Relational Abstraction in Community-Based Secure Collaboration Technical Report 2013-1045-12

Philip W. L. Fong Pooya Mehregan
Department of Computer Science
University of Calgary
{ pwlfong, pmehrega }@ucalgary.ca

Ram Krishnan
Department of Electrical and Computer Engineering
University of Texas at San Antonio
ram.krishnan@utsa.edu

November 2013

Abstract

Users of an online community are willing to share resources because they can expect reasonable behaviour from other members of the community. Such expectations are known as social contracts. In this work, we study the specification and enforcement of social contracts in a computer mediated collaboration environment. Specifically, we examine social contracts that contain both relationship- and history-based elements. A series of policy languages, all based on modal and temporal logics, with increasing expressiveness, have been proposed to express social contracts. Reference monitors are designed to correctly and efficiently enforce the specified policies. A technique called "relational abstraction" is employed to reduce the reference monitor into a purely relationship-based protection system, that is, what is commonly known as a social network system.

1 Introduction

Online communities are increasingly organized around social computing platforms or other collaborative software. Examples include collaborative authoring through Wiki and Google Docs, internet forums, online calendar sharing, open-source development platforms like Source Forge and proprietary collaboration tools such as IBM LotusLive and Microsoft Office 365. Individuals feel safe to share their resources in a community, because they can expect reasonable behaviour

from its other members. Such behavioural expectations that form the foundation of a community is called a ***social contract*** [26].

Social contracts must be carefully articulated as well as enforced in order for them to be effective in binding individuals into a community. This work is about supporting the effective articulation and enforcement of social contracts in online communities. We use the term ***Community-based Secure Collaboration (CSC)*** to refer to the kind of online collaboration that is mediated by a globally specified and enforced social contract.

From an access control perspective, social contracts determine what accesses are acceptable, and what are not. For example, a seller who has never missed a delivery shall be granted a preferred seller status, which in turn allows her to access privileged advertisement channels. As another example, a volunteer programmer who has contributed code to a certain module of an open source software shall not be a tester for that module. Social contracts typically contain access policies that are not only about who the requestor is (identity), what quality he has (attribute) or the kind of job functions he assumes (role). In the era of social computing, it is typical for social contracts to have policies of the following kinds:

- ***Relationship-based.*** Some policies base authorization decisions on how users relate to one another.
- ***History-based.*** Other policies determine accessibility by considering how users interacted with or related to one another in the past.

Previous works in access control have addressed isolated requirements of social contract articulation and enforcement, but none provides a generic framework for specifying and enforcing policies that are simultaneously relationship based and history based. Krukow *et al.* [24, 25] proposed a history-based framework for specifying the interaction protocol between a pair of users using Pure Past Linear Temporal Logic (PPLTL), and designed an execution monitor architecture for enforcing such policies. Yet, the scheme cannot support interaction protocols (aka social contracts) that span the joint behaviour of a community of individuals. Fong [16] proposed a formal model for Relationship-based Access Control (ReBAC), in which the protection state is a social network of users, authorization decisions are based on how users are related to one another, and policies are expressed in a modal logic, with subsequent extension to employ hybrid logic as a policy language [6]. That line of work, however, does not take into account the historical interactions and relationships among users. Krishnan *et al.* [23, 21] studied a specific type of social contract that is built on the protection abstraction of groups, and employed First-Order Linear Temporal Logic (FOLTL) to specify interaction protocols. The intended application is secure information sharing. They also studied the correspondence between the formal specification and its implementation via reference monitors [22]. While their work has a history-based flavour, it applies only to a specific kind of social contract, and access control constraints are phrased in terms of the concerned user and object in a group rather than generic relationships among them or among groups.

Building on the three lines of research above, this work makes the following contributions.

1. We devised an access control model for supporting CSC (§ 5). The protection system tracks the history of the community's state, which takes the form of generalized social networks. Social contracts are expressed as policies that guard the interaction events of users.

2. Extending the modal logic of Fong [16], we designed a policy language that incorporates both modal and temporal operators for expressing intermingled relationship-based and history-based policy elements (§ 6).

3. We proposed a reference monitor architecture for enforcing social contracts specified in our policy language (§ 7). To reduce the space usage of the monitor, we employed a technique that we call ***relational abstraction***, in which historical dealings between two parties are documented as binary relationships between them. A pleasantly surprising consequence is that the resulting reference monitor tracks a single generalized social network rather than a history of social networks. Thus we obtain the core thesis of this work: *although the CSC model offers specification convenience in terms of mixing history- and relationship-based policy elements, the underlying enforcement mechanism can be optimized into a ReBAC system!*

4. Recognizing the need for further expressiveness, we proposed a hybridized extension of our policy language (§ 8). By employing relational abstraction once again, we obtained a novel fragment of the hybridized policy language that is amenable to efficient enforcement, thereby recovering the core thesis that the reference monitor is essentially a ReBAC system.

5. Previous work advocates the superiority of stateless specifications of communities, and suggests the employment of inter-community relationships to guide authorization decisions [21]. We addressed both needs in § 9.

2 Related Work

Relationship-Based Access Control

With the advent of social network systems, Gates [19] and Carminati and Ferrari [10] independently coined the term Relationship-Based Access Control to refer the access control models in which inter-personal relationships form the basis of authorization decisions. Fong formalized this type of access control in a ReBAC model [16], and showed that classical access control approaches, such as RBAC, do not offer the right abstractions for expressing policies commonly found in social computing and electronic health record systems. By adopting relationship-based access control (ReBAC), one avoids tedious role parameterization and achieves delegation in an elegant manner. A modal logic [5] with a single propositional symbol has been adopted for specifying ReBAC policies. Subsequent works study the expressiveness of ReBAC policy languages [17], and extend the modal language by way of hybridization [6] (i.e., incorporating constructs from hybrid logic [1]).

Cheng *et al.* proposed a ReBAC model with a regular expression-based policy language [12]. They subsequently extended their model to capture not only user-to-user relationships, but also user-to-resource and resource-to-resource relationships [11].

None of the above works take into account the history of interactions and relationships in policy specification and enforcement. Support for temporal reasoning is necessary for expressing interaction protocols in social contracts. A contribution of this work is to demonstrate that such a history-based extension of ReBAC is a non-trivial research undertake, involving complex algorithm design issues and shift of representation (via relational abstraction) to achieve space-efficient

enforcement. This present work is also capable of expressing user-to-resource and resource-to-resource relationships.

Other Protection Models for Social Computing

Carminati *et al.* proposed an access control model for social network systems that is based on a trust metric [9]. A subsequent work explored the use of established semantic web technologies for encoding policies and relationships [7, 8]. ACON is an access control model designed specifically for social computing [29, 28]. It controls user activities by considering, e.g., user and resource attributes as well as user-to-user relationships.

History-Based Access Control and Runtime Verification

History-based access control tracks past access events and uses access history as the basis for authorization. Previous work includes both system implementation techniques [14] as well as theoretical study of enforceability [30, 15, 20, 27]. Of particular relevance to this work is the work of Krukow *et al.* [24, 25]. They used a PPLTL as a policy language for specifying the interaction protocol between two parties, and proposed a reference monitor framework for enforcing the policies. Their work does not capture community-wide policies.

Our reference monitor framework employs two related techniques for optimizing space usage: (1) using LTL expansion laws [2, Chapter 5] to reduce dependencies on history, and (2) augmenting the generalized social network with additional edges to record historical dealings between parties. Similar techniques have been employed in previous works: (1) in [3, 24, 25] and (2) in [13, 4]. Our novelty has been in applying these techniques in (a) demonstrating that a ReBAC system is all that one requires to enforce CSC policies specified in our policy languages, and (b) devising an appropriate fragment of our hybridized policy language that is amenable to such optimization techniques.

Group-centric Secure Information Sharing

Krishnan *et al.* proposed a formal specification of group-centric secure information sharing (g-SIS) [23, 21]. LTL is employed to specify the various properties of groups, as well as accessibility conditions. In this sense, g-SIS authorization is history-based. Yet g-SIS is only a specific form of community, and the specification approach does not support notions of accessibility that depends on relationships among users or groups. Subsequent work also applies formal methods to establish the correspondence between a stateless specification and a stateful implementation (i.e., reference monitor) of g-SIS [22]. In our work, the policy specification is guaranteed to be faithfully enforced by the reference monitor architecture. Manual application of formal methods is not required. Their work, however, inspired the language extensions in § 9 for facilitating stateless specification.

3 Use Cases

This section outlines use cases that are representative of the applications targeted by our CSC protection model.

Example 1 (Online Social Networks (OSNs))**.** *OSNs such as Facebook have recently been used for purposes other than staying in touch with friends. Specifically, people in many countries co-ordinated and planned their political gatherings and demonstrations using Facebook. This drew the attention of totalitarian governments, resulting in surveillance of such OSNs. It is therefore desirable for users to be able to set up fine-grained access control policies. Here is an example.*

Sarah, a member of an OSN, is a supporter for a minority party called Fighting Citizens (FC). FC has been banned from further activities by the majority party, which has risen to power recently. In the past, Sarah has posted contents in favour of FC in an OSN group of FC supporters. She as well as other members of that group wish to prevent certain users from joining that group. Specifically, OSN users who previously joined OSN groups associated with the governing party are not allowed to join the FC-supporter group.

Example 2 (Online Coauthoring Communities)**.** *Some online communities, such as Wikipedia, provide users with an environment for coauthoring contents and collective editing. Since deviant users may not conform to the social contract that defines a community, such communities need to be equipped with means for formal articulation of social contracts as well as mechanisms for enforcing these contracts. The following are sample components of such social contracts.*

1. *A user who has been reported for using inappropriate language twice is suspended for further editing.*
2. *A user who has already created two distinct objects that have since remained untouched by any member of the community (including herself) is not allowed to further create new objects.*

Example 3 (Group-centric Secure Information Sharing)**.** *Consider the "***secure meeting room (SMR)***" metaphor in [23, 21]. Each SMR is a virtual group that defines a protection domain. Objects can be created in a group. Users may join, leave or rejoin a group. Whether a user may access an object depends on the membership history of that user in the group where the object was created. The access control system tracks multiple groups, and a user may be a member of more than one group. Typical applications of SMR include secure information sharing: e.g., companies collaborating to develop a product, and using an SMR as an arena for engineers of participating companies to share documents. The following are a few possible features of an SMR system.*

GSIS-1 *Some sanity conditions must be met: (a) a user shall not leave a group unless she has joined it; (b) a user shall not join a group unless either (i) she has never joined the group, or (ii) she has left the group; (c) an object can only be created once.*

GSIS-2 *A user may read an object if the latter is created while the user still belongs to the group in which the object is created.*

GSIS-3 *A user may read an object if, at the time of access, the user belongs to the group in which the object is created.*

GSIS-4 *There is a hierarchical dominance relation among groups. If group g_1 dominates group g_2, then membership in g_1 implies membership in g_2. (This feature is proposed as future work in [23, 21].)*

4 Notations

Sets and Functions Let $\mathbb{N}$ be the set of natural numbers $\{0, 1, 2, \ldots\}$, and $\mathbb{B}$ be $\{0, 1\}$, the boolean values.

Given a function $f : A \to B$, we write $\mathsf{dom}(f)$ for the domain of f (i.e., the set A), and $\mathsf{ran}(f)$ for the range of f (i.e., the set $\{f(x) \mid x \in A\}$). We write $f : A' \rightharpoonup B$ if $\mathsf{dom}(f) \subseteq A'$ and $\mathsf{ran}(f) \subseteq B$. Suppose $f : A \rightharpoonup B$, $a \in A$, and $b \in B$. We then write $f[a \mapsto b]$ to denote the function $g : A \rightharpoonup B$ such that $g(a) = b$, and $g(x) = f(x)$ for $x \neq a$. Given $a_1, \ldots, a_n \in A$ and $b_1, \ldots, b_n \in B$, where $a_1, \ldots, a_n$ are pairwise distinct, we write $\{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$ to denote the function $g : A \rightharpoonup B$ such that $g(a_1) = b_1, \ldots, g(a_n) = b_n$.

Relational Structures Intuitively, a ***relational structure*** (or simply a ***graph***) is an edge-labelled directed graph. Formally, a relational structure is a pair $G = \langle V, \{R_l\}_{l \in \mathcal{L}} \rangle$. The component V is a *finite* set of vertices. The component $\{R_l\}_{l \in \mathcal{L}}$ is an indexed family of binary relations. The set $\mathcal{L}$ is a *finite* set of edge labels. For each label $l \in \mathcal{L}$, $R_l \subseteq V \times V$ is a binary relation defined over V. We also write $V(G)$, $E(G)$ and $R_l(G)$ to represent V, $\{R_l\}_{l \in \mathcal{L}}$ and R_l respectively.

Given a *countably infinite* set $\mathcal{V}$ and a *finite* set $\mathcal{L}$, we denote by $\mathcal{G}[\mathcal{V}, \mathcal{L}]$ the set of all relational structures $\langle V, \{R_l\}_{l \in \mathcal{L}} \rangle$ for which $V \subseteq \mathcal{V}$. Note that $\mathcal{L}$ is *common* across every $G \in \mathcal{G}[\mathcal{V}, \mathcal{L}]$, and $V(G)$ is always finite.

Given $G \in \mathcal{G}[\mathcal{V}, \mathcal{L}]$, $u, v \in \mathcal{V}$, and e may or may not be a member of $\mathcal{L}$, then $G + e(u, v)$ denotes the graph $\langle V', \{R'_l\}_{l \in \mathcal{L} \cup \{e\}} \rangle$, where $V' = V(G) \cup \{u, v\}$, $R'_e = R_e \cup \{(u, v)\}$ if $e \in \mathcal{L}$, $R'_e = \{(u, v)\}$ if $e \notin \mathcal{L}$, and $R'_l = R_l$ if $l \neq e$.

5 Access Control Model

This section defines a formal model for CSC. The model consists of two parts: (a) a behaviour model for the community (§ 5.1), and (b) a protection system that constrains the behaviour of that community based on explicitly specified policies (§ 5.2).

5.1 Community

A community is modeled as a state transition system. At any point, the community is in a ***social state*** (§ 5.1.1). A ***communication event*** (§ 5.1.2) that occurs within a state triggers a ***state transition*** (§ 5.1.3), bringing the community to a new social state. Formally, a community is specified as a ***community schema***, which is a 4-tuple $\langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \rightarrow \rangle$. We explain these 4 components below.

5.1.1 Social States

A community evolves over time. At any point of time the community is in a certain social state. In this work we are interested in basing authorization decisions on the history of this evolving social state. We see the social state of a community to be relational in nature: the social state is characterized by, for example, interpersonal relationships that are either persistant or transient, past dealings with one another, and their reputational assessment of one another. There are also times when users and their relationships are not sufficient for modelling the community's state.

For example, there may be non-user entities including resources (e.g., files, photos, user profiles, etc) and social constructions (e.g., groups, departments, etc), as well as their relationships with one another and with users, that may inform authorization decisions. We use the generic term ***entities*** to refer to either user or non-user entities.

We model the social state of a community as a relational structure. Recall that a relational structure is an edge-labelled directed graph. The vertices of the graph represent entities (i.e., either users or non-user entities). A directed edge models a relationship between two entities, and the label of the edge indicates the type of relationship represented by that edge. Note that the social network of the users in the community is a subgraph of this relational structure.

In a community schema $\langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \rightarrow \rangle$, the component $\mathcal{V}$ is a countably infinite set of ***entity identifiers***. The social state of the community consists of a finite subset of $\mathcal{V}$. The component $\mathcal{R}$ is a finite set of ***relation identifiers***. The set $\mathcal{R}$ specifies the set of relationship types among entities. Therefore, the space of social states is $\mathcal{G}[\mathcal{V}, \mathcal{R}]$.

5.1.2 Communication Events

As a community evolves over time, its social state changes. We assume that a community evolves only because of user activities, and model an activity as a ***communication event*** (or simply an ***event***). In essence, the model does not account for any spontaneous change in social state that is not accompanied by an event explicitly tracked by the protection system[1]. In short, an event is the sole trigger for a community to transition from one social state to another.

The objective of the protection system is to control the triggering of communication events. While a communication event may involve many participants, we assume that only two of them are significant for the purpose of authorization. More specifically, the request to trigger a communication event presents to the reference monitor an ***event signature*** of the form $e(u, v)$, where:

- e is an ***event identifier***, the type of this event,
- u is the ***initiator***, the entity to initiate this event and
- v is the ***target***, the entity to which this event is directed.

For example, the event signature $\mathsf{send}(\mathsf{john}, \mathsf{mary})$ represents the event of John sending Mary a message. From the perspective of the protection system, only the event signature (but not the event) is visible. Hereafter we use the terms event and event signature interchangeably. We also write $\mathcal{ES}[\mathcal{E}, \mathcal{V}]$ for the set of all event signatures $e(u, v)$ for which $e \in \mathcal{E}$ and $u, v \in \mathcal{V}$. The component $\mathcal{E}$ in a community schema $\langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \rightarrow \rangle$ is a *finite*, *non-empty* set of event identifiers for that community.

5.1.3 State Transition

Given a community schema $\langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \rightarrow \rangle$, $\cdot \stackrel{\cdot}{\rightarrow} \cdot \subseteq \mathcal{G}[\mathcal{V}, \mathcal{R}] \times \mathcal{ES}[\mathcal{E}, \mathcal{V}] \times \mathcal{G}[\mathcal{V}, \mathcal{R}]$ is a transition relation. That is, $G \xrightarrow{e(u,v)} H$ when the occurrence of event $e(u, v)$ in state G transitions the

[1]Examples of such spontaneous events may include time out, change of environmental conditions, or feedback events generated by the system itself. One way to accommodate such events into our model, is to introduce a special entity into the relational structure, for representing the community itself. An event in which both the initiator and target are this special entity represents a spontaneous event.

community into state H.

Furthermore, the transition relation must be ***well-formed***: if $G \xrightarrow{e(u,v)} H$, then $V(G) = V(H)$ and $u, v \in V(G)$. A well-formed transition does not alter the vertex set of the graph, and the initiator and target are both members of the vertex set. That is, entity creation is modeled as dormant entities becoming active over time[2].

5.2 Protection System

If a community schema specifies the possible behaviour of a community, then a protection system specifies a mechanism that constrains such behaviour (§ 5.2.3). Specifically, such a protection system is a reference monitor. When a communication event is to be triggered, the signature of that event is first presented to the protection system for authorization (§ 5.2.4). Unless the system grants the event, the latter will be suppressed. The reference monitor is stateful: it tracks the historical evolution of social state of the community, as well as events that occurred in the past (§ 5.2.1). The reference monitor applies an access control policy to the reference monitor state for authorization decisions (§ 5.2.2).

5.2.1 Traces

We model the history of the community in a ***trace***, which is a sequence of relational structures. More specifically, a sequence of transitions

$$G_0 \xrightarrow{e_1(u_1,v_1)} G_1 \xrightarrow{e_2(u_2,v_2)} G_2 \cdots \xrightarrow{e_n(u_n,v_n)} G_n$$

can be encoded as the sequence of relational structures

$$G_0 \cdot (G_1 + e_1(u_1, v_1)) \cdot (G_2 + e_2(u_2, v_2)) \cdots (G_n + e_n(u_n, v_n))$$

That is, the first element of the sequence, G_0, is the oldest snapshot in the history, and, for each time point $0 < i \leq n$, the relational structure $G_i + e_i(u_i, v_i)$ contains both the social state G_i as well as an edge that records the communication event that produced G_i (i.e., the event $e_i(u_i, v_i)$). In summary, the history of a community is modeled as a sequence from $\mathcal{G}[\mathcal{V}, \mathcal{L}]^+$, where $\mathcal{L} = \mathcal{R} \uplus \mathcal{E}$. An edge with a label in $\mathcal{R}$ is a ***relation edge***, and an edge with a label in $\mathcal{E}$ is an ***event edge***. Such a sequence is what we call a trace.

Suppose we fix $\mathcal{V}$ and $\mathcal{L} = \mathcal{R} \uplus \mathcal{E}$. A trace $\gamma = G_0 G_1 \cdots G_n$ is ***well formed*** if both of the following hold:

- $G_0 \in \mathcal{G}[\mathcal{V}, \mathcal{R}]$, and
- for $1 \leq i \leq n$, $G_i = G'_i + e_i(u, v)$ for some $G'_i \in \mathcal{G}[\mathcal{V}, \mathcal{R}]$ and $e \in \mathcal{E}$.

We denote by $\mathcal{T}[\mathcal{V}, \mathcal{R}, \mathcal{E}]$ the set of all well-formed traces.

[2]This requirement is imposed to reduce the complexity of presentation. The subtlety of entity creation can be easily incorporated into the model, at the cost of unnecessary complexities that do not inspire.

5.2.2 Access Control Policies

A protection system controls the authorization of communication events in a community. In such a system, one needs to impose policies for access control. A ***policy predicate*** (or simply a ***policy***) P is a function of the following signature.

$$\mathcal{T}[\mathcal{V}, \mathcal{R}, \mathcal{E}] \times \mathcal{V} \times \mathcal{V} \to \mathbb{B} \tag{1}$$

That is, a policy $P(\gamma, u, v)$ takes as input a well-formed trace γ, an initiator vertex u and a recipient vertex v, and returns an authorization decision. We further demand that, if G is the last element of γ, then $P(\gamma, u, v) = 1$ implies $u, v \in V(G)$. We write $\mathcal{P}[\mathcal{V}, \mathcal{R}, \mathcal{E}]$ for the set of all such policy predicates.

5.2.3 Protection System

Given a community schema $\mathcal{C} = \langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \to \rangle$, a ***protection system*** for $\mathcal{C}$ is a pair $\langle \mathcal{C}, \mathit{policy} \rangle$, where the function $\mathit{policy} : \mathcal{E} \to \mathcal{P}[\mathcal{V}, \mathcal{R}, \mathcal{E}]$ assigns a policy predicate to each event identifier. That is, policy predicate $\mathit{policy}(e)$ mediates the triggering of events of type e.

5.2.4 Authorization

A ***protection state*** (i.e., a state of the protection system) contains the information by which authorization decisions are made. In our case, a protection state is simply a well-formed trace γ. Note that this identification of protection states to well-formed traces is conceptual, meaning that the actual implementation of the protection system may choose to encode the protection state in some space-efficient representations, which is exactly what we will do in § 7. For now, we focus on the conceptual description of the model.

Let $\gamma = G_0 \cdot (G_1 + e_1(u_1, v_1)) \cdot (G_2 + e_2(u_2, v_2)) \cdots (G_n + e_n(u_n, v_n))$ be a well-formed trace (i.e., a protection state), where $n \geq 0$, $G_i \in \mathcal{G}[\mathcal{V}, \mathcal{R}]$ for $0 \leq i \leq n$, and $e_i \in \mathcal{E}$ for $1 \leq i \leq n$. We write $\mathsf{head}(\gamma)$ for the graph G_n.

We model *both* the transition of protection states and authorization with the binary relation $\Longrightarrow \subseteq \mathcal{T}[\mathcal{V}, \mathcal{R}, \mathcal{E}] \times \mathcal{T}[\mathcal{V}, \mathcal{R}, \mathcal{E}]$. Intuitively, the relation $\Longrightarrow$ transitions one protection state to another protection state (which are both well-formed) of length one more than the first. Formally, the transition $\gamma \Longrightarrow \gamma \cdot (G + e(u, v))$ holds whenever both of the following conditions are met:

1. $\mathsf{head}(\gamma) \xrightarrow{e(u,v)} G$. That is, G is the next social state according to the social state transition relation.

2. $\mathit{policy}(e)(\gamma, u, v) = 1$. That is, the policy predicate assigned to event e authorizes the transition.

Example 4 (Online Social Networks)**.** *We model Example 1 using our protection model. We begin by describing the community $\mathcal{C} = \langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \to \rangle$. The entity set is $\mathcal{V} = \mathcal{U} \uplus \mathcal{G}$, which is partitioned into the set $\mathcal{U}$ of users and the set $\mathcal{G}$ of user groups. We assume that groups may "black-list" one another. For example, the FC group black-lists all the groups associated with the majority party. To support that, the relation identifier set is set up to be $\mathcal{R} = \{ \mathsf{bl} \}$, such that if there is a* bl *edge*

from group g_1 to group g_2, it means g_1 has black-listed g_2. We assume that the bl *edges are static, and they do not change over time. At run time, users may join groups. We model that by setting the event identifier set to be $\mathcal{E} = \{\,\mathsf{join}\,\}$, such that the event signature $\mathsf{join}(u, g)$ signifies the event of user u joining group g. Now the transition relation models group joining activities. Specifically, $G \xrightarrow{\mathsf{join}(u,g)} G$. That is, the relational structure that represents the social state of the community does not change over time, as the state contains only* bl *edges. However, the graphs in a trace will contain* join *edges that record joining events.*

The protection system $\langle \mathcal{C}, policy \rangle$ specifies a policy predicate $policy(\mathsf{join})$ for guarding group joining events. Specifically, $policy(\mathsf{join})(\gamma, u, g)$ will check if u has previously joined a group g' that is black-listed by g.

Example 5 (Online Coauthoring Communities)**.** *Consider $\mathcal{C} = \langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \rightarrow \rangle$, the community of Example 2. There are two kinds of entities. $\mathcal{S}$ is the set of subjects, $\mathcal{O}$ is the set of objects, and we set $\mathcal{V} = \mathcal{S} \uplus \mathcal{O}$. There is only one relation identifier: $\mathcal{R} = \{\,\mathsf{own}\,\}$. An* own *edge from subject s to object o signifies that s is the creator of o. We assume that objects that have not yet been created are already in the relational structure. "Uncreated" objects are not joined to any* own *edge.*

We model three event identifiers: $\mathcal{E} = \{\mathsf{create}, \mathsf{edit}, \mathsf{report}\}$. The event $\mathsf{create}(s, o)$ signifies that s is creating a new object o. The event $\mathsf{edit}(s, o)$ signifies that s is editing an existing object o. The event $\mathsf{report}(s_1, s_2)$ indicates that s_1 reports the use of inappropriate language by s_2.

The transition "$G \xrightarrow{\mathsf{create}(s,o)} G + \mathsf{own}(s, o)$" adds an own *edge from the initiator s to the target o. The transition "$G \xrightarrow{\mathsf{edit}(s,o)} G$" does not alter the social state, and the same is true for "$G \xrightarrow{\mathsf{report}(s_1,s_2)} G$".*

The protection system $\langle \mathcal{C}, policy \rangle$ defines the policy predicates $policy(\mathsf{create})$, $policy(\mathsf{edit})$ and $policy(\mathsf{report})$ for guarding the three kinds of events.

Example 6 (Group-centric Secure Information Sharing)**.** *Example 3 can be specified as a community schema $\mathcal{C} = \langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \rightarrow \rangle$. The entity set $\mathcal{V} = \mathcal{U} \uplus \mathcal{O} \uplus \mathcal{G}$ consists of three disjoint types of entities: $\mathcal{S}$, $\mathcal{O}$ and $\mathcal{G}$ are respectively the set of users, objects and groups. There is no relation identifiers: i.e., $\mathcal{R} = \emptyset$. There are four events: $\mathcal{E} = \{\mathsf{join}, \mathsf{leave}, \mathsf{create}, \mathsf{read}\}$. The event $\mathsf{join}(s, g)$ indicates that user u joins group g. The event $\mathsf{leave}(u, g)$ corresponds to u leaving g. The event $\mathsf{create}(g, o)$ marks the creation of object o in group g. The event $\mathsf{read}(u, o)$ signifies u reading o. The transition relation $\rightarrow$ is an identity mapping. A protection system $\langle \mathcal{C}, policy \rangle$ can be defined so that $policy(\mathsf{join})$, $policy(\mathsf{leave})$ and $policy(\mathsf{create})$ jointly enforce **GSIS-1**, while $policy(\mathsf{read})$ enforces one of **GSIS-2**, **GSIS-3** or **GSIS-4**.*

5.3 Discussion

A limitation in the scope of this work is that our protection model handles only communication events with a binary signature. While many interaction patterns are binary or reducible to binary signatures, some other lose semantics if we force a binary representation. Group messaging is an example of events that can be represented as a binary event. The idea is to model the group by an entity, and members are related to the group entity by membership edges. The initiator of the communication event is the message sender, and the target is the group entity. While the effect of the event may involve updating a subgraph that contains the member vertices, the *event signature*

itself involves only the message sender (initiator) and the group vertex (target). An example of a high-arity event is a real-estate transaction, involving a buyer, a seller, and their agents. This time the event signature has to capture all the four parties, since it is necessary to ensure that the agents are indeed representing the buyer and the seller, and that there is no issue of dual agency. When the access control policy for an event needs to verify a relationship among three or more parties, a binary event signature is not sufficient. Although we are fully aware there are event signatures with an arity higher than two, this work illustrates that even by considering only binary events, the technical challenges are already daunting, necessitating careful design of enforcement mechanisms (§7 and §8.2). Further extension to handle events of higher arity is a future work.

6 Policy Language

One of the goals of this work is to provide configurability to online communities founded on social contracts. As social contract evolves, it should not necessitate recompiling the system source code. Rather, the community administrator shall be equipped with declarative policy languages that let him or her revise the social contract without rebuilding the system. This section describes such a declarative policy language. The targeted users of our policy languages are not end users in the community, but rather community administrators. The situation is analogous to, say, configuring a system with technically-oriented policy languages such as XACML. Our design therefore balances expressiveness and enforceability.

Recall that authorization decisions are reached by consulting policy predicates, each of which takes as input the community's history, the initiator and the target, and returns a boolean authorization decision. The policy language in this section is designed for expressing such a predicate. This language forms the basis of a more expressive language to be described in § 8. For now we examine this basic policy language to facilitate the development of our core thesis in § 7.

Our policy language is an extension of the modal language proposed in [16]. That modal language offers a means to specify binary relations between a pair of vertices in a relational structure. Inspired by [21], we extend that language by incorporating the pure-past temporal operators of Linear Temporal Logic (LTL). The result is a language that allows us to articulate binary relations of a pair of vertices in the backdrop of an evolving relational structure. We name this language $\mathcal{TR}$ (for temporal-relational).

6.1 Syntax

Policies are expressed as ***formulas*** in $\mathcal{TR}$. The abstract syntax of formulas is given below:

$$\phi ::= \top \mid \mathsf{target} \mid \neg\phi \mid \phi \vee \phi \mid \langle l \rangle \phi \mid \langle -l \rangle \phi \mid \mathcal{Y}\,\phi \mid \phi\,\mathcal{S}\,\phi$$

where $l \in \mathcal{L}$, with $\mathcal{L}$ being the edge label set (i.e., $\mathcal{R} \uplus \mathcal{E}$). We write $\mathsf{Form}_{\mathcal{TR}}(\mathcal{L})$ to denote the set of all formulas generated by the above grammar.

A formula ϕ essentially specifies a search to be performed by an agent that "crawls around" a given trace $\gamma = G_0 G_1 \cdots G_n$, and determines along the way if the initiator and the target forms a certain binary relation. As the search unfolds, at any point of time, the agent is standing at a certain coordinate (i, u) within the trace γ, where i is a time point, and u is a vertex of the relational structure G_i. The agent may do three things: (1) Check the current location of the agent (via target

or $\top$); (2) Move about in the relational structure of the current time point (via $\langle l \rangle \phi$ or $\langle -l \rangle \phi$); (3) Move back in time (via $\mathcal{Y}\,\phi$ or $\phi_1\,\mathcal{S}\,\phi_2$). The above checks can be composed using basic boolean connectives such as negation ($\neg$) and disjunction ($\vee$). We formalize the above intuitive description in the following.

6.2 Semantics

The semantics of $\mathcal{TR}$ is defined in terms of the satisfaction relation $\models$. Given a trace $\gamma = G_0 G_1 \cdots G_n$ for which $n \geq 0$, an index i such that $0 \leq i \leq n$, and vertices $u, v \in V(G_n)$, the satisfaction relation $\gamma, i, u, v \models \phi$ is defined as follows:

- $\gamma, i, u, v \models \top$ always holds.
- $\gamma, i, u, v \models \mathsf{target}$ iff $u = v$.
- $\gamma, i, u, v \models \neg\phi$ iff it is not the case that $\gamma, i, u, v \models \phi$.
- $\gamma, i, u, v \models \phi_1 \vee \phi_2$ iff $\gamma, i, u, v \models \phi_1$ or $\gamma, i, u, v \models \phi_2$.
- $\gamma, i, u, v \models \langle l \rangle \phi$ iff there exists $u' \in V(G_i)$ such that $(u, u') \in R_l(G_i)$ and $\gamma, i, u', v \models \phi$.
- $\gamma, i, u, v \models \langle -l \rangle \phi$ iff there exists $u' \in V(G_i)$ such that $(u', u) \in R_l(G_i)$ and $\gamma, i, u', v \models \phi$.
- $\gamma, i, u, v \models \mathcal{Y}\,\phi$ iff $i > 0$ and $\gamma, i-1, u, v \models \phi$.
- $\gamma, i, u, v \models \phi_1\,\mathcal{S}\,\phi_2$ iff there exists j, $0 \leq j \leq i$, such that both (a) $\gamma, j, u, v \models \phi_2$, and (b) for all k, $j < k \leq i$, we have $\gamma, k, u, v \models \phi_1$.

Note that, in the inductive definition of $\models$, the pair (i, u) is the "coordinate" of the agent, and γ and v remains constant as the structural induction unfolds. In particular v gives the meaning of the propositional symbol target, which records the target entity's location.

Lastly, we also write the shorthand $\gamma, u, v \models \phi$ to mean $\gamma, n, u, v \models \phi$. (Recall that $n = |\gamma| - 1$.) That is, we begin traversal at the most recent time point. Essentially, this shorthand checks if a certain binary relationship exists between u and v within the history γ. The binary relation in question is specified as ϕ, with the propositional symbol target representing v, and the agent begins evaluation at u.

6.3 Derived forms

Standard derived forms can be defined as follows:

$$\bot = \neg\top \qquad \phi_1 \wedge \phi_2 = \neg(\neg\phi_1 \vee \neg\phi_2) \qquad \mathcal{O}\,\phi = \top\,\mathcal{S}\,\phi$$
$$[l]\phi = \neg\langle l \rangle \neg\phi \qquad [-l]\phi = \neg\langle -l \rangle \neg\phi \qquad \mathcal{H}\,\phi = \neg\,\mathcal{O}\,\neg\phi$$

The box operator $[l]$ is the dual of $\langle l \rangle$. Specifically, the formula $[l]\phi$ holds at coordinate (i, u) if every l-neighbor of u satisfies ϕ at time point i. The converse $[-l]$ can be understood similarly. The "once" operator $\mathcal{O}$ checks that there was at least one point in the past for which the operand formula ϕ holds. The "historically" operator $\mathcal{H}$ checks that the formula ϕ holds in every past time point.

6.4 Policy predicate

Given a formula ϕ, we write $[\![\phi]\!]$ to denote the policy predicate induced by ϕ. Specifically, $[\![\phi]\!](\gamma, u, v)$ returns 1 whenever $\gamma, u, v \models \phi$.

Previously, a protection system is defined as a pair $\langle \mathcal{C}, policy \rangle$. The component $policy$ is supposed to map event identifiers to *policy predicates*. As we will use our policy language to express policy predicates, we also allow $policy$ to be a function of type $\mathcal{E} \to \mathsf{Form}_{\mathcal{TR}}(\mathcal{R} \cup \mathcal{E})$: that is, $policy$ assigns a formula to each event identifier. We then take $[\![policy(e)]\!]$ to be the policy predicate guarding event identifier e. We adopt this convention uniformly in the following.

With the convention above, the range of the function $policy$, written $\mathsf{ran}(policy)$, is the set of all formulas that are used by the protection system for guarding event identifiers. This set will be of interest to us in the sequel.

6.5 Examples

Example 7 (Online Social Networks)**.** *Continuing with Example 4, we capture the policy of Example 1 as* $policy(\mathsf{join})$*. Specifically, the formula below specifies the policy predicate that guards the event* $\mathsf{join}(u, g)$*:*

$$\neg\, \mathcal{O}\langle \mathsf{join} \rangle \langle -\mathsf{bl} \rangle \mathsf{target}$$

The formula exploits the fact that the social state does not change over time.

Example 8 (Online Coauthoring Communities)**.** *Continuing with Example 5, we capture Policy 1 of Example 2 as* $policy(\mathsf{create})$*. Specifically, the event* $\mathsf{create}(s, o)$ *is allowed only if the following holds.*

$$\neg\, \mathcal{O}\left((\langle -\mathsf{report} \rangle \top) \;\wedge\; (\mathcal{Y}\, \mathcal{O} \langle -\mathsf{report} \rangle \top) \right) \tag{2}$$

The formula detects violation of Policy 1 by looking for two distinct, historical occurrences of report *edges that point to the initiator.*

Example 9 (Group-centric Secure Information Sharing)**.** *Continuing with Example 6, we formulate* $policy(\mathsf{leave})$ *and* $policy(\mathsf{join})$ *to enforce part (a) and (b) of* ***GSIS-1*** *from Example 3. Specifically, the events* $\mathsf{leave}(u, g)$ *and* $\mathsf{join}(u, g)$ *are guarded respectively by the following:*

$$(\neg \langle \mathsf{leave} \rangle \mathsf{target})\, \mathcal{S} (\langle \mathsf{join} \rangle \mathsf{target}) \tag{3}$$

$$(\neg\, \mathcal{O} \langle \mathsf{join} \rangle \mathsf{target}) \vee ((\neg \langle \mathsf{join} \rangle \mathsf{target})\, \mathcal{S} (\langle \mathsf{leave} \rangle \mathsf{target})) \tag{4}$$

Condition (3) demands that, to leave a group, the user must have joined the group in the past, and have not left the group since then. Condition (4) requires that, to join a group, the user must satisfy either one of the following: (a) the user has never joined the group, or (b) the user has not joined the group since last leaving it.

7 Reference Monitor

7.1 Motivation

The enforcement of policies can be achieved by a reference monitor, which is essentially an implementation of the protection system as described in § 5.2. To fix thoughts, let G_0 be the initial

social state of the community, and G_t be the social state of the community after the t'th transition ($t \geq 1$). Define also traces $\gamma_0 = G_0$, and $\gamma_t = \gamma_{t-1} \cdot G_t$ (for $t \geq 1$). That is, γ_t is the history of the community after the t'th transition. At time point t, if the reference monitor intercepts an event signature $e(u, v)$, it evaluates the policy predicate $policy(e)$ against γ_t, u and v. Therefore, a naively implemented reference monitor will have to track the entire history of the community (i.e., γ_t). In addition, if authorization is successful, then transition occurs, and the reference monitor will have to update its internal state to $\gamma_{t+1} = \gamma_t \cdot G_{t+1}$. This naive implementation strategy is obviously not practical for its unreasonable memory requirement. We explore in the following a space-efficient implementation strategy, in which the memory usage is comparable to tracking only one relational structure.

7.2 Implementation Strategy

The core insight is inspired by the work of Krukow *et al.* [25], who proposed a scheme for enforcing a safety property, specified as a pure-past LTL formula, using a reference monitor. Every authorization decision is made by a constant-time check against the current state of the reference monitor. The space requirement for their reference monitor is linear to the size of the formula but independent of the size of the history. Every system transition involves the execution of an "update algorithm," which changes the internal state of the reference monitor. Efficient memory usage is possible because of the extensive application of ***expansion laws*** in LTL [2, Chapter 5]. Applying their insights to our case, we observe the following:

$$\gamma_{t+1}, u, v \models \mathcal{Y}\,\phi \text{ iff } \gamma_t, u, v \models \phi \tag{5}$$

$$\gamma_{t+1}, u, v \models \phi_1\,\mathcal{S}\,\phi_2 \text{ iff } (\gamma_{t+1}, u, v \models \phi_2) \text{ or } \Big((\gamma_t, u, v \models \phi_1\,\mathcal{S}\,\phi_2) \text{ and } (\gamma_{t+1}, u, v \models \phi_1)\Big) \tag{6}$$

Note that, in the cases above, the evaluation of a formula at time point $t+1$ can be reduced to either (i) the evaluation of a subformula at time point t, or (ii) the evaluation of a *proper* subformula at time point $t+1$. This suggests the reference monitor only needs to track the evaluation results of all subformulas at a given time, and, when a transition occurs, the reference monitor updates the evaluation results by consulting the evaluation results of the previous state, and performs the updates from subformulas to superformulas.

7.3 Notation

We begin by defining a few supporting notations. We write $\mathsf{succ}^l_G(u)$ to denote the set $\{v \in V(G) \mid (u, v) \in R_l(G)\}$. That is, $\mathsf{succ}^l_G(u)$ is the set of all l-successors of vertex u in graph G. Similarly, the set of all l-predecessors of u in G, written $\mathsf{pred}^l_G(u)$, is the set $\{v \in V(G) \mid (v, u) \in R_l(G)\}$. Let $\mathsf{sub}(\phi)$ be the set of all subformulas of ϕ. Note that ϕ is also considered a subformula of ϕ itself. A subformula of ϕ that is not ϕ itself is called a ***proper subformula***. Given a set Φ of formulas, we also write $\mathsf{sub}(\Phi)$ for $\bigcup_{\phi\in\Phi} \mathsf{sub}(\phi)$. An enumeration of formulas in a set Φ is in an ***admissible order*** if the enumeration is generated as follows:

AO-1 First, list the formulas for which the outermost operator is $\mathcal{Y}$. Among these formulas, list the superformulas before the subformulas.

Algorithm 1: $Update(G, B)$

in : $G \in \mathcal{G}[\mathcal{V}, \mathcal{R} \cup \mathcal{E}]$
in out: $B : \mathsf{sub}(\mathsf{ran}(policy)) \times V(G) \times V(G) \to \mathbb{B}$
for $\phi \in \mathsf{sub}(\mathsf{ran}(policy))$, *in an admissible order,* **do**
  for $u \in V(G)$ **do**
    for $v \in V(G)$ **do**
      switch ϕ **do**
        case $\top$
          $B[\phi][u][v] \leftarrow 1$;
        case target
          $B[\phi][u][v] \leftarrow (u = v)$;
        case $\neg\phi_1$
          $B[\phi][u][v] \leftarrow \neg B[\phi_1][u][v]$;
        case $\phi_1 \vee \phi_2$
          $B[\phi][u][v] \leftarrow B[\phi_1][u][v] \vee B[\phi_2][u][v]$;
        case $\langle l \rangle \phi_1$
          $B[\phi][u][v] \leftarrow \bigvee_{u' \in \mathsf{succ}^l_G(u)} B[\phi_1][u'][v]$;
        case $\langle -l \rangle \phi_1$
          $B[\phi][u][v] \leftarrow \bigvee_{u' \in \mathsf{pred}^l_G(u)} B[\phi_1][u'][v]$;
        case $\mathcal{Y}\,\phi_1$
          $B[\phi][u][v] \leftarrow B[\phi_1][u][v]$;
        case $\phi_1\,\mathcal{S}\,\phi_2$
          $B[\phi][u][v] \leftarrow B[\phi_2][u][v] \vee (B[\phi][u][v] \wedge B[\phi_1][u][v])$;

AO-2 Second, list the remaining formulas, with subformulas listed before the superformulas.

For example, the following enumeration of the members of $\mathsf{sub}(\mathcal{Y}\,\mathcal{Y}\langle l \rangle \mathsf{target})$ is in an admissible order:

$$\mathcal{Y}\,\mathcal{Y}\langle l \rangle \mathsf{target} \qquad \mathcal{Y}\langle l \rangle \mathsf{target} \qquad \mathsf{target} \qquad \langle l \rangle \mathsf{target}$$

7.4 Reference Monitor State

The mutable state of the reference monitor is captured in a pair of data structures $\langle G, B \rangle$:

1. G is a relational structure, and
2. $B : \mathsf{sub}(\mathsf{ran}(policy)) \times V(G) \times V(G) \to \mathbb{B}$ is a three-dimensional boolean matrix. The first index is a subformula of a member of $\mathsf{ran}(policy)$, which is the set of formulas used by the protection system for guarding event identifiers. The last two indices are vertices of G. Each entry of the matrix holds a boolean value.

It is an invariant of the reference monitor that, after the t'th transition, the following conditions hold:

$$G = G_t \tag{7}$$

$$B[\phi][u][v] = 1 \text{ iff } \gamma_t, u, v \models \phi \tag{8}$$

7.5 Authorization and State Transition

At time point t, suppose an event signature $e(u, v)$ is intercepted, then the following steps will be performed by the reference monitor:

1. Look up formula $\phi = policy(e)$.
2. Look up $B[\phi][u][v]$. If the value is 0, reject the event signature. Otherwise, proceed with the following.
3. Update G to G_{t+1}.
4. Update B by invoking Algorithm 1. The algorithm takes two arguments: (a) the updated graph G, and (b) the matrix B. On return, B will be updated.

(It is assumed that system parameters such as $\mathcal{R}$, $\mathcal{E}$ and $policy$ are global information known to the algorithm.) Steps 1 and 2 are responsible for making authorization decisions. Steps 3 and 4 are responsible for updating the internal state of the reference monitor. At the core of this procedure is the invocation of Algorithm 1 to update B.

We explain how Algorithm 1 works at the intuitive level. The algorithm essentially goes through the entries of the matrix B in a specific order, updating each entry it encounters by values of other entries. Lines 8, 10, 12, 14, 16 and 18 are direct encoding of the semantic definitions of $\top$, target, $\neg$, $\vee$, $\langle l \rangle$ and $\langle -l \rangle$. Line 20 is the encoding of (5), the expansion law for $\mathcal{Y}$. Line 22 is the encoding of (6), the expansion law for $\mathcal{S}$. The correctness of the algorithm hinges on the order in which the subformulas are enumerated on line 3.

Theorem 10. *Suppose B satisfies invariant (8). If Algorithm 1 is invoked with G_{t+1} and B, then on return B satisfies the following condition:*

$$B[\phi][u][v] = 1 \textit{ iff } \gamma_{t+1}, u, v \models \phi$$

See Appendix A.1 for a proof of the above theorem.

To establish invariants (7) and (8) in the first place, the reference monitor is initialized as follows: (i) set G to G_0; (ii) set all entries of B to 0; (iii) invoke Algorithm 1. Proving the correctness of these initialization steps is an elementary exercise.

Theorem 11. *Let $M = \sum_{\phi \in \mathsf{ran}(policy)} |\phi|$, $N = |V(G)|$, and $E = \sum_{l \in \mathcal{R} \cup \mathcal{E}} |R_l(G)|$. Algorithm 1 runs in time $O(MN(N + E))$. The space requirement for the reference monitor is the space for maintaining G plus $O(MN^2)$ bits overhead for matrix B.*

See Appendix A.2 for a proof of this theorem.

7.6 Reference Monitor as ReBAC System

The boolean matrix B could be seen as an array of adjacency matrices. Specifically, for each formula ϕ in $\mathsf{sub}(\mathsf{ran}(policy))$, the submatrix $B[\phi][\cdot][\cdot]$ characterizes a binary relation R_ϕ over $V(G)$, such that $(u, v) \in R_\phi$ iff $\gamma_t, u, v \models \phi$. Therefore, the internal state of the reference monitor, comprising of G and B, could be seen as a single relational structure $\langle V(G), \{R_l\}_{l\in\mathcal{L}}\rangle$ where $\mathcal{L} = \mathcal{R} \cup \mathcal{E} \cup \mathsf{sub}(\mathsf{ran}(policy))$. If we interpret the reference monitor state in this way, then the reference monitor essentially tracks a single relational structure, and makes authorization decisions by consulting the topology of this relational structure. *In short, the reference monitor is essentially a ReBAC system!* Specifically, if the entities of the community are all users, then the reference monitor is a social network system.

Surprisingly, even though we have introduced a temporal dimension to policy predicates, it turns out that the enforcement of these policies requires only the tracking of a single relational structure, rather than a full history of relational structures, as long as we specify policy predicates using $\mathcal{TR}$. That is, space efficiency is predicated on restraining the expressiveness of our policy language.

What is even more interesting is the way history information is tracked. Specifically, history information is abstracted into binary relations between entities (i.e., the binary relations R_ϕ above). This is possible because the policy language can only express the past dealings between two parties (i.e., the initiator and the target). This is essentially a generalization of the core observation of [16]. It has been observed in [16] that a modal logic with exactly one propositional symbol can be used for representing a binary relation between two entities in a relational structure. In our policy language, this is generalized to include a temporal dimension: a modal-temporal language with exactly one propositional symbol can be used for representing the past dealings between two entities. We call this methodological technique ***relational abstraction***:

> *By constraining the policy language to express only past dealings between two entities, history information can be tracked by binary relations.*

8 A Hybrid Policy Language

While the policy language $\mathcal{TR}$ serves well in highlighting the core thesis of this work, it is limited in expressive power. In § 8.1, we extend $\mathcal{TR}$ by incorporating constructs from hybrid logic [1, 6]. Hybrid logic offers constructs that are normally found in first-order logic rather than modal logic: the ability to name an entity and later on refer to it. This is required for separating entities and to reposition search focus. Then in § 8.2, we look at the enforcement of policies under this extension, especially in how the core thesis of this work is played out in such an expressive language.

8.1 Hybridization

We introduce hybrid constructs into the policy language $\mathcal{TR}$, resulting in a policy language that we call $\mathcal{HTR}$.

8.1.1 Syntax

The syntax of $\mathcal{HTR}$ differs from $\mathcal{TR}$ in two ways: (1) the specialized propositional symbol target is now generalized to entity variables, and (2) new constructs are introduced for defining entity variables and repositioning the crawling agent. The grammar of a formula in $\mathcal{HTR}$ is shown below.

$$\begin{aligned}\phi ::= \top \mid \neg\phi \mid \phi \vee \phi \mid \langle l\rangle\phi \mid \langle -l\rangle\phi \mid \mathcal{Y}\,\phi \mid \phi\,\mathcal{S}\,\phi \mid \\ x \mid \downarrow x\,.\,\phi \mid @_x\,\phi\end{aligned} \tag{9}$$

where $l \in \mathcal{L}$, $\mathcal{L}$ is the edge label set, and $x \in \mathsf{Var}$ is an ***entity variable***. We assume that the set Var provides a countably infinite supply of entity variables, and that target is a distinguished member of Var.

On the first line of (9) are the constructs inherited from $\mathcal{TR}$ (with the omission of target). Hybridization introduces three new constructs on the second line. Common to the three constructs is the assumption that a formula is evaluated with respect to a ***variable assignment***. A variable assignment defines the values of some entity variables. In particular, each variable is bound to a single entity in the relational structure. The first new construct x, an entity variable, tests if the crawling agent is currently located at the entity referenced by x. This is a generalization of target, as target is but a member of Var. A second new construct is the ***binder*** ($\downarrow$). The formula $\downarrow x\,.\,\phi$ introduces a new variable definition, binding x to the current location of the crawling agent. The third new construct is the ***satisfaction operator*** ($@_x$). The formula $@_x\,\phi$ repositions the crawling agent to the entity referenced by x, and then evaluates ϕ from that location. The formal semantics is given below.

8.1.2 Semantics

The semantics of formulas is given by the relation $\gamma, i, g, u \Vdash \phi$. As before, ϕ is evaluated in the context of a trace $\gamma = G_0 \ldots G_n$, recording the history of evolution of a relational structure. During evaluation, the crawling agent is situated a certain time index i where $0 \leq i \leq n$, and located at a certain entity $u \in V(G_i)$. In addition, the formula is evaluated against a variable assignment $g : \mathsf{Var} \rightharpoonup \mathcal{V}$, which assigns an entity to each of the variables in $\mathsf{dom}(g)$. Formally, the evaluation of formulas are defined as follows.

- $\gamma, i, g, u \Vdash \top$ always holds.
- $\gamma, i, g, u \Vdash \neg\phi$ iff it is not the case that $\gamma, i, g, u \Vdash \phi$.
- $\gamma, i, g, u \Vdash \phi_1 \vee \phi_2$ iff $\gamma, i, g, u \Vdash \phi_1$ or $\gamma, i, g, u \Vdash \phi_2$.
- $\gamma, i, g, u \Vdash \langle l\rangle\phi$ iff there exists $u' \in V(G_i)$ such that $(u, u') \in R_l(G_i)$ and $\gamma, i, g, u' \Vdash \phi$.
- $\gamma, i, g, u \Vdash \langle -l\rangle\phi$ iff there exists $u' \in V(G_i)$ such that $(u', u) \in R_l(G_i)$ and $\gamma, i, g, u' \Vdash \phi$.
- $\gamma, i, g, u \Vdash \mathcal{Y}\,\phi$ iff $i > 0$ and $\gamma, i-1, g, u \Vdash \phi$.
- $\gamma, i, g, u \Vdash \phi_1\,\mathcal{S}\,\phi_2$ iff there exists j, $0 \leq j \leq i$, such that both (a) $\gamma, j, g, u \Vdash \phi_2$, and (b) for all k, $j < k \leq i$, we have $\gamma, k, g, u \Vdash \phi_1$.

- $\gamma, i, g, u \Vdash x$ iff $u = g(x)$.
- $\gamma, i, g, u \Vdash \downarrow x \,.\, \phi$ iff $\gamma, i, g[x \mapsto u], u \Vdash \phi$.
- $\gamma, i, g, u \Vdash @_x \, \phi$ iff $\gamma, i, g, g(x) \Vdash \phi$.

All the semantic rules are familiar except for the last three. The construct x tests if the crawling agent is located at the entity referenced by x (i.e., $u = g(x)$). The construct $\downarrow x \,.\, \phi$ extends the current assignment into $g[x \mapsto u]$, in which x is bound to the current location u, and then the subformula ϕ is evaluated with this extended assignment. The construct $@_x \, \phi$ repositions the crawling agent at $g(x)$, that is, the entity referenced by variable x, before ϕ is evaluated.

We introduce two shorthands. First, we write $\gamma, u, v \Vdash_x \phi$ to assert $\gamma, n, \{x \mapsto v\}, u \Vdash \phi$, where $n = |\gamma| - 1$. This shorthand asserts that a certain binary relationship exists between u and v within the trace γ, and the binary relation is specified as formula ϕ, where the variable x is used in ϕ to denote the second end point of the relationship. Second, we write $\gamma, u, v \Vdash \phi$ to assert $\gamma, u, v \Vdash_{\mathsf{target}} \phi$.

8.1.3 Derived forms and other syntactic conventions

The same set of derived forms from § 6.3 can be defined for the extended language.

With the introduction of entity variables and the binder, the notion of ***free variables*** will play a crucial role in the following discussion. A free variable is one that is not within the scope of a binder that defines that variable. Given a formula ϕ, $\mathsf{fv}(\phi)$ denotes the set of free variables of ϕ. Formal definition of $\mathsf{fv}(\cdot)$ is given in Appendix B.1.

A formula ϕ is ***regular*** iff it satisfies the following:

1. No two binders in ϕ bind the same variable. (That is, every binder defines a unique variable.)
2. For every subformula of ϕ that has the form $\downarrow x \,.\, \phi_1$, we have $x \in \mathsf{fv}(\phi_1)$. (i.e., no binder is redundant.)

By renaming variables and eliminating redundant binders, every irregular formula can be rewritten into a semantically equivalent regular formula, without increase in size. From now on, we assume that all formulas are regular. We call this the ***regularity convention***.

8.1.4 Policy Predicate

Given a formula ϕ for which $\mathsf{fv}(\phi) \subseteq \{\mathsf{target}\}$, we write $[\![\phi]\!]$ to denote the policy predicate induced by ϕ, such that $[\![\phi]\!](\gamma, u, v)$ returns 1 iff $\gamma, u, v \Vdash \phi$. As before, we adopt the convention that the *policy* component of a protection system maps event identifiers to formulas.

8.1.5 Examples

We illustrate the utility of hybridization with two examples. The first illustrates the use of hybrid constructs to "reposition" the crawler.

Example 12 (Group-centric Secure Information Sharing)**.** *We further develop Example 6. We formulate* $policy(\mathsf{create})$ *to enforce* ***GSIS-1****, part (c), from Example 3. Specifically, we guard the event* $\mathsf{create}(g, o)$ *as below.*

$$@_{\mathsf{target}} \neg \mathcal{O}\langle -\mathsf{create}\rangle \top \tag{10}$$

We also formulate $policy(\mathsf{read})$ *to enforce* ***GSIS-2*** *(Example 3). Specifically, event* $\mathsf{read}(u, o)$ *is guarded as below.*

$$\downarrow u \,.\, @_{\mathsf{target}}\, \mathcal{O}\langle -\mathsf{create}\rangle((\neg\langle -\mathsf{leave}\rangle u)\, \mathcal{S}(\langle -\mathsf{join}\rangle u)) \tag{11}$$

The advanced example below illustrates the use of hybrid constructs to name entities in the present, and verify past relationships among named entities.

Example 13 (Online Coauthoring Communities)**.** *Revisiting Example 8, we specify Policy 2 of Example 2 using* $\mathcal{HTR}$*. We begin by defining a utility formula* ϕ_o*.*

$$\phi_o = @_o \neg \mathcal{O}\langle -\mathsf{edit}\rangle \top \tag{12}$$

Given an entity variable o*, the formula* ϕ_o *checks that* o *has never been edited in the past. With this, the requirements of Policy 2 can be captured as below.*

$$\neg\Big(\downarrow s \,.\langle \mathsf{own}\rangle \downarrow o_1 \,.\, @_s\langle \mathsf{own}\rangle \downarrow o_2 \,.(\neg o_1 \wedge \phi_{o_1} \wedge \phi_{o_2})\Big) \tag{13}$$

The formula above begins by defining two entity variables, o_1 *and* o_2*, both are owned by the initiator* s *in the present (*$\downarrow s \,.\langle \mathsf{own}\rangle \downarrow o_1 \,.\, @_s\langle \mathsf{own}\rangle \downarrow o_2 \ldots$*.). It then checks that* o_1 *and* o_2 *are distinct (*$\neg o_1$*). The check* ϕ_o *is then applied to both* o_1 *and* o_2*, verifying that each of* o_1 *and* o_2 *has never been edited in the past. Finally, the above check corresponds to a violation of Policy 2, and thus the outermost negation (*$\neg$*) ensures absence of violations. To appropriate the above, we set* $policy(\mathsf{create})$ *to the conjunction of (2) and (13).*

8.2 Enforcement

The enforcement of $\mathcal{HTR}$ is non-trivial. With $\mathcal{TR}$, the evaluation of every subformula can be tracked by a binary relation in the internal state of the reference monitor. This is possible because there is only one propositional symbol. By hybridization, we introduce an arbitrary number of variables into subformulas. Specifically, a formula with k free variables represents a relation of arity $k + 1$. Therefore, the space requirement of the reference monitor grows exponentially with the number of variables involved.

Fortunately, one can constrain the expressiveness of $\mathcal{HTR}$ slightly, to obtain a language fragment that still retains much of the advantages of the hybrid language, but yields a reference monitor with the same space requirement of $\mathcal{TR}$. The trick, again, is relational abstraction, now applied in a meticulous manner. We will demonstrate once again the core thesis of § 7.6, that the reference monitor of the resulting policy language is comparable to a ReBAC system.

8.2.1 Overview of Strategy

A ***temporal formula*** is one in which the outermost operator is a temporal operator (i.e., $\mathcal{Y}\,\phi$ or $\phi_1\,\mathcal{S}\,\phi_2$). Note that the reference monitor needs to track additional state information (i.e., B) other than the current relational structure (i.e., G) because of the existence of temporal subformulas. If a formula contains no temporal operators, then a local model checking algorithm for hybrid logic, such as the one in [6], will suffice. It is when state transition occurs that the update algorithm needs to abstract the evaluation of temporal subformulas into binary relations. To apply this technique of relational abstraction, we impose syntactic restrictions on the hybrid policy language, such that every temporal subformula of a policy must have no more than one free variable. With this restriction, temporal subformulas again represent binary relations, which can be tracked by the reference monitor in a space-efficient manner. As before, this amounts to the reference monitor tracking a single relational structure, such that history information is recorded as edges.

The syntactic restriction still allows certain subformulas to express higher-arity relations, so long as they are not temporal formulas. Intuitively, the crawling agent is allowed to remember multiple locations through the use of variables so long as it is crawling within the relational structure of a single time point. When the agent moves through time, then it is only allowed to remember at most one variable. With these restrictions, the subformulas that represent binary relations are handled by a bottom-up update algorithm in the spirit of [25], and the higher-arity relations are handled by a top-down update algorithm [18], which trades time efficiency for space efficiency. Our novelty is in the careful scheduling of top-down and bottom-up processing by exploiting the the syntactic structure of the language fragment, as well as combining the techniques of [18] and [25].

8.2.2 A Language Fragment

A formula ϕ is ***proper*** iff both conditions below hold.

1. $|\mathsf{fv}(\phi')| \leq 1$ for every temporal subformula $\phi' \in \mathsf{sub}(\phi)$.
2. $\mathsf{fv}(\phi) \subseteq \{\mathsf{target}\}$.

Intuitively, a proper formula is such that every temporal subformula has at most one free variable, and the only free variable permitted in the formula itself is target. Let $\mathcal{HTR}^-$ be the fragment of $\mathcal{HTR}$ that contains only proper formulas.

Given a formula in $\mathcal{HTR}$, checking if it is proper (and thus belongs to $\mathcal{HTR}^-$) takes time linear to the size of the formula. This is achieved simply by calling $\mathsf{fv}(\cdot)$ once on the formula, annotating each subformula by its free variables, and checking that the 2 conditions above hold.

The following example illustrates that, despite requiring temporal subformulas to be proper, $\mathcal{HTR}^-$ is still expressive enough for specifying useful policies.

Example 14. *Formulas (10) and (11) from Example 12 are both proper. The same is true of formulas (12) and (13) from Example 13. See Example 15 for a justification of why formulas (12) and (13) are proper.*

8.2.3 Preprocessing via Annotation

Our claim is that $\mathcal{HTR}^-$ permits a space-efficient implementation of the reference monitor, because temporal formulas represent binary relations rather than high-arity relations. To help the reference monitor recognize subformulas that correspond to binary relations, so that the reference monitor can switch to an appropriate model checking strategy for each kind of subformulas (i.e., bottom-up for binary relations, and top-down for higher-arity relations), we introduce an annotation scheme for formulas in $\mathcal{HTR}$. Specifically, we introduce an annotation construct $\lceil\cdot\rceil_x$ for $\mathcal{HTR}$ formulas and their subformulas.

The grammar for an ***annotated formula*** φ is as below.

$$\begin{aligned}\varphi &::= \rho \mid \lceil\rho\rceil_x \\ \rho &::= \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle l\rangle\varphi \mid \langle -l\rangle\varphi \mid \mathcal{Y}\,\varphi \mid \varphi\,\mathcal{S}\,\varphi \mid \\ &\quad\ x \mid \downarrow x\,.\,\varphi \mid @_x\,\varphi\end{aligned}$$

An annotated formula is like an $\mathcal{HTR}$ formula, except that each subformula (including the formula itself) can be optionally decorated by an annotation construct. If an $\mathcal{HTR}$ formula ϕ is obtained from an annotated formula φ by removing all occurrences of the annotation construct, then ϕ is said to be the ***erasure*** of φ, and φ is said to be an ***annotation*** of ϕ.

The intended meaning of an annotated formula $\lceil\rho\rceil_x$ is that $\mathsf{fv}(\rho) \subseteq \{x\}$. Intuitively, $\lceil\cdot\rceil_x$ imposes an "upper bound" to the free variable set of ρ. We make this requirement formal as follows. First, we extend the definition of $\mathsf{fv}(\cdot)$ in Appendix B.1 to account for annotated formulas:

$$\mathsf{fv}(\lceil\rho\rceil_x) = \mathsf{fv}(\rho) \cup \{x\} \tag{14}$$

Second, φ is ***well annotated*** if every subformula (including φ itself) of the form $\lceil\rho\rceil_x$ is such that $\mathsf{fv}(\rho) \subseteq \{x\}$. That is, a well annotated formula is one in which the annotation constructs mean as intended.

A well annotated formula φ is ***properly annotated*** (or simply ***proper***) iff both of the following conditions hold.

1. Every temporal subformula ρ is decorated by an annotation construct, as in $\lceil\rho\rceil_x$, for some x.
2. φ is of the form $\lceil\rho\rceil_{\mathsf{target}}$.

A properly annotated formula φ is ***completely annotated*** (or simply ***complete***) iff every subformula ρ that satisfies $|\mathsf{fv}(\rho)| \leq 1$ is decorated by an annotation construct $\lceil\cdot\rceil_x$ for some appropriately selected x. Intuitively, a complete annotation is one in which every subformula that can potentially be annotated without violating the well-annotation requirement is indeed annotated. That is, a complete annotation conveys maximal information to the model checker.

Example 15 (Online Coauthoring Communities)**.** *A complete annotation of (12) in Example 13 is the following.*

$$\phi_o = \lceil @_o \lceil \neg \lceil \mathcal{O} \lceil \langle -\mathsf{edit}\rangle \lceil \top \rceil_o \rceil_o \rceil_o \rceil_o \rceil_o$$

A complete annotation of (13) in Example 13 is below.

$$\lceil \neg \lceil \downarrow s\,.\,\lceil \langle \mathsf{own}\rangle \lceil \downarrow o_1\,.\,@_s \lceil \langle \mathsf{own}\rangle \lceil \downarrow o_2\,.\,(\lceil \lceil \neg \lceil o_1 \rceil_{o_1} \rceil_{o_1} \wedge \phi_{o_1} \rceil_{o_1} \wedge \phi_{o_2}) \rceil_{o_1} \rceil_{o_1} \rceil_s \rceil_s \rceil_{\mathsf{target}} \rceil_{\mathsf{target}}$$

The following proposition offers alternative characterizations of the $\mathcal{HTR}^-$ language.

Proposition 16.

1. *An $\mathcal{HTR}$ formula ϕ is in $\mathcal{HTR}^-$ iff ϕ has a proper annotation.*
2. *The complete annotation of an $\mathcal{HTR}$ formula exists whenever the formula belongs to $\mathcal{HTR}^-$. When it exists, the complete annotation is unique.*
3. *Given an $\mathcal{HTR}^-$ formula ϕ, constructing its complete annotation takes only time linear to its size.*

See Appendix B.2 for a proof.

Given a completely annotated formula φ, a subformula of the form $\lceil \rho \rceil_x$ is called an α subformula: i.e., α subformulas represent binary relations. A subformula that is not decorated in φ is called a β subformula: i.e., β subformulas represent higher-arity relations. See Appendix B.3 for a formal treatment.

Hereafter we assume that the $policy$ component of the protection system maps event identifiers to completely annotated formulas. That is, the inputs to the update algorithm of reference monitor are completely annotated.

8.2.4 Reference Monitor Implementation

The reference monitor for $\mathcal{HTR}^-$ has the same basic structure as the one in § 7. Specifically, its internal state is a pair $\langle G, B \rangle$, similar to the pair discussed in § 7.4, and this state is updated using the steps described in § 7.5. The main differences are two. First, the dimensions of B are slightly different from before. Second, in Step 4 of the update procedure, rather than invoking Algorithm 1 for updating B, a different algorithm is invoked.

The component B of the monitor state is a three-dimensional boolean matrix. There is an entry $B[\alpha][u][v]$ for every α subformula of members of $\mathsf{ran}(policy)$ and for every $u, v \in V(G)$. The difference is that, in the previous monitor state the first dimension ranges among the *subformulas* of $\mathsf{ran}(policy)$, whereas for the present monitor state the first dimension ranges among the α *subformulas* of $\mathsf{ran}(policy)$.

The invariant of the reference monitor is the conjunction of (7) and the following.

$$B[\lceil \rho \rceil_x][u][v] = 1 \text{ iff } \gamma_t, u, v \Vdash_x erasure(\rho) \tag{15}$$

Here, $erasure(\rho)$ denotes the erasure of ρ. While the index v in (8) is the entity referenced by target, the index v in (15) is the entity referenced by the annotation variable x.

The updating of B is conducted by a pair of algorithms in Appendix C.1. The updating of entries $B[\varphi][\cdot][\cdot]$, for α subformulas φ, is handled by a bottom-up algorithm in the style of Algorithm 1. A top-down algorithm is invoked as a subroutine to process β subformulas.

8.2.5 Complexity

To articulate the time complexity of the update algorithms, we define the notion of ***local binder nesting degree (LBND)*** of an α formula. Specifically, the LBND of $\lceil \rho \rceil_x$ is the nesting degree of

binders ($\downarrow$) that appear within the top-level decoration $\lceil\cdot\rceil_x$ but not in nested decorations. For e.g., LBND of the formula below is 2.

$$\lceil(\downarrow x_2 \,.\, \downarrow x_3 \,.\, \ldots \lceil\downarrow x_5 \,.\, \ldots\rceil_{x_4}) \wedge (\downarrow x_6 \,.\, \ldots)\rceil_{x_1}$$

Theorem 17. *Let* $M = \sum_{\varphi \in \mathsf{ran}(policy)} |\varphi|$, $N = |V(G)|$, $E = \sum_{l \in \mathcal{R} \cup \mathcal{E}} |R_l(G)|$, *and* D *be the maximum LBND among all members of* $\mathsf{ran}(policy)$. *The update algorithm in the Appendix runs in time* $O(MN^{D+1}(N+E))$. *The space usage is the size of* G *plus* $O(MN^2)$ *bits overhead for matrix* B.

So long as we limit D to a constant, the time complexity is polynomial. To put this into perspective, the formulas in Example 13 has a maximum LBND of 1 (see also Example 15). See Appendix C.2 for a proof of this theorem.

9 Stateless Representation and Inter-group Relationship

In dialogue with [23, 21, 22], we discuss how the policy language can be further extended to facilitate stateless specification of communities (§ 9.1), and how collaborating groups can be modeled in our scheme (§ 9.2).

9.1 Stateless Versus Stateful Representations

9.1.1 Two Styles of Representation

In a stateful representation of a community, the social state is significant. It contains either immutable relationships that provide background information to the reference monitor, or evolving edges that record significant events. In a stateless representation, policies are articulated only in terms of the temporal properties of communication events. The relationship edges in the social state are irrelevant. Previous work [22] argues that stateless representations are superior because they are purely declarative. The specification of a stateless community does not consider how communication events should be tracked by edges in the social states. A stateful reference monitor implementation is then crafted manually. Formal methods are applied to ensure that the two capture the same behaviour. In the following, we extend our policy language to support stateless specification of communities. Coupled with our reference monitor technology, the conversion of the stateless specification to a stateful enforcement mechanism becomes fully automatic, and the correctness of the latter is guaranteed.

A community schema $\langle \mathcal{V}, \mathcal{R}, \mathcal{E}, \rightarrow \rangle$ is ***stateless*** iff $\mathcal{R} = \emptyset$. Otherwise it is ***stateful***. Certain protection system can be specified via either a stateful or stateless community schema. In a stateful representation, the policies will refer to both event and relationship edges, while the policies of a stateless representation refers only to event edges. A community is ***stationary*** if the transition relation $\rightarrow$ is the identity mapping. A stateless community is necessarily stationary. Example 6 is stateless and thus stationary; Example 5 is stateful but not stationary; Example 4 is stateful as well as stationary.

9.1.2 Extension of Policy Language

We introduce a novel construct into the syntax of $\mathcal{HTR}$.

$$\phi ::= \ldots \mid \langle\langle x \,.\, \phi \rangle\rangle \phi$$

where $x \in \mathsf{Var} \setminus \{\mathsf{target}\}$ is an entity variable. In addition, a formula $\langle\langle x \,.\, \phi_1 \rangle\rangle \phi_2$ is considered syntactically well formed only if $\mathsf{fv}(\phi_1) \subseteq \{x\}$ and $x \notin \mathsf{fv}(\phi_2)$[3]. The semantics of the new construct is given below[4].

- $\gamma, i, g, u \Vdash \langle\langle x \,.\, \phi_1 \rangle\rangle \phi_2$ iff there exists $u' \in V(G_i)$ such that $\gamma, i, \{x \mapsto u'\}, u \Vdash \phi_1$ and $\gamma, i, g, u' \Vdash \phi_2$.

Intuitively, $\langle\langle x \,.\, \phi_1 \rangle\rangle$ behaves in a way similar to $\langle l \rangle$. With one free variable, ϕ_1 represents a binary relation, say R_{ϕ_1}. Formula $\langle\langle x \,.\, \phi_1 \rangle\rangle \phi_2$ looks for an R_{ϕ_1}-neighbour that satisfies ϕ_2.

Example 18 (Online Coauthoring Communities)**.** *Formula (13) refers to the relation identifier* own*. According to Example 5, an* own*-edge from s to o is simply a record that the event* $\mathsf{create}(s, o)$ *occurred in the past. That is, the binary relation R_{own} in the social state is nothing but a stateful representation of the formula $\mathcal{O}\langle\mathsf{create}\rangle x$. Therefore, (13) can be reformulated by replacing every occurrence of $\langle\mathsf{own}\rangle$ with $\langle\langle x \,.\, \mathcal{O}\langle\mathsf{create}\rangle x \rangle\rangle$, yielding the following.*

$$\neg\Big(\downarrow s \,.\langle\langle x \,.\, \alpha_x \rangle\rangle \downarrow o_1 \,.\, @_s \langle\langle x \,.\, \alpha_x \rangle\rangle \downarrow o_2 \,.(\neg o_1 \wedge \phi_{o_1} \wedge \phi_{o_2})\Big)$$

Here, the macro α_x expands to $\mathcal{O}\langle\mathsf{create}\rangle x$. The result is that the own *edges are no longer needed. The set $\mathcal{R}$ can be set to $\emptyset$, and the transition relation $\rightarrow$ be set to the identity mapping. The introduction of $\langle\langle \cdot \,.\, \cdot \rangle\rangle$ enables specifying the community in a stateless manner.*

Example 19 (Group-centric Secure Information Sharing)**.** *Continuing with Example 6, we specify* $policy(\mathsf{read})$ *in order to enforce* ***GSIS-3*** *from Example 3. To that end, we guard event* $\mathsf{read}(u, o)$ *by the formula below.*

$$\langle\langle g \,.\, (\neg\langle\mathsf{leave}\rangle g)\, \mathcal{S}(\langle\mathsf{join}\rangle g) \rangle\rangle\, \mathcal{O}\langle\mathsf{create}\rangle\mathsf{target} \tag{16}$$

Intuitively, the binary relation specified by $(\neg\langle\mathsf{leave}\rangle g)\, \mathcal{S}(\langle\mathsf{join}\rangle g)$ indicates that the user is currently a member of g. Note that (11) and (16) specify two fundamentally different policies: (11) demands membership at the time of object creation, while (16) demands membership at the time of access.

To facilitate enforcement, we extend the notions of proper formulas and proper annotations to account for the new construct. We write $\mathcal{HTR}^\sharp$ to refer to the language obtained by extending $\mathcal{HTR}$ to incorporate the $\langle\langle \cdot \,.\, \cdot \rangle\rangle$ construct, and including only proper formulas.

[3]The definition of $\mathsf{fv}(\cdot)$ in Appendix B.1 is extended by $\mathsf{fv}(\langle\langle x \,.\, \phi_1 \rangle\rangle \phi_2) = \mathsf{fv}(\phi_2)$.

[4]The $\langle\langle \cdot \,.\, \cdot \rangle\rangle$ construct is essentially a constrained version of the existential quantifier ($\exists$) in full hybrid logic [1]. We eschew adopting the existential quantifier because of its significant impact to the efficiency of the update algorithm. The $\langle\langle \cdot \,.\, \cdot \rangle\rangle$ construct is a compromise that extends expressiveness without severe impact to efficiency.

9.1.3 Reference Monitor Implementation

The new construct $\langle\!\langle x \,.\, \phi_1 \rangle\!\rangle \phi_2$ is handled in a way analogous to $\langle l \rangle \phi$. The difference is that the binary relation captured by $B[\phi_1][\cdot][\cdot]$ is consulted instead of R_l. See Appendix D.1 for details.

Note that the reference monitor is the stateful enforcement mechanism of the stateless communities in Examples 18 and 19. No manual engineering of reference monitor is needed, and correctness is guaranteed.

Theorem 20. *The time complexity of the update algorithm in Appendix D.1 is $O(MN^{D+3})$ (i.e., replacing the term $(N + E)$ in Theorem 17 by N^2). The space complexity remains the same.*

A proof of this theorem is given in Appendix D.2.

9.2 Inter-group Relationships

Krishnan *et al.* [23, 21] deal with groups that are isolated from one another. That is, either the groups are not related in anyway, or their relationships do no affect authorization decisions. This is not always true. **GSIS-4** of Example 3 is an example of inter-group relationships. The following example provides first evidence that $\mathcal{HTR}^\sharp$ can be used for modelling such relationships.

Example 21 (Group-centric Secure Information Sharing)**.** *We slightly modify the set up of Example 6, such that $\mathcal{R} = \{\mathsf{dominate}\}$. Specifically, a* dominate*-edge from group g_1 to g_2 indicates that g_1 dominates g_2, and so membership in g_1 implies membership in g_2. We assume that R_{dominate} encodes a partial ordering (i.e., reflexive, antisymmetric, and transitive). We further assume that transitions do not alter the dominance relation (i.e., stationary). Then formula (16) can be revised as follows to support **GSIS-4**.*

$$\langle\!\langle g \,.\, (\neg\langle \mathsf{leave} \rangle g)\, \mathcal{S}(\langle \mathsf{join} \rangle g) \rangle\!\rangle \langle \mathsf{dominate} \rangle\, \mathcal{O} \langle \mathsf{create} \rangle \mathsf{target}$$

10 Conclusion and Future Work

We proposed an access control model for supporting collaborative communities that are founded on social contracts made up of both relationship- and history-based elements. Through the development of a series of policy languages and their reference monitors, we demonstrated the thesis that, although CSC offers the convenience of specification via both relationship- and history-based policy elements, the underlying enforcement mechanism can be rendered purely relationship based. This space efficiency is achieved by relational abstraction, in which the past dealings between two parties are abstracted into edges in a relational structure.

Future work includes the formal characterization of expressiveness: What policies can be enforced via relational abstraction? What policies can be specified in a purely stateless manner? Another direction is to explore the optimization of the update algorithm for the reference monitor: If the transition relation only makes local changes to the social state, then can we bound the updates made to the reference monitor state? A third direction is the development of technologies for facilitating policy engineering. This includes the devising of graphical formalisms for editing policies, capturing reusable social contract patterns at a higher level of abstraction, and designing policy composition frameworks for combining canned policy modules.

References

[1] Carlos Areces and Balder ten Cate. Hybrid logics. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, chapter 14, pages 821–868. Elsevier, 2007.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT, 2008.

[3] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *Proc. of VMCAI'04*, volume 2937 of *LNCS*, 2004.

[4] D. Basin, F. Klaedtke, and S. Müller. Policy monitoring in first-order temporal logic. In *Proc. of CAV'10*, volume 6174 of *LNCS*, 2010.

[5] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge, 2001.

[6] G. Bruns, P. W. L. Fong, I. Siahaan, and M. Huth. Relationship-based access control: Its expression and enforcement through hybrid logic. In *Proc. of CODASPY'12*, San Antonio, TX, USA, Feb. 2012.

[7] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thurainsingham. A semantic web based framework for social network access control. In *Proc. of SACMAT'09*, Stresa, Italy, June 2009.

[8] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisinghaim. Semantic web-based social network access control. *Computers and Security*, 30(2–3):108–115, 2011.

[9] B. Carminati, E. Ferrari, and A. Perego. Enforcing access control in Web-based social networks. *ACM TISSEC*, 13(1), 2009.

[10] Barbara Carminati and Elena Ferrari. Enforcing relationships privacy through collaborative access control in web-based social networks. In *Proceedings of CollaborateCom'09*, November 2009.

[11] Y. Cheng, J. Park, and R. Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *Proc. of the 4th IEEE Int. Conf. on Information Privacy, Security, Risk & Trust (PASSAT'12)*, Amsterdam, Netherlands, September 2012.

[12] Y. Cheng, J. Park, and R. Sandhu. A user-to-user relationship-based access control model for online social networks. In *Proceedings of DBSec'12*, 2012.

[13] Jan Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM TODS*, 20(2), 1995.

[14] Úlfar Erlingsson and Fred B. Schneider. IRM enforcement of Java stack inspection. In *Proc. of the 2000 IEEE Symp. on Security and Privacy (S&P'00)*, pages 246–255, Berkeley, CA, USA, May 2000.

[15] Philip W. L. Fong. Access control by tracking shallow execution history. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy (S&P'04)*, pages 43–55, Berkeley, CA, USA, May 2004.

[16] Philip W. L. Fong. Relationship-based access control: Protection model and policy language. In *Proc. of CODASPY'11*, San Antonio, TX, USA, Feb. 2011.

[17] Philip W. L. Fong and Ida Siahaan. Relationship- based access control policies and their policy languages. In *Proc. of SACMAT'11*, Innsbruck, Austria, June 2011.

[18] M. Franceschet and M. de Rijke. Model checking hybrid logics. *J. Applied Logic*, pages 279–304, 2006.

[19] Carrie E. Gates. Access Control Requirements for Web 2.0 Security and Privacy. In *IEEE W2SP'07*, Oakland, CA, USA, 2007.

[20] K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *ACM TOPLAS*, 28(1), 2006.

[21] R. Krishnan, J. Niu, R. Sandhu, and W. H. Winsborough. Group-centric secure information sharing models for isolated groups. *ACM TISSEC*, 14(3), 2011.

[22] R. Krishnan and R. Sandhu. Authorization policy specification and enforcement for group-centric secure information sharing. In *Proc. of the 7th Int. Conf. on Info. Sys. Security (ICISS'11)*, volume 7093 of *LNCS*. Springer, 2011.

[23] Ram Krishnan, Ravi Sandhu, Jianwei Niu, and William Winsborough. Foundations for group-centric secure information sharing models. In *Proceedings of SACMAT'09*, Stresa, Italy, June 2009.

[24] K. Krukow, M. Nielsen, and V. Sassone. A framework for concrete reputation-systems with applications to history-based access control. In *Proceedings of CCS'05*, Alexandria, VA, USA, November 2005.

[25] K. Krukow, M. Nielsen, and V. Sassone. A logical framework for history-based access control and reputation systems. *J. Computer Security*, 16(1):63–101, 2008.

[26] M. H. Lessnoff. *Social Contract*. Macmillan, 1986.

[27] J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of non-safety policies. *ACM TISSEC*, 12(3), 2009.

[28] J. Park, R. Sandhu, and Y. Cheng. ACON: Activity-centric access control for social computing. In *Proc. of the 5th Int. Conf. on Availability, Reliability & Security (ARES'11)*, Vienna, Austria, August 2011.

[29] J. Park, R. Sandhu, and Y. Cheng. A user-activity-centric framework for access control in online social networks. *IEEE Internet Computing*, 15(5), 2011.

[30] F. B. Schneider. Enforceable security policies. *ACM TISSEC*, 3(1), 2000.

A Correctness and Time Complexity of Algorithm 1

A.1 Correctness: Proof of Theorem 10

Proof. Note the order in which entries of B are updated by Algorithm 1. The subformulas of the members of $\mathsf{ran}(policy)$ are enumerated in an admissible order. For each subformula ϕ, all the entries of the submatrix $B[\phi][\cdot][\cdot]$ are updated before the algorithm moves on to the next subformula. We argue that this order ensures that the updates are correct.

Case $\top$, target: The update is obviously correct.

Case $\neg\phi_1$, $\phi_1 \vee \phi_2$, $\langle l\rangle\phi_1$, $\langle -l\rangle\phi_1$: The entries of B appearing on the right-hand sides of lines 12, 14, 16 and 18 are supposed to be the evaluations of subformulas ϕ_1 and ϕ_2 at time point $t+1$. Thus, correctness hinges on the fact that those entries have *already* been updated. If ϕ_1 or ϕ_2 is a $\mathcal{Y}$ formula, then **AO-1** guarantees that its evaluations have already been updated. Otherwise, **AO-2** ensures that its evaluations have already been updated.

Case $\mathcal{Y}\,\phi_1$: According to the expansion law (5), the entry $B[\phi_1][u][v]$ appearing on the right-hand side of line 20 is supposed to be the evaluation of ϕ_1 at time t. Contrary to the previous case, this time correctness hinges on the fact that $B[\phi_1][u][v]$ has *not yet* been updated, and thus carries the evaluation of ϕ_1 at time point t. If ϕ_1 is a $\mathcal{Y}$ formula, then **AO-1** ensures that $B[\phi_1][u][v]$ has not been updated yet. Otherwise, ϕ_1 is not a $\mathcal{Y}$ formula, and **AO-2** guarantees that $B[\phi_1][u][v]$ has not been updated yet.

Case $\phi_1\,\mathcal{S}\,\phi_2$: Three entries appear on the right-hand side of line 22: $B[\phi_2][u][v]$, $B[\phi][u][v]$ and $B[\phi_1][u][v]$. According to the expansion law (6), the first and the third are supposed to be evaluations of subformulas ϕ_1 and ϕ_2 at time point $t+1$, while the second, $B[\phi][u][v]$, is the evaluation of ϕ at time point t. That the evaluations of the subformulas ϕ_1 and ϕ_2 have already been updated can be established using the same argument as in the case of $\neg\phi_1$. As for $B[\phi][u][v]$, this is exactly the entry that line 22 is attempting to update. Thus when the right-hand side is being evaluated, the entry holds the evaluation of ϕ at time point t.

□

A.2 Time Complexity: Proof of Theorem 11

Proof. Explanation of the term $(N+E)$ is in order for the time complexity $O(MN(N+E))$. In the innermost for loop of Algorithm 1, the switch statement is executed for N times. Most of the cases execute in constant time, except for the cases of $\langle l\rangle\phi_1$ and $\langle -l\rangle\phi_1$ in lines 16 and 18 respectively. The total amount of time needed to execute them are proportional to the total in- and out-degrees of the vertices, that is, E. Therefore, the innermost loop takes $(N+E)$ time to execute. □

B The Hybrid Languages $\mathcal{HTR}$ and $\mathcal{HTR}^-$

B.1 Free Variables

The formal definition of $\mathsf{fv}(\phi)$, the set of free variables for an $\mathcal{HTR}$ formula ϕ, is given below.

$$\begin{aligned}
\mathsf{fv}(\top) &= \emptyset & \mathsf{fv}(\phi_1 \vee \phi_2) &= \mathsf{fv}(\phi_1) \cup \mathsf{fv}(\phi_2) \\
\mathsf{fv}(\neg\phi) &= \mathsf{fv}(\phi) & \mathsf{fv}(\phi_1 \,\mathcal{S}\, \phi_2) &= \mathsf{fv}(\phi_1) \cup \mathsf{fv}(\phi_2) \\
\mathsf{fv}(\langle l\rangle\phi) &= \mathsf{fv}(\phi) & \mathsf{fv}(x) &= \{x\} \\
\mathsf{fv}(\langle -l\rangle\phi) &= \mathsf{fv}(\phi) & \mathsf{fv}(\downarrow x\,.\,\phi) &= \mathsf{fv}(\phi) \setminus \{x\} \\
\mathsf{fv}(\mathcal{Y}\,\phi) &= \mathsf{fv}(\phi) & \mathsf{fv}(@_x\,\phi) &= \mathsf{fv}(\phi) \cup \{x\}
\end{aligned}$$

B.2 Characterization of $\mathcal{HTR}^-$: Proof of Proposition 16

Lemma 22. *If φ is an annotation of $\mathcal{HTR}$ formula ϕ, then $\mathsf{fv}(\phi) \subseteq \mathsf{fv}(\varphi)$.*

Proof. By (14) and a straightforward structural induction. □

Lemma 23. *If $\lceil\rho\rceil_x$ is a well annotation of $\mathcal{HTR}$ formula ϕ, then $\mathsf{fv}(\phi) \subseteq \{x\}$.*

Proof. By Lemma 22 and a straightforward structural induction. □

Lemma 24. *If a $\mathcal{HTR}$ formula ϕ has a proper annotation, then $\phi \in \mathcal{HTR}^-$.*

Proof. By Lemmas 22 and 23. □

The following (deterministic) algorithm constructs a complete annotation for a given $\mathcal{HTR}^-$ formula ϕ.

1. Compute $\mathsf{fv}(\cdot)$ for every subformula of ϕ.
2. Annotate the root of the Abstract Syntax Tree (AST) of ϕ by $\lceil\cdot\rceil_{\mathsf{target}}$.
3. Visit the AST nodes of ϕ in a top-down fashion. For each node ϕ', do the following:
   (a) If $\mathsf{fv}(\phi') = \{x\}$, annotate ϕ' by the annotation $\lceil\cdot\rceil_x$.
   (b) If $\mathsf{fv}(\phi') = \emptyset$, annotate ϕ' by the annotation $\lceil\cdot\rceil_x$, where x is the annotation of the closest ancestor.
   (c) Otherwise, $|\mathsf{fv}(\phi')| > 1$, and ϕ' is left unannotated.

The following is a proof of Proposition 16.

Proof. Lemma 24 establishes the "if" direction of Statement 1.

The above deterministic construction guarantees the existence and uniqueness of complete annotation for $\mathcal{HTR}^-$ (i.e., Statement 2). It also guarantees the "only if" direction of Statement 1.

Observe that the construction can be implemented in time linear to the size of the input formula, thereby guaranteeing Statement 3. □

B.3 Structures of Complete Annotations

Completely annotated formulas have certain structures.

Observation 25. *The complete annotation φ of an $\mathcal{HTR}^-$ formula satisfies the following grammar:*

$$\begin{aligned}
\varphi &::= \alpha \\
\psi &::= \alpha \mid \beta \\
\alpha &::= \lceil \top \rceil_x \mid \lceil \neg\alpha \rceil_x \mid \lceil \alpha \vee \alpha \rceil_x \mid \lceil \langle l \rangle \alpha \rceil_x \mid \lceil \langle -l \rangle \alpha \rceil_x \mid \\
&\quad \lceil \mathcal{Y}\, \alpha \rceil_x \mid \lceil \alpha\, \mathcal{S}\, \alpha \rceil_x \mid \lceil x \rceil_x \mid \lceil \downarrow x\,.\,\psi \rceil_x \mid \lceil @_x\, \alpha \rceil_x \\
\beta &::= \neg\psi \mid \psi \vee \psi \mid \langle l \rangle \psi \mid \langle -l \rangle \psi \mid \downarrow x\,.\,\psi \mid @_x\, \psi
\end{aligned}$$

If we construct the abstract syntax tree (AST) of a completed annotated formula φ according to the grammar above, then we call an AST node that corresponds to the non-terminal α an α subformula, and one that corresponds to the non-terminal β a β subformula. A number of important observations can be made regarding the gramatical structure of complete annotations. These observations will be used in Appendix C.1 in the design of the update algorithms.

Observation 26.

1. *A complete annotation is always an α formula.*
2. *Immediate subformulas of an α formula are almost always α formulas, except for the case of $\lceil \downarrow x\,.\,\psi \rceil_y$, where the immediate subformula ψ can be either an α or a β formula. If we take the regularity convention into account, then ψ can only be a β formula.*
3. *The immediate subformulas of a β formula can be either α or β formulas.*
4. *The $\mathcal{HTR}$ constructs $\top$ and x are always decorated by an annotation construct. This means that β formulas cannot be formed solely by nesting β subformulas. A β formula is always "bottomed out" by α formulas.*

The syntactic structure of a complete annotation φ provides the recursive structure of the model checker. As we shall see, the α subformulas are processed bottom-up, while the β subformulas are processed top-down.

Let $\mathsf{sub}_\alpha(\varphi)$ and $\mathsf{sub}_\beta(\varphi)$ denote respectively the set of α subformulas and the set of β subformulas of a completely annotated formula φ. We write $\mathsf{sub}_\psi(\varphi)$ for $\mathsf{sub}_\alpha(\varphi) \cup \mathsf{sub}_\beta(\varphi)$. As before, we overload $\mathsf{sub}_\alpha(\cdot)$, $\mathsf{sub}_\beta(\cdot)$ and $\mathsf{sub}_\psi(\cdot)$ to take a *set* of completely annotated formulas as an argument: i.e., if Φ is a set of completely annotated formulas, then $\mathsf{sub}_\alpha(\Phi) = \cup_{\varphi \in \Phi} \mathsf{sub}_\alpha(\varphi)$.

Given a set Φ of α formulas, an enumeration of Φ's members is in an ***admissible order*** if the enumeration is generated in accordance to the following.

α-AO-1 First, list the α formulas of the form $\lceil \mathcal{Y}\, \alpha \rceil_x$. Among them, list superformulas before subformulas.

α-AO-2 Second, list the remaining α formulas, with subformulas listed before superformulas.

Algorithm 2: $Update^{\Uparrow}(G, B)$

in : $G \in \mathcal{G}[\mathcal{V}, \mathcal{R} \cup \mathcal{E}]$
in out: $B : \mathsf{sub}_\alpha(\mathsf{ran}(policy)) \times V(G) \times V(G) \to \mathbb{B}$
local : $C : \mathsf{sub}_\psi(\mathsf{ran}(policy)) \times V(G) \to \mathbb{B}$
for $\alpha \in \mathsf{sub}_\alpha(\mathsf{ran}(policy))$, *in an admissible order,* **do**
  for $u \in V(G)$ **do**
    for $v \in V(G)$ **do**
      switch α **do**
        case $\lceil \top \rceil_x$
          $B[\alpha][u][v] \leftarrow 1$;
        case $\lceil \neg\alpha_1 \rceil_x$
          $B[\alpha][u][v] \leftarrow \neg B[\alpha_1][u][v]$;
        case $\lceil \alpha_1 \vee \alpha_2 \rceil_x$
          $B[\alpha][u][v] \leftarrow B[\alpha_1][u][v] \vee B[\alpha_2][u][v]$;
        case $\lceil \langle l \rangle \alpha_1 \rceil_x$
          $B[\alpha][u][v] \leftarrow \bigvee_{u' \in \mathsf{succ}^l_G(u)} B[\alpha_1][u'][v]$;
        case $\lceil \langle -l \rangle \alpha_1 \rceil_x$
          $B[\alpha][u][v] \leftarrow \bigvee_{u' \in \mathsf{pred}^l_G(u)} B[\alpha_1][u'][v]$;
        case $\lceil \mathcal{Y}\, \alpha_1 \rceil_x$
          $B[\alpha][u][v] \leftarrow B[\alpha_1][u][v]$;
        case $\lceil \alpha_1\, \mathcal{S}\, \alpha_2 \rceil_x$
          $B[\alpha][u][v] \leftarrow B[\alpha_2][u][v] \vee (B[\alpha][u][v] \wedge B[\alpha_1][u][v])$;
        case $\lceil x \rceil_x$
          $B[\alpha][u][v] \leftarrow (u = v)$;
        case $\lceil \downarrow x\,.\,\psi_1 \rceil_y$
          $Update^{\Downarrow}(G, B, C, \psi_1, \{x \mapsto u, y \mapsto v\})$;
          $B[\alpha][u][v] \leftarrow C[\psi_1][u]$;
        case $\lceil @_x\, \alpha_1 \rceil_x$
          $B[\alpha][u][v] \leftarrow B[\alpha_1][v][v]$;

C Reference Monitor for $\mathcal{HTR}^-$

C.1 Updating Algorithms

In Step 4 of the update procedure, Algorithm 2 is invoked to update B. Algorithm 2 in turn invokes Algorithm 3.

Algorithm 2, $Update^{\Uparrow}$, is a bottom-up model checking algorithm for α subformulas. Algorithm 3, $Update^{\Downarrow}$, is a top-down model checking algorithm invoked by $Update^{\Uparrow}$ to evaluate β subformulas. The recursion pattern of the two algorithms closely follow the grammar in Observation 25.

Algorithm 2. The bottom-up model checker $Update^{\Uparrow}$ has a structure almost identical to that of $Update$. It enumerates α subformulas in some admissible order, as defined by α**-AO-1** and α**-AO-**

Algorithm 3: $Update^{\Downarrow}(G, B, C, \psi, g)$

in : $G \in \mathcal{G}[\mathcal{V}, \mathcal{R} \cup \mathcal{E}]$
in : $B : \mathsf{sub}_{\alpha}(\mathsf{ran}(policy)) \times V(G) \times V(G) \to \mathbb{B}$
in out: $C : \mathsf{sub}_{\psi}(\mathsf{ran}(policy)) \times V(G) \to \mathbb{B}$
in : $\psi \in \mathsf{sub}_{\psi}(\mathsf{ran}(policy))$
in : $g : \mathsf{Var} \rightharpoonup V(G)$
switch ψ **do**
  case $\neg\psi_1$
    $Update^{\Downarrow}(G, B, C, \psi_1, g)$;
    for $u \in V(G)$ **do**
      $C[\psi][u] \leftarrow \neg C[\psi_1][u]$;
  case $\psi_1 \vee \psi_2$
    $Update^{\Downarrow}(G, B, C, \psi_1, g)$;
    $Update^{\Downarrow}(G, B, C, \psi_2, g)$;
    for $u \in V(G)$ **do**
      $C[\psi][u] \leftarrow C[\psi_1][u] \vee C[\psi_2][u]$;
  case $\langle l \rangle \psi_1$
    $Update^{\Downarrow}(G, B, C, \psi_1, g)$;
    for $u \in V(G)$ **do**
      $C[\psi][u] \leftarrow \bigvee_{u' \in \mathsf{succ}^{l}_{G}(u)} C[\psi_1][u']$;
  case $\langle -l \rangle \psi_1$
    $Update^{\Downarrow}(G, B, C, \psi_1, g)$;
    for $u \in V(G)$ **do**
      $C[\psi][u] \leftarrow \bigvee_{u' \in \mathsf{pred}^{l}_{G}(u)} C[\psi_1][u']$;
  case $@_x \, \psi_1$
    $Update^{\Downarrow}(G, B, C, \psi_1, g)$;
    for $u \in V(G)$ **do**
      $C[\psi][u] \leftarrow C[\psi_1][g(x)]$;
  case $\downarrow x \, . \, \psi_1$
    for $u \in V(G)$ **do**
      $Update^{\Downarrow}(G, B, C, \psi_1, g[x \mapsto u])$;
      $C[\psi][u] \leftarrow C[\psi_1][u]$;
  case $\lceil \rho \rceil_x$
    for $u \in V(G)$ **do**
      $C[\psi][u] \leftarrow B[\psi][u][g(x)]$;

2 (line 4). For each subformula α, it updates the submatrix $B[\alpha][\cdot][\cdot]$. The updates of lines 9, 11, 13, 15, 17, 19 and 21 are identical to their counterparts in Algorithm 1. Line 23 functions in the same way as line 10 of Algorithm 1. Line 28 performs update in the case of $\lceil @_x\, \alpha_1 \rceil_x$. (Note that the x in $\lceil \cdot \rceil_x$ must be the same variable as the x in $@_x\, \alpha_1$.) Essentially the evaluation of $@_x\, \alpha_1$ at u is the same as the evaluation of α_1 at v, which is the entity referenced by x.

The genuinely complex case is that of $\lceil \downarrow x\, .\, \psi_1 \rceil_y$, the update of which begins on line 24. What is at stake is that B is only capable of differentiating the evaluations of an α formula $\lceil \rho \rceil_x$ at entity u for different bindings of x. The binder ($\downarrow$) now introduces an additional binding, causing B not to be able to capture the bindings of multiple variables. Consequently, the bottom-up model checking strategy no longer applies. The top-down updating algorithm (Algorithm 3) is thus invoked (line 25) to evaluate ψ_1 with an extended variable assignment (containing assignments for both x and y). The result of evaluation is deposited into the matrix C. Such results are then copied back into B (line 26).

Algorithm 3. The top-down algorithm $Update^{\Downarrow}$ evaluates a formula ψ in G at every $u \in V(G)$, with a variable assignment g that defines the free variables in ψ. On return, the evaluation results are stored in a boolean matrix C, such that $C[\psi][u]$ holds the evaluation of ψ at entity u.

The storage for C is allocated entirely in Algorithm 2 (line 3), and C is then passed into Algorithm 3.[5] The matrix B is passed as an *input* argument: i.e., it is not mutated.

The algorithm computes recursively. The base case is when ψ is an α formula. Admissible ordering guarantees that the evaluation of this α formula is already in B. Line 33 copies that evaluation from B to C. (Recall that, if ψ is of the form $\lceil \rho \rceil_x$, then $B[\psi][u][v]$ is the evaluation of ψ at u with the variable assignment $g = \{x \mapsto v\}$.) The inductive case is when ψ is a β formula. In this case, $Update^{\Downarrow}$ is called recursively to evaluate the subformulas of ψ, the results of which is used for evaluating ψ itself.

The cases of $\neg\psi_1$, $\psi_1 \vee \psi_2$, $\langle l \rangle \psi_1$, $\langle -l \rangle \psi_1$ and $@_x\, \psi_1$ are relatively straightforward. The case of $\downarrow x\, .\, \psi_1$, however, warrants further explanation. Specifically, when $Update^{\Downarrow}$ is recursively invoked to evaluate the subformula ψ_1 on line 29, the variable assignment is extended to $g[x \mapsto u]$, such that x points to u during the evaluation of ψ_1.

Correctness. The correctness of Algorithms 2 and 3 can be articulated and proved in a manner similar to Theorem 10. Note also that, if all the subformulas are α formulas, then $Update^{\Downarrow}$ will not be invoked at all, and the behaviour of $Update^{\Uparrow}$ will mirror that of $Update$.

C.2 Time Complexity: Proof of Theorem 17

Proof. Comparing the time complexities in Theorems 11 and 17, the difference is a factor of N^D. That difference comes from the exponential behavior of the top-down algorithm (Algorithm 3): i.e., the case of $\downarrow x\, .\, \psi_1$. Note that exponentiation stops when the algorithm encounters an α formula: i.e., the case of $\lceil \rho \rceil_x$. The maximum number of times $\downarrow$ occurs before this happens is exactly D. □

[5] One could have presented Algorithm 3 in such a way that the storage for C is allocated dynamically, on demand. Such a presentation reduces comprehensibility without saving anything in the asymptotic space complexity.

Algorithm 4: $Update^{\Uparrow}(G, B)$

```
1 ...
2 switch α do
3   ...
4   case ⌈⟨⟨x . α1⟩⟩α2⌉y
5     B[α][u][v] ← ∃u′ ∈ V(G) . B[α1][u][u′] ∧ B[α2][u′][v];
```

Algorithm 5: $Update^{\Downarrow}(G, B, C, \psi, g)$

```
1 ...
2 switch ψ do
3   ...
4   case ⟨⟨x . α1⟩⟩ψ2
5     Update⇓(G, B, C, ψ2, g);
6     for u ∈ V(G) do
7       C[ψ][u] ← ∃u′ ∈ V(G) . B[α1][u][u′] ∧ C[ψ2][u′];
```

D Reference Monitor for $\mathcal{HTR}^{\sharp}$

D.1 Updating Algorithms

To handle the new construct $\langle\langle x \,.\, \phi_1 \rangle\rangle \phi_2$, Algorithms 2 and 3 are revised by introducing an additional case for the construct, as shown in Algorithms 4 and 5. Observe that line 5 of Algorithm 4 is merely an adaptation of line 15 of Algorithm 2, and similarly line 7 of Algorithm 5 is an adaptation of line 19 of Algorithm 3.

D.2 Time Complexity: Proof of Theorem 20

Proof. The term $(N + E)$ must now be replaced by N^2 because the existential quantification in line 5 of Algorithm 4 can take $O(N)$ to verify. So the innermost loop takes $O(N^2)$ time whenever it is executed. □