ABSTRACT
The Relationship-Based Access Control (ReBAC) model was recently proposed as a general-purpose access control model. It supports the natural expression of parameterized roles, the composition of policies, and the delegation of trust. Fong proposed a policy language that is based on Modal Logic for expressing and composing ReBAC policies. A natural question is whether such a language is representationally complete, that is, whether the language is capable of expressing all ReBAC policies that one is interested in expressing.

In this work, we argue that the extensive use of what we call Relational Policies is what distinguishes ReBAC from traditional access control models. We show that Fong’s policy language is representationally incomplete in that certain previously studied Relational Policies are not expressible in the language. We introduce two extensions to the policy language of Fong, and prove that the extended policy language is representationally complete with respect to a well-defined subclass of Relational Policies.

Categories and Subject Descriptors
D.4.6 [Security and Protection]: Access Controls

General Terms
Security, Language, Theory

Keywords
Access control policies, modal logic, policy languages, relationship-based access control, social networks

1. INTRODUCTION
The advent of social computing introduces the world to a new paradigm of access control, in which interpersonal relationships are explicitly tracked by the protection system for the purpose of authorization. Gates coined the term Relationship-Based Access Control (ReBAC) to refer to this paradigm of access control [19]. In a typical ReBAC system, the protection state consists of a knowledge base of primitive relationships (e.g., friend) between individual users [7], and an access control policy (e.g., friend-of-friend) is expressed in terms of composite relationships induced by the primitive relationships (e.g., friend ◦ friend).

While previously proposed ReBAC systems are designed mostly for social computing applications [25, 8, 7, 37, 38, 18], a formal ReBAC model was proposed recently as a general-purpose access control model [16]. The model features poly-relational social networks with asymmetric relations, as well as a context-dependent authorization procedure that limits the scope of relationships to their applicable contexts. Fong articulates three benefits of the ReBAC paradigm in general, and of his ReBAC model in particular. First, formulating policies in terms of binary relationships between users supports the natural expression of authorization decisions that are based on the relative attributes of the accessor as perceived by the resource owner (e.g., trust, professional association, etc). Traditional access control systems base their authorization decisions on some unary predicates of the users (e.g., identities, roles, etc) that is defined by a central authority. The limitations of such an approach becomes evident as notions such as parameterized roles [32] (or role templates [21]) creep into RBAC systems: e.g., the parameterized role manager(john) signifies the role assumed by the manager of John. In many emerging application domains, such as social computing and Electronic Health Records Systems, in which the protection of user-contributed contents is prominent, accessibility depends not on the intrinsic attributes of the accessor, but on the relationship between the accessor and the owner (e.g., the professional relationship). ReBAC provides natural support for such situations. Second, relationship-based access control policies support a richer form of policy composition. With policies that are based on unary predicates, the only interesting form of policy composition would be boolean combinations (i.e., conjunction, disjunction and negation), which can be captured readily by, say, a role hierarchy. Binary relations, however, supports a much richer form of composition, including relational composition (friend ◦ friend) and transitive closure (friend+). Third, a composite binary relation supports delegation of trust in a natural way: e.g., by adopting the policy friend-of-friend, I am delegating to my friends to decide who...
can access.

Fong also proposed a policy language for expressing ReBAC policies [16]. The language is based on a basic modal logic, and it has been shown to be capable of expressing complex, composite relationships between an owner and an accessor, such as those found in an Electronic Health Records (EHR) system [4]. One natural question to ask is whether the policy language of Fong is representationally complete, that is, whether it is capable of expressing all the ReBAC policies we would like to express. In this work, we give a negative answer to the question, we extend the policy language to cover its shortcomings, and we formally characterize the expressive power of the extended policy language. In this last respect, this research has close affinity to enforceability research, as exemplified in [36, 29, 14, 23, 39, 30, 31], which formally characterize the structural relationships between the family of policies enforceable by a given enforcement mechanism and other naturally occurring policy families.

Our contributions are threefold:

1. A distinguished feature of ReBAC is its extensive use of what has come to be known as relational policies [1]. We demonstrate, in Section 2, that a number of relational policies previously studied in the context of Facebook-style Social Network Systems [18] cannot be expressed in the policy language of Fong.

2. We introduce, in Section 3, two new features into the ReBAC policy language, and demonstrate that the resulting language is capable of expressing the above-mentioned relational policies.

3. In Section 4, we formally characterize the expressiveness of the extended policy language by proving both an “upper bound” and a “lower bound” for the family of ReBAC policies expressible in the language. The upper-bound result shows that, every policy expressible in the extended policy language belongs to the family of owner-checkable policies. The lower-bound result identifies a natural subclass of relational policies, namely, finitary relational policies, that are provably expressible in the extended language: the extended language is representationally complete with respect to finitary relational policies.

2. INADEQUACY OF B

This section provides an overview of the background on which this work is based, and formally articulate the representational incompleteness of the basic policy language of Fong [16].

2.1 Notation

Given a binary relation \( R \subseteq X \times Y \) and individuals \( x \in X \) and \( y \in Y \), we write \( R(x, y) \) iff \( (x, y) \in R \). We write \( 2^X \) for the powerset of \( X \) (i.e., the set of all subsets of \( X \)), \( [X]^k \) for the set of all subsets of \( X \) that have a cardinality \( k \), and \( [X]^{\leq \omega} \) for the set of all finite subsets of \( X \). We write \( f : X \to Y \) whenever \( f \) is a function with a subset of \( X \) as its domain and \( Y \) as its co-domain. We write \( \emptyset \) for a function with an empty domain. Consider a function \( f : X \to Y \) and individuals \( x_0 \in X \) and \( y_0 \in Y \). We write \( f(x_0) \to y_0 \) to denote the function \( f' : X \to Y \) defined as follows: \( f'(x) = y_0 \) if \( x = x_0 \), but \( f'(x) = f(x) \) if \( x \neq x_0 \).

2.2 Social Networks and Relation Identifiers

We assume that an SNS defines a finite set \( I \) of relation identifiers. Each identifier denotes a type of relationships that is tracked by the system (e.g., parent-child, patient-physician, etc.). A typical member of \( I \) is denoted by \( i \).

A social network is essentially a directed graph with multiple kinds of edges. While individuals are represented by vertices, each kind of directed edges represents a distinct type of relationship between users. Formally, a social network \( G \) is a relational structure [5] of the form \( \langle V, \{ R_i \}_{i \in I}\rangle \), where:

- \( V \) is a finite set of vertices, each representing an individual in the social network.
- \( \{ R_i \}_{i \in I} \) is a family of binary relations. The binary relation \( R_i \subseteq V \times V \) specifies the pairs of individuals participating in relationship type \( i \).

We write \( V(G) \) and \( R_i(G) \) respectively for the \( V \) and \( R_i \) components of the social network \( G \).

Standard graph-theoretic concepts apply to social networks in an expected manner. We outline a few to fix thoughts. We write \( G \subseteq G' \) whenever \( V(G) \subseteq V(G') \) and \( R_i(G) \subseteq R_i(G') \) for every \( i \in I \). \( G \) is said to be a subgraph of \( G' \). Suppose \( U \subseteq V(G) \). We write \( G[U] \) to denote the social network \( \langle U, \{ R_i \}_{i \in I}\rangle \), where \( R'_i = R_i(G) \cap (U \times U) \). \( G[U] \) is said to be the subgraph \( G \) induced by \( U \).

Two social networks \( G, G' \) are isomorphic iff there is a bijection function \( \pi : V(G) \to V(G') \) such that \( (u, v) \in R_i(G) \) whenever \( (\pi(u), \pi(v)) \in R_i(G') \). In this case, we write \( G \cong G' \), and call \( \pi \) an isomorphism between \( G \) and \( G' \).

Given a social network \( G \), we define the acquaintance graph \( acq(G) \) to be the simple graph \( \langle V(G), E \rangle \), where \( E = \{ (u, v) \in V(G)^2 \mid \exists i \in I . (u, v) \in R_i(G) \} \). That is, an acquaintance graph shares the same vertex set as the social network, and there is an undirected edge between two distinct vertices in the acquaintance graph iff there is a typed, directed edge between the same pair of vertices in the social network. Note that, although loops may occur in the social network, the acquaintance graph is a simple graph, and thus it contains no loops. The purpose for defining the acquaintance graph is to reuse connectivity-related concepts from standard graph theory. Specifically, notions such as distance, connectedness, and components can be applied to a social network. For example, two vertices in a social network \( G \) are said to be connected if there is an undirected path between them in \( acq(G) \); the distance between two connected vertices in \( G \) is the length of the shortest path between them in \( acq(G) \); an (extended) neighbourhood of a vertex \( u \) in \( G \) is the set \( N_G(u) \) of vertices connected to \( u \) in \( acq(G) \); the component of \( G \) to which \( u \) belongs is the social network \( C_G(u) \), which is the set of all vertices in \( G \) that are reachable from \( u \) by a path in \( G \).
defined to be $G[N^G_v(u)]$. Lastly, we write $C_G(u; v)$ to denote the social network $G[N^G_v(u) \cup \{v\}]$, which is called an augmented component. The idea is to form the component of $G$ to which $u$ belongs, but with the augmentation of vertex $v$. If $u$ and $v$ are connected, then $C_G(u) = C_G(u; v)$. Otherwise, $C_G(u; v)$ contains $C_G(u)$ plus an additional, isolated vertex $v$.

Suppose $U$ is a countable set of user identifiers. We denote by $\mathcal{G}(U, I)$ the set of all finite social networks defined for user set $U$ and relation identifier set $I$. That is, $\mathcal{G}(U, I) = \{(V, \{R_i\}_{i \in I}) \mid V \in [U]^{<\omega}, R_i \subseteq V \times V\}$. $\mathcal{G}(U, I)$ contains all the social networks with a vertex set that is a finite subset of $U$. Since $I$ is finite, such a social network has only finitely many edges. Note that, although every member of $\mathcal{G}(U, I)$ is finite in size, the size of such a member is not bounded.

2.3 ReBAC Policies

In [16] a formal model of ReBAC is introduced. The state of a ReBAC system is a collection of social networks, as defined above. State transition involves the mutation of the social networks (e.g., addition and deletion of relationship edges). An access attempt involves an accessor requesting access to a resource owned by an owner. When an access attempt is attempted at a given state, the policy that controls the accessibility of the requested resource will be evaluated against a certain social network induced by the current state. The policy either allows or denies the request. In the following, we formally specify what a ReBAC policy is.

A ReBAC policy defines a desired binary relation between resource owners and resource accessors in the context of a given social network. Suppose we are given a fixed set $U$ of users and a fixed set $I$ of relation identifiers. A ReBAC policy is a family of binary relations over $U$, indexed by social networks from $\mathcal{G}(U, I)$. More specifically, a policy $P : \mathcal{G}(U, I) \rightarrow 2^{U \times U}$, such that $P(G) \subseteq V(G) \times V(G)$ for every social network $G \in \mathcal{G}(U, I)$. Intuitively, a policy $P$ specifies, for each social network $G$, a binary relation $P(G)$ that relates each resource owner to those accessors that the policy grants access. That is, $(u, v) \in P(G)$ iff owner $u$ grants access to accessor $v$ in the context of social network $G$. Note that $P(G)$ contains only finitely many pairs because $G$ has only finitely many vertices. Using the notation defined in Section 2.1, we write $P(G)(u, v)$ whenever $(u, v) \in P(G)$.

In a typical ReBAC system, only a certain vocabulary of ReBAC policies is supported. For example, in Facebook, the policy vocabulary includes ReBAC policies such as me, friend, friend-of-friend, everyone, etc. A resource owner will adopt a supported policy from the policy vocabulary to protect a resource. The following are examples of ReBAC policies. They were previously proposed in [18] in the study of Facebook-style Social Network Systems (FSNSs). They have been adapted here as examples of ReBAC policies.

Example 1 (Distance). Suppose the set of relational identifiers is $I = \{friend\}$. Suppose further that the relation identified by friend is symmetric (but not necessarily reflexive). We call such social networks pseudo-Facebook social networks (the friend relation of a genuine Facebook-style social network is reflexive). Define the policy $\text{dist}_k$, such that $\text{dist}_k(G)(u, v)$ iff $u$ and $v$ is at a distance $k$ or less from one another. This policy is a generalization of the friend-of-friend policy in Facebook. Specifically, the distance between two vertices is taken as a measure of the strength of trust between two individuals. Having the trust level exceeding the threshold $k$ results in access granting.

Example 2 (Common Friends). Consider again pseudo-Facebook social networks. Define the policy $\text{cf}_k$ such that $\text{cf}_k(G)(u, v)$ iff (a) $u = v$, or (b) $v$ is a neighbour of $u$, or (c) there are $k$ (or more) common neighbours between $u$ and $v$. Intuitively, access is granted to a stranger if $k$ friends of the owner witness to the trustworthiness of this stranger. This policy is another generalization of the friend-of-friend policy in Facebook. Specifically, the number of common neighbours is taken as a measure of the strength of trust between two strangers. Having the trust level exceeding the threshold $k$ results in access granting.

Example 3 (Clique). Consider yet again pseudo-Facebook social networks. Define the policy $\text{clique}_k$ such that $\text{clique}_k(G)(u, v)$ iff (a) $u = v$, or (b) $u$ and $v$ belong to a clique of size $k$ (or more) in $acq(G)$. Intuitively, access is granted if the accessor and the owner belong to a close-knit community of size $k$ (or more). This policy is a generalization of the friend policy in Facebook. Specifically, the size of the largest common clique to which two neighbouring individuals belong is taken as a measure of the strength of trust between them. Having a trust level exceeding the threshold $k$ results in access granting. Other notions of close-knit communities [9], such as clans, plexes and cores, can also be modelled in a similar way.

2.4 Relational Policies

In an unpublished manuscript [1], Anwar et al. identify a family of policies, called relational policies, the use of which distinguishes ReBAC from traditional access control paradigms. Intuitively, a relational policy is special in two ways. First, a relational policy does not base authorization decisions on the identities of the owner or the accessor. Instead, only the topological structure the social network is analyzed for authorization. Second, a relational policy specifies how the owner and the accessor shall be related in the social network (e.g., the owner and the accessor are at a distance no more than $k$), rather than individual properties of the owner or the accessor (e.g., the accessor has a vertex degree no less than $k$). These two characteristics are captured in the following definitions, which we generalized from those in [1].

Definition 4. A policy $P$ is topology-based iff, for every pair of isomorphic social networks $G, G' \in \mathcal{G}(U, I)$ with isomorphism $\pi : V(G) \rightarrow V(G')$, we have $(u, v) \in P(G)$ whenever $(\pi(u), \pi(v)) \in P(G')$.

In short, a topology-based policy does not take the identities of the owner and the accessor into consideration. Instead,
such a policy consumes only topological information of the social network.

The next definition characterizes policies that specify how the owner and the accessor are related to one another.

**Definition 5.** A policy \( P \) is **local** iff, given \( G \subseteq G' \) and \( u,v \in V(G) \) for which \( (u,v) \in P(G) \cap P(G') \), there exists \( i \in I \) such that \( R_i(G') \setminus R_i(G) \) contains a pair \((u',v')\) for which each of \( u' \) and \( v' \) is connected to each of \( u \) and \( v \) in \( G' \).

The intuition of the definition is the following. Suppose \( P \) is a local policy. Suppose further that the authorization decision of \( P \) is altered for a pair of individuals \((u,v)\) after one adds vertices and edges into the social network \( G \) to obtain social network \( G' \). Then \( G' \) must have received a new edge \((u',v')\) of some relationship type \( i \), such that each of \( u' \) and \( v' \) is connected to each of \( u \) and \( v \). In short, adding vertices or edges outside of the shared component of the owner and the accessor never alters the authorization decision of a local policy. Put more succinctly, the ends of every edge that has an influence on authorization decisions must connect both the owner and the accessor. That is, such an edge must contribute to the “connectedness” between the owner and the accessor. In this way, we capture the intuitive requirement that the policy must specify how the owner and the accessor are related to one another, rather than specifying individual graph properties of the owner or the accessor.

**Definition 6.** A policy is **relational** iff it is both topology based and local.

Note that the three example policies (i.e., distance, common friends and cliques) are relational policies.

### 2.5 The ReBAC Policy Language \( \mathcal{B} \)

Fong proposed a policy language for expressing ReBAC policies [16]. We call his language \( \mathcal{B} \) to differentiate it from the extended language \( \mathcal{E} \) to be presented in the sequel. The language \( \mathcal{B} \) is a basic modal logic [5]. Each formula in \( \mathcal{B} \) expresses a ReBAC policy: i.e., a desired relationship between two individuals in a social network. The syntax of a \( \mathcal{B} \) formula, which is based on a set \( I \) of relation identifiers, is given below:

\[
\phi, \psi ::= \top \mid a \mid \neg \phi \mid \phi \land \psi \mid \langle i \rangle \phi \mid \langle \neg i \rangle \phi
\]

where \( i \in I \) is a relation identifier.

A policy (i.e., a formula) \( \phi \) is interpreted in the context of a social network \( G \in \mathcal{G}(V,I) \) and individuals \( u,v \in V(G) \). Specifically, the semantics of \( \mathcal{B} \) is given in a satisfiability relation \( G,u,v \models_\mathcal{B} \phi \), which asserts that, in social network \( G \), owner \( u \) and accessor \( v \) possess the relationship prescribed by policy \( \phi \).

Intuitively, \( \top \) is a relationship satisfiable by any pair of individuals; \( a \) asserts that the accessor is the owner; \( \langle i \rangle \phi \) asserts that the owner has an \( i \)-neighbour that is related to the accessor in the manner specified by \( \phi \); \( \langle \neg i \rangle \phi \) asserts that the owner is the \( i \)-neighbour of some individual that is related to the accessor in the manner specified by \( \phi \). The difference between \( \langle i \rangle \) and \( \langle \neg i \rangle \) is that the former traverses along the direction of an edge, while the latter traverses against the direction of an edge. Standard derived forms are defined:

\[

\top = \neg \top \quad [i] \phi = \neg \langle \neg i \rangle \neg \phi \\
\phi \lor \psi = \neg \langle \neg \phi \land \neg \psi \rangle \\
\neg \langle \neg i \rangle \phi = \neg \langle \neg \langle \neg i \rangle \phi \rangle

\]

Intuitively, \([i] \phi \) asserts that every \( i \)-neighbour of the owner is related to the accessor in the manner specified by \( \phi \).

A ReBAC policy \( P \) is **definable in \( \mathcal{B} \)** iff there is a \( \mathcal{B} \) formula \( \phi \) such that \( P(G) \models_\mathcal{B} \phi \) whenever \( G,u,v \models_\mathcal{B} \phi \). It has been demonstrated that many complex relational policies are definable in \( \mathcal{B} \). Specifically, Fong [16] was able to use \( \mathcal{B} \) to capture all the trust delegation policies identified in the Electronic Health Records case study developed by Becker and Sewell [4]. We illustrate the usage of \( \mathcal{B} \) with a few examples. Consider social networks with relation identifiers \( I = \{ \text{parent}, \text{child}, \text{spouse}, \text{sibling} \} \), which signify respectively the child-parent relation, its inverse, the reflexive spouse-spouse relation, and the reflexive sibling-sibling relation. The following policies are definable in \( \mathcal{B} \):

- **“Grant access to grand parents.”** The \( \mathcal{B} \) formula that expresses this policy is \( \langle \text{parent} \rangle \langle \text{parent} \rangle \).

- **“Grant access to a sibling who is not married.”** The \( \mathcal{B} \) formula that expresses this policy is:

\[
\langle \text{sibling} \rangle (a \land \neg [\text{spouse}] \bot)
\]

- **“Grant access if accessor is the only child of the owner.”** A formula to express the policy is:

\[
\langle \text{child} \rangle a \land \neg \langle \text{child} \rangle a
\]

### 2.6 Limitations of \( \mathcal{B} \)

\( \mathcal{B} \) is not without limitations. It turns out that some relational policies are not definable in \( \mathcal{B} \).

**Example 7.** Assume the setting of Example 2. The policy \( c_k \), for every \( k > 1 \), is not definable in \( \mathcal{B} \). The reason is that the models for these policies are bisimilar to \( c_k \), and thus indistinguishable by a formula of \( \mathcal{B} \) [5]. For instance,
consider $c_f$. A naive attempt to express $c_f$ in $B$ yields the following formula:
\[ a \lor (\text{friend}) a \lor ((\text{friend}) (\text{friend}) a \land (\text{friend}) (\text{friend}) a) \]
The intention is that the disjuncts “a” and “(friend) a” respectively express requirements (a) and (b) in Example 2, while the third disjunct expresses requirement (c). Supposedly, each of the two conjuncts “(friend) (friend) a” identifies a common friend of the owner and the accessor. Unfortunately, this formula does not express $c_f$. Consider the scenario in which the owner and the accessor share exactly one common neighbour. Since boolean conjunction is idempotent, the formula above will be satisfied even though $c_f$ should fail.

In the formula above, the original intention is that the two conjuncts specify two disjoint sets of intermediaries (i.e., two distinct common friends). Unfortunately, boolean conjunction fails to capture this requirement. This illustrates the need for an ReBAC policy language to provide a non-idempotent version of conjunction for specifying disjoint sets of intermediaries.

**Example 8.** The policy clique$_k$ is not definable in $B$, for every $k > 2$. For instance, consider clique$_3$. A naive attempt to express the policy is the following formula:
\[ a \lor (\neg a \land (\text{friend}) a \land (\text{friend}) a) \]
The intention is that the disjunct “a” expresses requirement (a) in Example 3, while the second disjunct expresses requirement (b). In the second disjunct, we assert that the owner is not the accessor (\neg a), the accessor is a friend of the owner ((friend) a), and the accessor is a friend of a friend of the owner ((friend) (friend) a). Yet, the following two social networks are not distinguishable by $B$ formulas.

\[ v \]
\[ u \]

Specifically, the sub-formula “(friend) (\neg a \land (\text{friend}) a)” fails to ensure that the owner (a) and the intermediary (\bullet) are two distinct vertices.

The above example illustrates an important shortcoming of the policy language $B$, namely, its lacking a vertex identification mechanism: i.e., the ability to name vertices and subsequently test if they are revisited.

### 3. THE REBAC POLICY LANGUAGE $E$

In the previous section, we identified two limitations of the policy language $B$, namely, disjoint intermediaries and vertex identification. In the following, we will look at how the two features can be incorporated into a ReBAC policy language. The result of this exploration is an extended ReBAC policy language we call $E$.

**Syntax.** As before, $E$ is defined in the context of a countable set $\mathcal{U}$ of vertices and a countable set $\mathcal{I}$ of relation identifiers.

We also assume that there is a countably infinite set $\mathcal{P}$ of propositional symbols. We write $p$ and $q$ for typical members of $\mathcal{P}$. We also assume that $\mathcal{P}$ contains a distinguished member $a$.

The syntax of $E$ is given below:
\[ \phi, \psi ::= \top | p \land \phi \land \psi | (i) \phi | (\neg i) \phi | @p.\phi | \phi \odot \psi \]
where $p \in \mathcal{P}$ is a propositional symbol, and $i \in \mathcal{I}$ is a relation identifier. Intuitively, $@p.\phi$ introduces a propositional symbol for identifying the owner. Specifically, the propositional symbol $p$ can be used inside $\phi$ to test if a vertex is the owner. The formula $\phi \odot \psi$ holds whenever the relationships prescribed by $\phi$ and $\psi$ both hold between the accessor and the owner, but the set of intermediary vertices used for establishing these two relationships are disjoint. A more precise definition of the semantics of these new constructs is given below.

The notion of free and bound occurrences of propositional symbols can be defined in a standard way. For example, in the formula $(p \land @q.(i) q)$, the propositional symbol $p$ occurs free but $q$ occurs bound. A formula is a-closed if no propositional symbol other than $a$ occurs free in the formula. Otherwise the formula is a-open.

**Semantics.** Given a social network $G$, a function $\Sigma : \mathcal{P} \rightarrow V(G)$ is called a binding environment$^5$. A binding environment interprets a propositional symbol as a vertex.

The semantics of $E$ is defined via the satisfaction relation $G, u, v \models_E \phi$, which is specified in terms of an auxiliary relation $G, \Sigma, u \models \phi$, where $\Sigma$ is a binding environment.

- $G, \Sigma, u \models \top$.
- $G, \Sigma, u \models p$ iff $\Sigma(p) = u$.
- $G, \Sigma, u \models \neg \phi$ iff it is not the case that $G, \Sigma, u \models \phi$.
- $G, \Sigma, u \models \phi \land \psi$ iff both $G, \Sigma, u \models \phi$ and $G, \Sigma, u \models \psi$.
- $G, \Sigma, u \models (i) \phi$ iff there exists $u' \in V$ such that $(u, u') \in R_i$ and $G, \Sigma, u' \models \phi$.
- $G, \Sigma, u \models @p.\phi$ iff there exists $u' \in V$ such that $(u', u) \in R_p$ and $G, \Sigma, u' \models \phi$.
- $G, \Sigma, u \models @p.\phi$ iff the following holds:

Let $v$ be $\Sigma(a)$. There exists two subsets $V_1$ and $V_2$ of $V(G)$, such that $V(G) = V_1 \cup V_2$, $V_1 \cap V_2 = \{u, v\}$, and both $G[V_1], \Sigma, u \models \phi$ and $G[V_2], \Sigma, u \models \phi$ hold.

Lastly, $G, u, v \models_E \phi$ iff $G, @[a \rightarrow v], u \models \phi$.

$^5$Note that our binding environment plays the role of a valuation or labelling function in the standard literature of modal logic [5], in which a valuation is a function with signature $\mathcal{P} \rightarrow 2^{V(G)}$, and a labelling function is a function with signature $V(G) \rightarrow 2^\mathcal{P}$. In our logic, a propositional symbol is interpreted as a single vertex. We therefore adapt the definition to ease presentation.
We begin our discussion with the family of topology-based policies that are definable in \( G \). Specifically, \( G, \Sigma, u \models \phi \iff \text{vertex identification.} \)

Examples. The following examples illustrate how the new features of \( \mathcal{E} \) address the needs for disjoint intermediaries and vertex identification.

**Example 9.** The policy \( c_0 \) is definable in \( \mathcal{E} \), via the following formula:

\[
a \lor \{\text{friend}\} a \lor (\{\text{friend}\} a) \otimes (\{\text{friend}\} (\{\text{friend}\} a))
\]

**Example 10.** The policy \( \text{clique}_3 \) is definable in \( \mathcal{E} \), via the following formula:

\[
a \lor (\neg a \land \{\text{friend}\} a \land \oplus p. (\{\text{friend}\}) (\neg p \land \neg a \land \{\text{friend}\} a))
\]

4. **Expressiveness of \( \mathcal{E} \)**

It has been pointed out in Section 2.4 that ReBAC is distinguished by its extensive use of access control policies are relational; i.e., authorization decisions consume only topological information of the social network (topology-based), and express how the owner and the accessor are related in the social network (local). In this section, we will examine the expressiveness of \( \mathcal{E} \) by identifying what family of relational policies are definable in \( \mathcal{E} \). We will first establish, in Section 4.1, an “upper bound” of the policies definable in \( \mathcal{E} \); i.e., identifying a natural family of ReBAC policies to which every \( \mathcal{E} \)-definable policy belongs. We will then establish, in Section 4.2, a “lower bound” of the expressiveness of \( \mathcal{E} \). That is, we will identify a natural subclass of relational policies that are definable in \( \mathcal{E} \). In other words, we establish the representational completeness of \( \mathcal{E} \) with respect to that subclass of relational policies. We conclude the section with a structural analysis of the hierarchy of policy families identified along the way (Section 4.3).

4.1 **Owner-checkable policies**

We begin our discussion with the family of topology-based policies that can be enforced by an agent that traverses a neighbourhood of the owner.

**Derived Forms.** We define the derived forms \( \bot, \phi \lor \psi, [i] \phi \) and \([-i] \phi \) as before. In addition, we introduce the following derived form:

\[
\phi \oplus \psi = \neg (\neg \phi \land \neg \psi)
\]

The connective \( \oplus \) is the dual of \( \otimes \). Specifically, \( G, \Sigma, u \models \phi \oplus \psi \) iff the following holds:

Let \( v \) be \( \Sigma(a) \). For every two subsets \( V_1 \) and \( V_2 \) of \( V(G) \) such that \( V(G) = V_1 \cup V_2 \), \( V_1 \cap V_2 = \{u, v\} \), either \( G[V_1], \Sigma, u \models \phi \) or \( G[V_2], \Sigma, u \models \phi \) holds.

In other words, \( G, \Sigma, u \models \phi \oplus \psi \) whenever, for every subset \( V \) of \( V(G) \) such that \( \{u, \Sigma(a)\} \subseteq V \), either \( G[V], \Sigma, u \models \phi \) or \( G[V], \Sigma, u \models \phi \) holds.

**Definability.** A policy \( P \) is **definable in \( \mathcal{E} \)** iff there is an \( a \)-closed formula \( \phi \) in \( \mathcal{E} \) such that \( P(G)(u, v) \) whenever \( G, u, v \models \phi \).

**Examples.** The following examples illustrate how the new features of \( \mathcal{E} \) address the needs for disjoint intermediaries and vertex identification.

**Example 9.** The policy \( c_0 \) is definable in \( \mathcal{E} \), via the following formula:

\[
a \lor \{\text{friend}\} a \lor (\{\text{friend}\} a) \otimes (\{\text{friend}\} (\{\text{friend}\} a))
\]

**Example 10.** The policy \( \text{clique}_3 \) is definable in \( \mathcal{E} \), via the following formula:

\[
a \lor (\neg a \land \{\text{friend}\} a \land \oplus p. (\{\text{friend}\}) (\neg p \land \neg a \land \{\text{friend}\} a))
\]

4. **Expressiveness of \( \mathcal{E} \)**

It has been pointed out in Section 2.4 that ReBAC is distinguished by its extensive use of access control policies are relational; i.e., authorization decisions consume only topological information of the social network (topology-based), and express how the owner and the accessor are related in the social network (local). In this section, we will examine the expressiveness of \( \mathcal{E} \) by identifying what family of relational policies are definable in \( \mathcal{E} \). We will first establish, in Section 4.1, an “upper bound” of the policies definable in \( \mathcal{E} \); i.e., identifying a natural family of ReBAC policies to which every \( \mathcal{E} \)-definable policy belongs. We will then establish, in Section 4.2, a “lower bound” of the expressiveness of \( \mathcal{E} \). That is, we will identify a natural subclass of relational policies that are definable in \( \mathcal{E} \). In other words, we establish the representational completeness of \( \mathcal{E} \) with respect to that subclass of relational policies. We conclude the section with a structural analysis of the hierarchy of policy families identified along the way (Section 4.3).

4.1 **Owner-checkable policies**

We begin our discussion with the family of topology-based policies that can be enforced by an agent that traverses a neighbourhood of the owner.
social network to provide an existential proof of compliance. The following definition formalizes the idea.

**Definition 14.** A policy $P$ is **monotonic** iff $G \subseteq G'$ implies $P(G) \subseteq P(G')$.

That is, adding relationships into a social network never reduces accessibility, and removing relationships never expands accessibility. Such a policy makes authorization decisions by verifying the presence of relationships rather than the absence of relationships. The notion of monotonicity was originally proposed in [18] for Facebook-style Social Network Systems. Here we generalize the notion for ReBAC policies.

Adopting policies that are exclusively monotonic enables a decentralized implementation of ReBAC in the style of trust management systems [6, 40, 27, 26]. Specifically, an accessor who seeks authorization may present to the reference monitor a fragment of the social network (e.g., a collection of certificates of relationships) as a proof of compliance. ReBAC, however, differs from trust management systems in its extensive use of relational policies, which do not rely on the identity of an individual for making authorization decisions. We are therefore interested in policies that are both monotonic and topology based.

**Definition 15.** A policy $P$ is **positive** iff it is both topology based and monotonic.

We review here a characterization of positive policies. Our characterization is phrased in terms of birooted graphs.

**Definition 16.** A birooted graph $G_{(u,v)}$ is a triple $(G, u, v)\text{ such that } G$ is a social network and $u,v \in V(G)$. The vertices $u$ and $v$ are the roots of $G_{(u,v)}$, and they need not be distinct. Specifically $u$ is the owner root, and $v$ is the accessor root. We write $B(S,I) = \{G_{(u,v)} \mid G \in \mathcal{G}(S,I), u,v \in V(G)\}$ to denote the set of all birooted graphs based on vertex set $S$. $G'_{(u',v')}$ is a (birooted) subgraph of $G_{(u,v)}$, written as $G'_{(u,v)} \subseteq G_{(u,v)}$, iff $G' \subseteq G$. (Note the matching roots.) We say that $G_{(u,v)}$ and $G'_{(u',v')}$ are isomorphic iff there exists an isomorphism $\pi : V(G) \rightarrow V(G')$ between social networks $G$ and $G'$, such that $\pi(u) = u'$ and $\pi(v) = v'$. In this case we write $G_{(u,v)} \cong G'_{(u',v')}$. We also write $G''_{(u'',v'')} \subseteq G_{(u,v)}$ whenever there is a birooted graph $G''_{(u'',v'')}$ such that $G''_{(u'',v'')} \cong G''_{(u,v)}$ and $G''_{(u,v)} \subseteq G_{(u,v)}$.

We are now ready to state the characterization theorem for positive policies.

**Definition 17.** The policy positively induced by a set $B$ of birooted graphs is the policy $P_B^+$ for which $P_B^+(G)_{(u,v)}$ iff $\exists G'_{(u',v')} \in B, G'_{(u',v')} \subseteq G_{(u,v)}$.

Intuitively, the birooted graph set $B$ specifies topological “patterns” that must exist between an owner and an accessor in the social network in order for access to be granted by the policy. It was proven in [1] that every positive policy can be characterized by a set of birooted graph patterns. The result was originally established for Facebook-style Social Network Systems. Here, we adapt the result to the more general context of ReBAC.

**Theorem 18.** Every positive policy is positively induced by a set of birooted graphs. The minimal set of birooted graphs to positively induce a given policy $P$ is defined to be the set $B$ for which there exists no proper subset of $B$ that also positively induces $P$. This minimal set does not contain a pair of distinct birooted graphs $G_{(u,v)}$ and $G'_{(u',v')}$ such that $G_{(u,v)} \subseteq G'_{(u',v')}$. Such a minimal set always exists, and is unique up to birooted graph isomorphism.

Intuitively, the theorem states that every positive policy can be uniquely characterized by a smallest set of birooted graph patterns.

We are particularly interested in a special kind of positive policies.

**Definition 19.** A positive policy $P$ is **finitary** iff the minimal set of birooted graphs to positively induce $P$ is a finite set.

Finitary policies can be characterized by a finite number of birooted graphs. While positive policies can be evaluated by checking for the presence of relationships, the finitary requirement ensures that only a bounded number of such relationships need to be enumerated. This requirement is computationally significant in two ways. Firstly, this translates to a bound in the search effort for a centralized implementation of the authorization procedure, for the search tree now has bounded depth. Secondly, in a distributed implementation of ReBAC, this constraint implies that a proof of compliance consists of a bounded number of certificates.

We are now ready to state the main theorem of this work.

**Theorem 20.** Every finitary OC policy is definable in $E$.

The theorem provides a “lower bound” of what can be expressed in $E$. The proof of this theorem can be found in Appendix A.

Recall that our core interest is in expressing relational policies, rather than OC policies. The following corollary, which follows directly from proposition 12, identifies a subclass of relational policies definable in $E$.

**Corollary 21.** Every finitary relational policy is definable in $E$.

Note that all the three example policies (i.e., distance, common friends, and clique) are finitary relational policies.

We can further strengthen the above result. Suppose $P_1$ and $P_2$ are definable in $E$ via the formulas $\phi_1$ and $\phi_2$. Consider
The term Relationship-Based Access Control (ReBAC) was first coined by Gates [19] when she articulated the need for an access control paradigm that is based on interpersonal relationships. A number of ReBAC models and/or systems have been proposed for social computing applications [25, 8, 7, 37, 38, 18]. The targets of study include the distributed evaluation of trust [8], the employment of semantic web technology to encode social networks and ReBAC policies [7], the automatic generation of ReBAC policies [37], multiple ownership of resources [38], the generalization of Facebook-style Social Network Systems [18], and the characterization of relational policies [1]. Fong advocates the employment of ReBAC in application domains outside of social computing [16], and demonstrates the utility and feasibility of doing so by the specification of a general-purpose ReBAC model and a ReBAC policy language. He points out that the various constructs in his ReBAC model arise naturally out of the generalization of foundational concepts such as roles, role hierarchy, sessions and constraints in Role-Based Access Control (RBAC) [33, 13]. His work also raises the question of representational completeness, which is the focus of this work. The core contribution of this work is (a) demonstrating that Fong’s policy language fails to encode a number of relational policies previously studied in [18], (b) incorporating the support of disjoint intermediaries and vertex identification into a ReBAC policy language that is based on modal logic, and (c) providing both an upper bound and a lower bound to the family of policies expressible in the extended policy language.

The question of representational completeness is related to enforceability research [36, 29, 14, 23, 39, 30, 31]. The goal of enforceability research is to establish the structural relationships between naturally occurring policy families on one hand (e.g., safety policies), and the family of policies enforceable by a specific kind of enforcement mechanisms (e.g., program monitors that have access only the execution history). Rather than relating policy families to enforcement mechanisms, the present work establishes upper bound and lower bound for the family of policies expressible in a policy language.

Policies such as $c_{k}$ (for $k > 2$) are not definable in B. The reason is that the models for these policies are bisimilar to $c_{1}$, and thus indistinguishable by a formula of B [5]. Therefore, in E a new connective $\otimes$ is introduced to require disjoint partitioning of the social network during a satisfiability check (except for the accessor and the owner). This approach is similar to manifold roles by Li et al. in RT$^{T}$ [27]. The credential using $\otimes$ in RT$^{T}$ is a union of the sets where the intersection of such sets is empty. The concept of threshold structure by Li et al. in Delegation logic [26] is also similar to $\otimes$. In a threshold structure, $k$ out of $N$ principals are required, where each of the $k$ is unique.

6. FUTURE WORK

Some extensions to the ReBAC model have been suggested in [16]. We discuss in the following future directions that are specific to ReBAC policy languages.

A natural future work is to improve the precision of the characterization of policies definable in E. Specifically, there is a gap between the “upper bound” and “lower bound” presented in Section 4. As demonstrated in Section 4.3, the inclusion of FR in OC is proper. It is desirable if one can close the gap, and provide a precise identification of the family of policies definable in E. We anticipate that such an effort can benefit from the application of techniques from Finite Model Theory [28], which has long been applied to understand the expressiveness of database query languages [11, 10, 34].

One of the motivations for designing a policy language for ReBAC policies is to facilitate policy analysis. It has been shown [15] that proper static analysis can be applied to the

---

**Figure 1:** The hierarchy of policy families identified in this work: TB: topology-based policies; OC: owner-checkable policies; Rel: relational policies; Pos: positive policies; Fin: finitary policies; FR: finitary relational policies.

The policy $P$ defined such that $P(G) = P_{1}(G) \cap P_{2}(G)$. That is, $P$ grants access whenever both $P_{1}$ and $P_{2}$ grant access. It is easy to see that $P$ is also definable in E, via the formula $\phi_{1} \land \phi_{2}$. The same can be said about other boolean combinations (disjunction and negation).

**Corollary 22.** The family of policies definable in E is closed under boolean combinations. Consequently, boolean combinations of finitary relational policies are definable in E. Note that such boolean combinations of finitary relational policies are still relational policies.

This corollary gives us an easy extension of the definability result in Corollary 21.

4.3 Discussion

Figure 1 depicts the hierarchy of policy families identified in this section. Note that every inclusion in Figure 1 is proper. To demonstrate this, the following list states a member policy for each set difference.

- $\text{TB}\setminus(\text{Pos} \cup \text{OC})$: The social network contains no more than $k$ vertices (including the owner and the accessor).
- $\text{Pos}\setminus\text{Fin}$: The owner and the accessor are connected.
- $\text{Fin}\setminus\text{FR}$: The accessor has at least $k$ neighbours.
- $\text{OC}\setminus\text{Rel}$: The owner has no more than $k$ neighbours.
- $\text{Rel}\setminus\text{FR}$: The owner and the accessor are not connected.

5. RELATED WORK

The term Relationship-Based Access Control (ReBAC) was first coined by Gates [19] when she articulated the need for an access control paradigm that is based on interpersonal relationships. A number of ReBAC models and/or systems have been proposed for social computing applications [25, 8, 7, 37, 38, 18]. The targets of study include the distributed evaluation of trust [8], the employment of semantic web technology to encode social networks and ReBAC policies [7], the automatic generation of ReBAC policies [37], multiple ownership of resources [38], the generalization of Facebook-style Social Network Systems [18], and the characterization of relational policies [1]. Fong advocates the employment of ReBAC in application domains outside of social computing [16], and demonstrates the utility and feasibility of doing so by the specification of a general-purpose ReBAC model and a ReBAC policy language. He points out that the various constructs in his ReBAC model arise naturally out of the generalization of foundational concepts such as roles, role hierarchy, sessions and constraints in Role-Based Access Control (RBAC) [33, 13]. His work also raises the question of representational completeness, which is the focus of this work. The core contribution of this work is (a) demonstrating that Fong’s policy language fails to encode a number of relational policies previously studied in [18], (b) incorporating the support of disjoint intermediaries and vertex identification into a ReBAC policy language that is based on modal logic, and (c) providing both an upper bound and a lower bound to the family of policies expressible in the extended policy language.

The question of representational completeness is related to enforceability research [36, 29, 14, 23, 39, 30, 31]. The goal of enforceability research is to establish the structural relationships between naturally occurring policy families on one hand (e.g., safety policies), and the family of policies enforceable by a specific kind of enforcement mechanisms (e.g., program monitors that have access only the execution history). Rather than relating policy families to enforcement mechanisms, the present work establishes upper bound and lower bound for the family of policies expressible in a policy language.

Policies such as $c_{k}$ (for $k > 2$) are not definable in B. The reason is that the models for these policies are bisimilar to $c_{1}$, and thus indistinguishable by a formula of B [5]. Therefore, in E a new connective $\otimes$ is introduced to require disjoint partitioning of the social network during a satisfiability check (except for the accessor and the owner). This approach is similar to manifold roles by Li et al. in RT$^{T}$ [27]. The credential using $\otimes$ in RT$^{T}$ is a union of the sets where the intersection of such sets is empty. The concept of threshold structure by Li et al. in Delegation logic [26] is also similar to $\otimes$. In a threshold structure, $k$ out of $N$ principals are required, where each of the $k$ is unique.

6. FUTURE WORK

Some extensions to the ReBAC model have been suggested in [16]. We discuss in the following future directions that are specific to ReBAC policy languages.

A natural future work is to improve the precision of the characterization of policies definable in E. Specifically, there is a gap between the “upper bound” and “lower bound” presented in Section 4. As demonstrated in Section 4.3, the inclusion of FR in OC is proper. It is desirable if one can close the gap, and provide a precise identification of the family of policies definable in E. We anticipate that such an effort can benefit from the application of techniques from Finite Model Theory [28], which has long been applied to understand the expressiveness of database query languages [11, 10, 34].

One of the motivations for designing a policy language for ReBAC policies is to facilitate policy analysis. It has been shown [15] that proper static analysis can be applied to the
configuration of Facebook-style Social Network Systems to ensure the absence of Sybil attacks. It is interesting to explore if this or other policy analyses can be translated into an equivalent syntactic analysis for policies written in a ReBAC policy language.

Both this work and [16] assumes the ReBAC system tracks user relationships explicitly, and the relations are in turn identified by a fixed vocabulary of relation identifiers. This assumption has made the design of rich policy languages possible. In some applications, however, the presence of a relationship edge as well as its typing are induced by complex external factors. For example, consider a network consisting of users and devices such as mobile phones, in which the presence of a relationship is the result of the states of the mobile phones (e.g., battery level) and their relative proximity. Rational modelling of such external dependence would allow us to reuse ReBAC and our policy languages even in these application domains.

Another future direction is to combine the strengths of both RBAC and ReBAC, thereby allowing the access control system to reason about both the relationships between individuals, and the hierarchical organization of user roles. One possibility is to employ Description Logic (DL) technology [2] as the basis of both the policy language and the authorization engine. Highly related to modal logics [35], DL is a mature technology, with language families of various tractability, and the support of efficient reasoners (e.g., RACER [22] and FaCT [24]). A previous work on applying DL to access control is that of ReBAC [20, 41], in which DL is employed to construct an access control model akin to the Domain-Type Enforcement (DTE) model [3]. Further work is needed to enable the rational marriage of ReBAC and RBAC in one model.

7. CONCLUSION

In this work, we examined the relative expressiveness of two ReBAC policy languages. We began by demonstrating that there are known relational policies, such as $\text{cf}_6$ and $\text{clique}_4$, that cannot be expressed in the ReBAC policy language $\mathcal{B}$. We pointed out that the limitations were due to the lack of support for disjoint intermediaries and vertex identification. The two features were then incorporated into $\mathcal{B}$, resulting in an extended policy language $\mathcal{E}$. We showed that ReBAC policies definable in $\mathcal{E}$ are owner-checkable policies, a superset of relational policies. We also showed that every finitary relational policy is definable in $\mathcal{E}$, meaning that $\mathcal{E}$ is representationally complete with respect to that policy family. We have therefore identified a policy language that is representationally adequate for expressing useful ReBAC policies.

Acknowledgements

This work is supported in part by an NSERC Discovery Accelerator Supplements.

8. REFERENCES


of the First ACM Conference on Data and Application Security and Privacy (CODASPY’11), San Antonio, Texas, USA, February 2011. To appear. A preliminary version of the paper is available as [17].


APPENDIX
A. PROOF

Proposition 12. A relational policy is both OC and AC.

Proof. Suppose P is a relational policy. We show that P is OC. (The proof of the assertion that P is also AC is symmetrical.) Consider the social network G and vertices u, v ∈ V(G). It is obvious that C_G((u,v)) ⊆ G. There is no i ∈ I such that R_I((G)) \ C_G((u,v)) contains a pair (u',v') for which each of u' and v' is connected to u in G. Consequently, by Definition 5, P(C_G((u,v))) = P(G) \ P(C_G((u,v))) is an empty set, and thus P(G)(u,v) = P(C_G((u,v)))(u,v).

Theorem 13. A policy definable in E is OC.

Proof. Had it not been for the construct ⊕_p, this theorem could have been proven in a straightforward manner via structural induction on the abstract syntax tree of a formula. The subtlety arises from the fact that P may contain free occurrences of variables.

To address the presence of free variables, we prove a claim regarding a pair (Σ, φ), where Σ is an environment, and φ is a formula. A formula φ is said to be Σ-closed iff every free variable of φ belongs to the domain of Σ. A pair (Σ, φ) is a closure iff φ is Σ-closed. We also write π(Σ) to denote the environment Σ' defined over the same domain as Σ, such that Σ'(p) = π(Σ(p)).

Claim: Given a closure (Σ, φ) such that Σ(a) is defined, the following hold:

1. Let π be an isomorphism between social networks G and G'. Then G, Σ, u ⊨ φ iff G', π(Σ), π(u) ⊨ φ.
2. G, Σ, u ⊨ φ iff C_G; Σ(p), Σ, u ⊨ φ.

The proof of the claim is a straightforward structural induction on the abstract syntax tree of the formula in the closure.

The minimal set of birooted graphs to positively induce an OC policy has a special property that we will rely on in the proof of Theorem 20.

Lemma 23. Suppose P is both positive and OC. Let B be the minimal set of birooted graphs to positively induce P. For every G in B, every vertex in V(G) \ {v} is connected to u.

The proof of the lemma is elementary.

Theorem 20. Every finitary OC policy is definable in E.

Proof. We describe a sketch of the proof.

Suppose P is a finitary OC policy. As P is positive, let B be the minimal set of birooted graphs to positively induce P. Since P is finitary, B is finite. We outline in the following the construction of an E formula φ for each birooted graph G in B such that, for every G' in B, G' ⊨ φ implies G', p, φ' = G, φ'. The formula required by the theorem is the disjunction of the constructed formulas (recall that there are only finitely many of such formulas as B is finite).

Given birooted graph G in B, we construct a corresponding formula φ(u) as follows:

Step 1: We label each vertex by a distinct propositional symbol. (Since we have countably infinitely many propositional symbols, we always have enough symbols to work with.) In the following, we write p_u for the propositional symbol associated with a vertex u.

Step 2: Since P is both positive and OC, Lemma 23 implies that the vertices in G (perhaps with the exception of v) are connected to u. With u as the root, one can therefore build a depth-first search tree T. We assume that the search algorithm may traverse either along or against the direction of an edge. That is, a tree edge linking a parent vertex and a child vertex may go in either direction. Following standard terminology, we call the non-tree edges in G back edges. A back edge always links a descendant vertex with an ancestor vertex. (Again, a back edge may either point from a descendant vertex to an ancestor vertex, or point the other direction.) For more details regarding DFS trees, consult a standard text such as [12]. Note that v may or may not be part of T, but all other vertices are in T. Also, every edge in G is either a tree edge or a back edge.

Step 3: For each vertex x in search tree T, construct a formula φ(x) = ψ_1 \land ψ_2 \land ⊕_p_{x_1} (ψ_3 \land ψ_4), where:

• If x is not the root of search tree T (i.e., x ≠ u), then ψ_1 = ¬p_{x_1} \land ¬p_{x_2} \land ... \land ¬p_{x_k}, where x_1, x_2, ..., x_k are the proper ancestors of x (i.e., excluding x itself). In case x = u, then ψ_1 = Τ.
• If x = v, then ψ_2 = a. Otherwise, ψ_2 = ¬a.
• If x is a leaf of search tree T, then ψ_3 = Τ. Otherwise, a subformula is constructed for each child of x in T. Suppose (x, y) ∈ E(G), then we use (¬i) instead. Let ψ_{i_1}, ψ_{i_2}, ..., ψ_{m} be the subformulas constructed in this way. We construct ψ_3 = ψ_{i_1} \oplus ψ_{i_2} \oplus ... \oplus ψ_{m}.
• If x is not incident on a back edge in G, then ψ_4 = Τ. Otherwise, ψ_4 is a conjunction of formulas, one for each back edge linking x with one of its ancestors. Suppose (x, y) ∈ E(G) is such a back edge. The conjunct corresponding to this back edge is (i) p_y. If the back edge points in the other direction (i.e., (y, x) ∈ E(G)), then we use (¬i) instead.

The construction can proceed recursively, with φ(y) constructed prior to φ(x) whenever y is a child of x. The recursion terminates properly because T is a tree of finite size. The intuition behind the construction is that φ(u) is an encoding of G, including both the subtree T and all the back edges. More precisely, φ(u) encodes a DFS that
an authorization procedure can deploy to confirm that a birooted graph contains a subgraph isomorphic to the “pattern” \( G_{(u,v)} \). The detailed proof of this last claim is mechanical. □