.

[19] W. N. Hohfeld. *Fundamental Legal Conceptions as Applied in Judicial Reasoning and other Legal Essays*. Yale University Press, New Haven, Connecticut, 1919. A 1919 printing of articles from 1913.

[20] A. J. I. Jones and M. J. Sergot. On the characterisation of law and computer systems: The normative systems perspective. In J.-J. C. Meyer and R. J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*, chapter 12, pages 275–307. John Wiley and Sons, Chichester, UK, 1993.

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

[22] T. C. King, T. Li, M. D. Vos, V. Dignum, C. M. Jonker, J. Padget, and M. B. van Riemsdijk. A framework for institutions governing institutions. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, pages 473–481. IFAAMAS, 2015.

[23] F. Meneguzzi and M. Luck. Norm-based behaviour modification in BDI agents. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 177–184. IFAAMAS, 2009.

[24] F. Meneguzzi, S. Miles, M. Luck, C. Holt, and M. Smith. Electronic contracting in aircraft aftercare: A case study. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) Industry Track*, pages 63–70, Estoril, Portugal, May 2008. IFAAMAS.

[25] F. Meneguzzi, P. R. Telang, and M. P. Singh. A first-order formalization of commitments and goals for planning. In *Proceedings of the 27th Conference on Artificial Intelligence (AAAI)*, pages 697–703, Bellevue, Washington, July 2013. AAAI Press.

[26] S. Modgil, N. Oren, N. Faci, F. Meneguzzi, S. Miles, and M. Luck. Monitoring compliance with E-contracts and norms. *Artificial Intelligence and Law*, 23(2):161–196, 2015.

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

[28] OASIS. eXtensible Access Control Markup Language (XACML) version 3.0 specification document. *OASIS Standard*, Aug. 2010. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf.

[29] A. Rotolo, G. Governatori, and G. Sartor. Deontic defeasible reasoning in legal interpretation: Two options for modelling interpretive arguments. In *Proceedings of the 15th International Conference on Artificial Intelligence and Law*, pages 99–108. ACM, 2015.

[30] M. Sbisá. How to read Austin. *Pragmatics*, 17(3):461–473, Sept. 2007.

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

[32] M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, Dec. 1998.

[33] M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7(1):97–113, Mar. 1999.

[34] M. P. Singh. Information-driven interaction-oriented programming: BSPL, the Blindingly Simple Protocol Language. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 491–498, Taipei, May 2011. IFAAMAS.

[35] M. P. Singh. Semantics and verification of information-based protocols. In *Proceedings of the 11th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1149–1156, Valencia, Spain, June 2012.

[36] M. P. Singh. Norms as a basis for governing sociotechnical systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):21:1–21:23, Dec. 2013.

[37] M. P. Singh. Cybersecurity as an application domain for multiagent systems. In *Proceedings of the 14th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1207–1212, Istanbul, May 2015. IFAAMAS. Blue Sky Ideas Track.

[38] A. Skarlatidis, A. Artikis, J. Filippou, and G. Paliouras. A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming*, 15(02):213–245, 2015.

[39] N. A. M. Tinnemeier, M. Dastani, and J.-J. C. Meyer. Programming norm change. In *Proceedings of the 9th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 957–964, Toronto, 2010. IFAAMAS.

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

[41] M. B. van Riemsdijk, L. A. Dennis, M. Fisher, and K. V. Hindriks. Agent reasoning for norm compliance: A semantic approach. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 499–506. IFAAMAS, 2013.

[42] W. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman. Normative conflict resolution in multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 19(2):124–152, Oct. 2009.

[43] P. Yolum and M. P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534, Bologna, July 2002. ACM Press.