Beyond Separation of Duty: An Algebra for Specifying High-level Security Policies

Ninghui Li and Qihua Wang
Purdue University

The process of introducing security controls into a sensitive task, which we call *secure task design* in this paper, consists of two steps: high-level security policy design and low-level enforcement scheme design. A high-level security policy states an overall requirement for a sensitive task. One example of a high-level security policy is a separation of duty policy, which requires a task to be performed by a team of at least k users. Unlike low-level enforcement schemes such as security constraints in workflows, a separation of duty policy states a high-level requirement about the task without referring to individual steps in the task. While extremely important and widely used, separation of duty policies state only requirements on the number of users involved in the task and do not capture the requirements on these users' attributes. In this paper, we introduce a novel algebra that enables the formal specification of high-level policies that combine requirements on users' attributes with requirements on the number of users motivated by separation of duty considerations. We give the syntax and semantics of the algebra and study algebraic properties of its operators. After that, we study potential mechanisms to enforce high-level policies specified in the algebra and a number of computational problems related to policy analysis and enforcement.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; F.4.3 [**Mathematical Logic And Formal Languages**]: Formal Languages; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Complexity of proof procedures*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security, Theory, Verification

1. INTRODUCTION

Sensitive tasks, such as declassifying a secret document or releasing funds, require security controls, such as approval from appropriate authority or double verification, to prevent abuse and fraud. The process of introducing security controls into a sensitive task, which we call *secure task design* in this paper, consists of two steps: the high-level *policy design* and the low-level *enforcement design*. In policy design, a *security officer* evaluates the risks, effects, and sensitivity of the task and determines which users should be involved in the task. Security officers have the authority to design security policies, but they are not required to have detailed knowledge of the actual steps through which the tasks are carried out. For instance, a security officer may determine that two managers must be involved in releasing a confidential document to a business partner. But he/she may not know the detailed steps of document releasing, which may include document retrieval, preparation, approval, and transportation. In enforcement design, a *system designer* designs a mechanism to model and control the execution of the business task in compliance with a given high-level security policy. A popular enforcement approach is security constraints in workflows. A workflow divides a task into a number of well-defined steps, and there are security constraints on users performing these steps. In contrast to security officers, system designers have detailed knowledge of task execution. They know what are the steps that must be performed to complete a task and who should be responsible for those steps. And they need

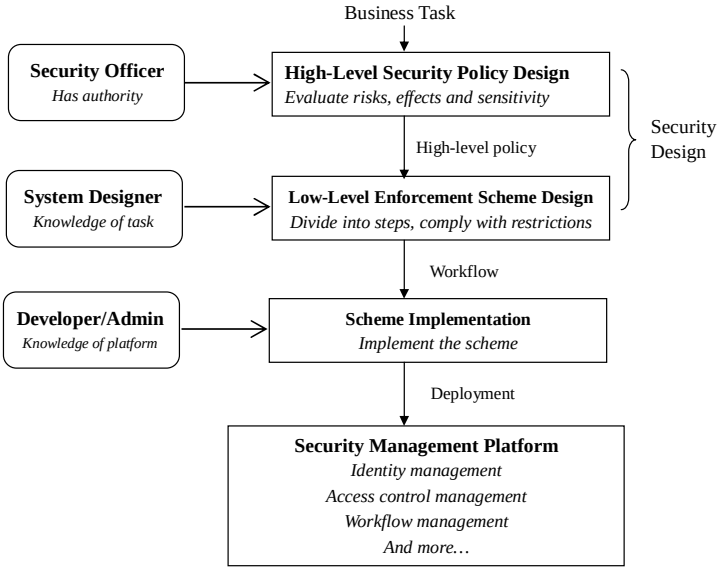


Fig. 1. Secure task design and the deployment of its output

to take into account efficiency, quality of service, and other practical restrictions. Finally, the output of a secure task design procedure is implemented and deployed in a security management solution, such as IBM Tivoli Identity Manager. Figure 1 briefly illustrates the procedure of secure task design.

As the focus of the first step of secure task design, a high-level security policy states an overall requirement that must be satisfied by any set of users that together complete a task. A well-known high-level security policy is Separation of Duty (SoD), which is widely recognized as a fundamental principle in computer security [Clark and Wilson 1987; Saltzer and Schroeder 1975]. In its simplest form, the principle states that a sensitive task should be performed by two different users acting in cooperation. The concept of SoD has long existed before the information age; it has been widely used in, for example, the banking industry and the military, sometimes under the name "the two-man rule". More generally, an SoD policy requires the cooperation of at least k different users to complete a task. SoD has been identified as a high-level mechanism that is "at the heart of fraud and error control" [Clark and Wilson 1987].

In many situations, however, it is not enough to require only that k different users be involved in a sensitive task; there are also minimal qualification requirements for these users. For example, one may want to require users involved in a task to be physicians, certified nurses, certified accountants, or directors of a company. Partly due to the lack of a concise-yet-expressive language for specifying such high-level security policies, people usually skip the formal specification of high-level security policies (perhaps expressing high-level security policies in a natural language) and specify qualification requirements at the level of enforcement mechanism. For example, if a designer believes that a task should involve a manager and two clerks, he/she may create a workflow with three steps and require two clerks to each perform Step 1 and Step 3, and a manager to perform Step

2.

However, formal specification of high-level security policies provides a number of important advantages. First of all, formal specification minimizes the possibility of misunderstanding between policy designers and system designers. Using a natural language could lead to ambiguity and misinterpretation, and are thus inappropriate to specify security policies, as a flaw in a policy could lead to major security breaches. Second, formal specification facilitates the analysis of security policies. Given a formal policy specification language, we may develop tools to analyze formally-specified policies, such as checking whether certain groups of users satisfy a policy, so as to detect policies that are too restrictive or too permissive when compared to actual needs in practice. It is beneficial to detect design flaws at an early stage, because low-level enforcement schemes, which contain execution details of tasks, are usually more difficult to analyze than high-level policies. As we will see in Example 2 in Section 3.2, low-level enforcement schemes such as workflows with security constraints may involve other factors in addition to security requirements, which complicates the analysis on those enforcement schemes. Finally, formal specification of high-level policies allows us to develop tools to verify whether a low-level enforcement scheme is compliant with a high-level security policy. For example, a workflow may contain branches and loops; it is important to verify that no route in the workflow bypasses the high-level security policy. As manual verification is time-consuming and error-prone, formal verification tools are highly desirable.

In this paper, we introduce a novel algebra that enables the formal specification of high-level policies that combine qualification requirements with quantity requirements motivated by separation of duty considerations. A term in our algebra specifies a requirement on sets of users (we call these *usersets*). A high-level policy, which associates a task with a term in the algebra, requires that all sets of users that complete an instance of the task satisfy the term. Our algebra has four binary operators: $\sqcup, \sqcap, \odot, \otimes$, and two unary operators $\neg, +$. An SoD policy that requires 3 different users can be expressed using the term $(\mathsf{All} \otimes \mathsf{All} \otimes \mathsf{All})$, where All is a keyword that refers to the set of all users. A policy that requires either a manager or two different clerks is expressed using the term $(\mathtt{Manager} \sqcup (\mathtt{Clerk} \otimes \mathtt{Clerk}))$.

We define the syntax and semantics of terms in the algebra, and study the algebraic properties of the operators. We then discuss enforcement mechanisms for policies specified in the algebra, such as static enforcement and dynamic enforcement. Furthermore, we study computational problems related to analysis and enforcement of policies specified in the algebra. These problems are: 1) the Term Satisfiability (TSAT) problem, which asks whether a term can be satisfied at all; 2) the Userset-Term Satisfaction (UTS) problem, which asks whether a userset (i.e. a set of users) satisfies a term; 3) the Userset-Term Safety (SAFE) problem, which asks whether a userset contains a subset that satisfies a term; and 4) the Static Safety Checking (SSC) problem, which asks whether an access control state is (statically) compliant with a high-level security policy.

Finally, some operators in our algebra are similar to the ones in regular expressions. A regular expression describes a set of strings, while a term in our algebra describes a set of sets. As a fundamental tool for defining sets of strings, regular expressions are used in many areas. Analogically, because our algebra is about the fundamental concept of defining sets of sets, we conjecture that, besides expressing security policies, the algebra could be used (perhaps with extensions) in other areas where set specification is desired.

The remainder of the paper is organized as follows. We introduce the syntax and semantics of the algebra in Section 2. We then discuss different enforcement mechanisms for policies specified in the algebra in Section 3. In Sections 4, 5, and 6, we study computational problems related to analysis and enforcement of policies. In Section 7, we discuss extensions to the syntax of the algebra, the relationship between the algebra and regular expressions, as well as limitations of the expressive power of the algebra. Finally, we discuss related work in Section 8 and conclude with Section 9. Proofs not included in the main body are included in the appendices unless otherwise stated.

2. THE ALGEBRA

In this section, we introduce an algebra for expressing high-level security policies.

2.1 Syntax, Semantics, and Examples

In our definition of the algebra, we use the notion of roles. We use a role to denote a set of users that have some common qualification or common job responsibility. We emphasize, however, that the algebra is not restricted to Role-Based Access Control (RBAC) systems [Sandhu et al. 1996]. In our algebra, a role is simply a named set of users. The notion of roles can be replaced by groups or user attributes. We use $\mathcal{U}$ to denote the set of all users, and $\mathcal{R}$ to denote the set of all roles.

DEFINITION 1 [TERMS IN THE ALGEBRA]. Terms in the algebra are defined as follows:

—An *atomic term* takes one of the following three forms: a role $r \in \mathcal{R}$, the keyword All, or a set $S \subseteq \mathcal{U}$ of users.

—An atomic term is a *unit term*; furthermore, if ϕ_1 and ϕ_2 are unit terms, then $\neg\phi_1$, $(\phi_1 \sqcap \phi_2)$ and $(\phi_1 \sqcup \phi_2)$ are also unit terms.

—A unit term is a *term*; if ϕ is a unit term, then ϕ^+ is a term; if ϕ_1 and ϕ_2 are terms, then $(\phi_1 \sqcup \phi_2)$, $(\phi_1 \sqcap \phi_2)$, $(\phi_1 \otimes \phi_2)$, and $(\phi_1 \odot \phi_2)$ are also terms.

The unary operator $\neg$ has the highest priority, followed by the unary operator $+$, then by the four binary operators (namely $\sqcap$, $\sqcup$, $\odot$, $\otimes$), which have the same priority.

We now give several simple example terms to illustrate the intuition behind the operators in the algebra. The term "(`Manager`$\sqcap$`Accountant`)" requires a user that is both a `Manager` and an `Accountant`. The term "(`Manager` $\sqcap \neg\{Alice, Bob\}$)" requires a user that is a manager, but is neither Alice nor Bob; here, the sub-term "$\neg\{Alice, Bob\}$" implements a blacklist. The term "(`Physician` $\sqcup$ `Nurse`)" requires a user that is either a `Physician` or a `Nurse`. The term "(`Manager` $\odot$ `Clerk`)" requires a user who is a `Manager` and a user who is a `Clerk`; in particular, when one user is both a `Manager` and a `Clerk`, that user by himself also satisfies the requirement. The term "((All$\otimes$All)$\otimes$All)" requires three different users. The keyword All allows us to refer to the set of all users. The term "`Accountant`$^+$" requires a set of one or more users, where each user in the set is an `Accountant`.

To formally assign meanings to terms, we need to first assign meanings to the roles used in the term. For this, we introduce the notion of configurations.

DEFINITION 2 [CONFIGURATIONS]. A *configuration* is given by a pair $\langle U, UR\rangle$, where U denotes the set of all users in the configuration, and $UR \subseteq U \times \mathcal{R}$ determines role memberships. When $(u, r) \in UR$, we say that u is a member of the role r.

Note that in a configuration $\langle U, UR\rangle$, UR should not be confused with the user-role assignment relation UA in RBAC. When an RBAC system has both UA and a role hierarchy RH, the two relations UA and RH together determine UR.

When describing the UR relation, we often use U_r to denote the set of users assigned to role r, i.e. $U_r = \{u \mid (u, r) \in UR\}$.

DEFINITION 3 [SATISFACTION OF A TERM]. Given a configuration $\langle U, UR\rangle$, we say that a userset $X \subseteq U$ *satisfies* a term ϕ under $\langle U, UR\rangle$ if and only if one of the following holds[1]:

—The term ϕ is the keyword All, and X is a singleton set $\{u\}$ such that $u \in U$.

—The term ϕ is a role r, and X is a singleton set $\{u\}$ such that $(u, r) \in UR$.

—The term ϕ is a set S of users, and X is a singleton set $\{u\}$ such that $u \in S$.

—The term ϕ is of the form $\neg\phi_0$ where ϕ_0 is a unit term, and X is a singleton set that does not satisfy ϕ_0.

—The term ϕ is of the form ϕ_0^+ where ϕ_0 is a unit term, and X is a nonempty userset such that for every $u \in X$, $\{u\}$ satisfies ϕ_0.

—The term ϕ is of the form $(\phi_1 \sqcup \phi_2)$, and either X satisfies ϕ_1 or X satisfies ϕ_2.

—The term ϕ is of the form $(\phi_1 \sqcap \phi_2)$, and X satisfies both ϕ_1 and ϕ_2.

—The term ϕ is of the form $(\phi_1 \otimes \phi_2)$, and there exist usersets X_1 and X_2 such that $X_1 \cup X_2 = X$, $X_1 \cap X_2 = \emptyset$, X_1 satisfies ϕ_1, and X_2 satisfies ϕ_2.

—The term ϕ is of the form $(\phi_1 \odot \phi_2)$, and there exist usersets X_1 and X_2 such that $X_1 \cup X_2 = X$, X_1 satisfies ϕ_1, and X_2 satisfies ϕ_2. This differs from the definition for $\otimes$ in that it does not require $X_1 \cap X_2 = \emptyset$.

For example, given the term $(\texttt{Manager} \odot \texttt{Clerk})$, and the configuration $\langle U = \{Alice, Bob\},\ UR\rangle$, in which UR is such that: $U_{\texttt{Manager}} = \{Alice\}$ and $U_{\texttt{Clerk}} = \{Alice, Bob\}$, we have $\{Alice\}$ satisfies the term and $\{Alice, Bob\}$ also satisfies the term.

Intuitively, a configuration $\langle U, UR\rangle$ represents the access control state of an organizational unit, a term ϕ defines the security requirement of a sensitive task T, and $X \subseteq U$ is a set of users in the organizational unit who are about to perform T. X satisfying ϕ indicates that the set of users meet the security requirement of T. Also, it is clear from Definition 3 that no term can be satisfied by an empty set.

The following examples help illustrate that one can express sophisticated policies in the algebra.

—$\{Alice, Bob, Carl\} \otimes \{Alice, Bob, Carl\}$
This term is satisfied by any two users out of the list of three.

—$(\texttt{Accountant} \sqcup \texttt{Treasurer})^+$
This term requires that all participants must be either an `Accountant` or a `Treasurer`. But there is no restriction on the number of participants except that the number is non-zero.

—$((\texttt{Manager} \odot \texttt{Accountant}) \otimes \texttt{Treasurer})$
This term is satisfied by a userset consisting of a `Manager`, an `Accountant`, and a `Treasurer`; the first two requirements can be satisfied by a single user.

[1] We sometimes say X satisfies ϕ, and omit "under $\langle U, UR\rangle$" when the configuration is clear from the context.

—$((\texttt{Physician} \sqcup \texttt{Nurse}) \otimes (\texttt{Manager} \sqcap \neg\texttt{Accountant}))$
This term is satisfied by a userset consisting of two different users, one of who is either a `Physician` or a `Nurse`, and the other is a `Manager`, but not an `Accountant`.

—$((\texttt{Manager} \odot \texttt{Accountant} \odot \texttt{Treasurer}) \sqcap (\texttt{Clerk} \sqcap \neg\{Alice, Bob\})^{+})$
This term is satisfied by a userset consisting of a `Manager`, an `Accountant` and a `Treasurer`. In addition, everybody in the userset must be a `Clerk` and must not be $Alice$ or Bob.

2.2 Satisfaction Trees

When a userset X satisfies a term ϕ under a configuration $\langle U, UR\rangle$, some subterms of ϕ are satisfied by subsets of X. We formalize this by the notion of a satisfaction tree. A satisfaction tree serves as an evidence of X satisfying ϕ that can be easily verified.

DEFINITION 4 [SATISFACTION TREE]. Given a term ϕ and a configuration $\langle U, UR\rangle$, we say that T is a satisfaction tree of ϕ under $\langle U, UR\rangle$ if and only if the following three conditions hold.

(1) T is a syntax tree of ϕ, where each inner node of T denotes a binary operator in ϕ, and each leaf node denotes a sub-term of ϕ that is either a unit term or takes the form ϕ_0^+. That is, sub-terms of the form ϕ_0^+ are not not further decomposed in T and are represented as leaves.

(2) Each node N in T is labeled with a (possibly empty) set of users, which is denoted as $L_T(N)$, and the following rules hold for every node N in T. We denote N_1 and N_2 as the left and right children of N, respectively.
—When N is a leaf node representing a unit term ϕ_0: either $L_T(N) = \emptyset$ or $L_T(N) = \{u\}$ satisfies ϕ_0 under $\langle U, UR\rangle$.
—When N is a leaf node representing a sub-term ϕ_0^+: either $L_T(N) = \emptyset$ or $L_T(N) = X$ satisfies ϕ_0^+ under $\langle U, UR\rangle$.
—When N represents $\sqcup$: either $(L_T(N) = L_T(N_1) \wedge L_T(N_2) = \emptyset)$ or $(L_T(N) = L_T(N_2) \wedge L_T(N_1) = \emptyset)$.
—When N represents $\sqcap$: $L_T(N) = L_T(N_1) = L_T(N_2)$.
—When N represents $\odot$: $L_T(N) = L_T(N_1) \cup L_T(N_2)$, and $(L_T(N) \neq \emptyset) \Rightarrow (L_T(N_1) \neq \emptyset \wedge L_T(N_2) \neq \emptyset)$, where $\Rightarrow$ denote logic implication.
—When N represents $\otimes$: $L_T(N) = L_T(N_1) \cup L_T(N_2)$, $L_T(N_1) \cap L_T(N_2) = \emptyset$, and $(L_T(N) \neq \emptyset) \Rightarrow (L_T(N_1) \neq \emptyset \wedge L_T(N_2) \neq \emptyset)$.

(3) $L_T(N_r) \neq \emptyset$, where N_r is the root of the tree T.

According to the conditions in the above definition, it can be easily shown that $L_T(N) \subseteq L_T(N')$, when N' is an ancestor of N in the satisfaction tree T.

Intuitively, in a satisfaction tree T, a node is either labeled with a userset that satisfies the sub-term represented by the sub-tree of T rooted at the node, or labeled with $\emptyset$, indicating that the node is in a branch connected by $\sqcup$ and the sub-term represented by that branch does not need to be satisfied (because the other branch is satisfied). The following lemma formalizes this intuition.

LEMMA 1. *Let T be a satisfaction tree of ϕ under $\langle U, UR\rangle$, for each node N in T, if $L_T(N) \neq \emptyset$, then $L_T(N)$ satisfies the sub-term of ϕ represented by the sub-tree of T rooted at N.*

The following theorem relates the existence of a satisfaction tree for a term ϕ with the satisfiability of ϕ.

THEOREM 2. Given a configuration $\langle U, UR\rangle$ and a term ϕ, a userset X satisfies ϕ under $\langle U, UR\rangle$ if and only if there exists a satisfaction tree of ϕ such that $L_T(N_r) = X$, where N_r is the root of T.

The proofs of Lemma 1 and Theorem 2 are given in Appendix A.

2.3 Evaluating a Term to a Set of Usersets

Given a configuration $\langle U, UR\rangle$ and a term ϕ, ϕ may be satisfied by multiple usersets, thus we can say that ϕ evaluates to a set of usersets.

DEFINITION 5 [VALUE OF A TERM]. Given a configuration $\langle U, UR\rangle$ and a term ϕ, $S_{\langle U,UR\rangle}(\phi)$ denotes the set of all usersets that satisfy ϕ under $\langle U, UR\rangle$, and is called the *value* of term ϕ under the configuration.

EXAMPLE 1. *Consider the term* $\phi = ((\texttt{Manager} \odot \texttt{Accountant} \odot \texttt{Treasurer}) \sqcap (\texttt{Clerk} \sqcap \neg\{Alice, Bob\})^+)$ *and the configuration* $\langle U = \{Alice, Bob, Carl, Doris, Elaine, Frank\},\ UR\rangle$*, in which* UR *is such that:*

$$\begin{aligned} U_{\texttt{Manager}} &= \{Alice, Doris, Elaine\} \\ U_{\texttt{Accountant}} &= \{Doris, Frank\} \\ U_{\texttt{Treasurer}} &= \{Bob, Carl, Doris\} \\ U_{\texttt{Clerk}} &= \{Alice, Bob, Carl, Doris, Frank\}. \end{aligned}$$

The sub-term $(\texttt{Clerk} \sqcap \neg\{Alice, Bob\})^+$ *blacklists* $Alice$ *and* Bob *so that only subsets of* $\{Carl, Doris, Frank\}$ *may satisfy* ϕ*. We have*

$$S_{\langle U,UR\rangle}(\phi) = \{\ \{Doris\},\ \{Carl, Doris\},\ \{Doris, Frank\}, \{Carl, Doris, Frank\}\ \}$$

That is, there are four usersets that satisfy the term ϕ*.*

2.4 Algebraic Properties

We now introduce the notion of equivalence among terms, which enables us to study the algebraic properties of the operators in the algebra.

DEFINITION 6 [TERM EQUIVALENCE]. We say that two terms ϕ_1 and ϕ_2 are *equivalent* (denoted by $\phi_1 \equiv \phi_2$) when for every userset X and every configuration $\langle U, UR\rangle$, X satisfies ϕ_1 under $\langle U, UR\rangle$ if and only if X satisfies ϕ_2 under $\langle U, UR\rangle$. In other words, $\phi_1 \equiv \phi_2$ if and only if $\forall\langle U, UR\rangle \left[S_{\langle U,UR\rangle}(\phi_1) = S_{\langle U,UR\rangle}(\phi_2)\right]$.

Using a straightforward induction on the structure of terms, one can show that if $\phi_1 \equiv \phi_2$, then, for any term ϕ in which ϕ_1 occurs, let ϕ' be the term obtained by replacing in ϕ one or more occurrences of ϕ_1 with ϕ_2, we have $\phi \equiv \phi'$.

THEOREM 3. The operators have the following algebraic properties:

(1) The operators $\sqcup, \sqcap, \odot, \otimes$ are commutative and associative. That is, for each $\mathsf{op} \in \{\sqcup, \sqcap, \odot, \otimes\}$, and any terms ϕ_1, ϕ_2, and ϕ_3, we have $(\phi_1\ \mathsf{op}\ \phi_2) \equiv (\phi_2\ \mathsf{op}\ \phi_1)$ and $((\phi_1\ \mathsf{op}\ \phi_2)\ \mathsf{op}\ \phi_3) \equiv (\phi_1\ \mathsf{op}\ (\phi_2\ \mathsf{op}\ \phi_3))$.
(2) The operators $\sqcup$ and $\sqcap$ distribute over each other. That is, $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3)) \equiv ((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$ and $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$.

(3) The operator $\odot$ distributes over $\sqcup$. That is, $(\phi_1 \odot (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$.
(4) The operator $\otimes$ distributes over $\sqcup$. That is, $(\phi_1 \otimes (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$.
(5) No other ordered pair of binary operators has the distributive property. (There are 12 such pairs altogether; the four of them listed above have the distributive property.)
(6) $(\phi_1 \sqcap \phi_2)^+ \equiv (\phi_1^+ \sqcap \phi_2^+)$
(7) DeMorgan's Law: $\neg(\phi_1 \sqcap \phi_2) \equiv (\neg\phi_1 \sqcup \neg\phi_2)$, $\neg(\phi_1 \sqcup \phi_2) \equiv (\neg\phi_1 \sqcap \neg\phi_2)$

See Appendix A for the proof of the above theorem, which also gives a counterexample for each case that the distributive property does not hold.

Because of the associativity properties, in the rest of this paper we omit parentheses in a term when doing so does not cause any confusion.

We now describe some other facts about the operators, to further illustrate the operators and their relationships.

—Any userset that satisfies $(\phi_1 \sqcap \phi_2)$ also satisfies $(\phi_1 \sqcup \phi_2)$, but not the other way around.
—Any userset that satisfies $(\phi_1 \sqcap \phi_2)$ also satisfies $(\phi_1 \odot \phi_2)$, but not the other way around.
—Any userset that satisfies $(\phi_1 \otimes \phi_2)$ also satisfies $(\phi_1 \odot \phi_2)$, but not the other way around.
—Any userset that satisfies $\phi_1^+ \sqcup \phi_2^+$ also satisfies $(\phi_1 \sqcup \phi_2)^+$, but not the other way around.

Proofs to the first three relationships are straightforward. Here, we prove the last one. If X satisfies $(\phi_1^+ \sqcup \phi_2^+)$, then X satisfies either ϕ_1^+ or ϕ_2^+. Without loss of generality, assume that X satisfies ϕ_1^+. Then, for every $u \in X$, $\{u\}$ satisfies ϕ_1 and thus satisfies $(\phi_1 \sqcup \phi_2)$. Hence, X satisfies $(\phi_1 \sqcup \phi_2)^+$. For the other direction, if $\{u_1\}$ satisfies ϕ_1 but not ϕ_2, and $\{u_2\}$ satisfies ϕ_2 but not ϕ_1, then $\{u_1, u_2\}$ satisfies $(\phi_1 \sqcup \phi_2)^+$ but not $\phi_1^+ \sqcup \phi_2^+$.

2.5 Rationale of the Design of the Algebra

We now discuss the rationale underlying some of the decisions we made in designing the algebra.

Monotonicity. SoD policies satisfy the property of monotonicity; that is, if an SoD policy requires two users to perform a task, then having three or more users certainly satisfies this policy. Similarly, one may want a security algebra like ours to also satisfy the monotonicity property; that is, if a userset X satisfies a term ϕ, then any superset of X also satisfies ϕ. McLean [McLean 1988] adopts this property in his security algebra for N-person policies.

Our algebra is designed to support both monotonic policies and policies that are not monotonic. For example, the term $(\texttt{Accountant} \otimes \texttt{Accountant})$ can be satisfied only by a set of two users; a set that contains more than two users cannot satisfy the term. More generally, in Definition 3, term satisfaction is defined in such a way that every user in the userset is used to satisfy certain component of the term. No "extra" user is allowed.

We have considered a design having the monotonicity property, in which we call the notion of satisfaction in Definition 3 "strict satisfaction" and define that a userset X satisfies a term ϕ if and only if X contains a subset that strictly satisfies ϕ. We chose our current design over the one that has the monotonicity property because the current design is more expressive. Consider the following example. When one says that "a task requires two `Accountants`", this may mean one of the following three policies:

(1) The task must be performed by a set of two users, both of whom are `Accountants`. A group containing more (or less) than two people is not allowed.

(2) The task must be performed by a set that contains two `Accountants`. In particular, a userset that contains two `Accountants` and a third user who is not an `Accountant` is allowed to perform the task.

(3) The task must be performed by a set of two or more `Accountants`. In particular, a set of three `Accountants` can perform the task, but a set of two `Accountants` and one non-`Accountant` cannot. This ensures that everyone involved in the task has the qualification of an `Accountant`.

Policies 1 and 3 cannot be expressed using an algebra that has the monotonicity property. Suppose that one tries to use a term ϕ to express policy 1 (or policy 3) in an algebra that has the monotonicity property, then a set X of two `Accountants` satisfies ϕ. By monotonicity property, any superset of X also satisfies ϕ. This violates the intention of policies 1 and 3. More generally, a monotonic algebra cannot express policies that disqualify usersets that contain extra users, nor can it express security requirements in the form of "all involved users must meet certain qualification requirements".

By dropping the monotonicity property, our algebra is able to express all the three policies. Policy 1 is expressed using the term $(\texttt{Accountant} \otimes \texttt{Accountant})$. Policy 2 is expressed using the term $((\texttt{Accountant} \otimes \texttt{Accountant}) \odot \mathsf{All}^+)$. Note that the term All^+ can be satisfied by any nonempty userset. Policy 3 is expressed using the term $(\texttt{Accountant} \otimes \texttt{Accountant}^+)$.

Restrictions on "¬" and "+". The syntax of our algebra (Definition 1) restricts that the two operators "¬" and "+" be applied only to unit terms, i.e., those terms that do not contain $\odot$, $\otimes$, or $+$. The motivation for this design decision is the psychological acceptability principle [Saltzer and Schroeder 1975]. We would like each operator to have a clear and intuitive meaning so that when one writes down a policy as a term, there is less chance to make mistakes and one is more confident that the term expresses the intended policy.

When $\neg$ is applied to a unit term, it expresses negative qualification about a single user; this has a clear meaning; the term $\neg\phi_0$ means a user that does not satisfy ϕ_0. However, if $\neg$ is applied to a term that involves $\odot$, $\otimes$, or $+$; then the meaning becomes less clear. Consider the term $\neg(\texttt{Accountant} \odot \texttt{Manager})$. Any userset of size three does not satisfy $(\texttt{Accountant} \odot \texttt{Manager})$; therefore, it should satisfy $\neg(\texttt{Accountant} \odot \texttt{Manager})$, even if every user in the userset is both an `Accountant` and a `Manager`. It is unclear to us what kind of real-world security policies such a term expresses.

The term ϕ_0^+, when ϕ_0 is a unit term, has a clear meaning; it means that every user must satisfy ϕ_0. The same term, when ϕ_0 involves operators such as $\odot$ and $\otimes$, has at least two possible meanings. One is to interpret $+$ as the closure operator of $\odot$, that is, a userset X satisfies ϕ_0^+ if and only if X can be divided into a number of (*possibly overlapping*) subsets such that each subset satisfies ϕ_0. The other is to interpret $+$ as the closure operator for $\otimes$, that is, a userset X satisfies ϕ_0^+ if and only if X can be divided into a number of *mutually disjoint* subsets such that each subset satisfies ϕ_0. The two meanings coincide when ϕ_0 is a unit term. We could use two operators, one for each meaning, and allow them to be applied to non-unit terms. However, this adds complexity to the algebra and we have not seen a need for this. For simplicity and usability, we chose to allow $+$ only be applied to unit terms. The algebra can be extended to have two closure operators that can be applied to non-unit terms, if a need for them arises in other application domains.

3. ENFORCING POLICIES SPECIFIED IN THE ALGEBRA

Once a high-level security policy has been specified in the algebra, we may proceed to enforcement design step. Before doing so, it is beneficial to perform certain analyses on the high-level policy to detect design flaws at an early stage.

A basic level of sanity check is to determine whether a term is satisfiable at all, as a term that cannot be satisfied in any configuration is probably not what a policy author intended. We define the Term Satisfiability (TSAT) problem for such an analysis. A problem similar to TSAT is the Term-Configuration Satisfiability (TCSAT) problem, which asks whether a term is satisfiable under a given configuration. This is useful when determining whether a term is meaningful in the current configuration of an organization. Formal definitions of TSAT and TCSAT are given in below, and we will study their computational complexity in Section 4.

DEFINITION 7 [TSAT]. Given a term ϕ, the *Term Satisfiability* (TSAT) *problem* determines whether there exists a configuration $\langle U, UR\rangle$ and a userset X such that X satisfies ϕ under $\langle U, UR\rangle$.

DEFINITION 8 [TCSAT]. Given a term ϕ and a configuration $\langle U, UR\rangle$, the *Term-Configuration Satisfiability* (TCSAT) *problem* determines whether there exists a userset X that satisfies ϕ under $\langle U, UR\rangle$.

Besides basic sanity checks on satisfiability, it is useful to select a number of targeted usersets and determine whether these usersets satisfy the term. If a set of users who are expected to perform the task guarded by the policy does not satisfy the term, the policy is too restrictive; if a set of users who should not be able to perform the task satisfies the term, the policy is too permissive. In either case, the policy is flawed and must be redesigned. We define the Userset-Term Satisfaction (UTS) problem for such an analysis. The computational complexity of UTS will be studied in Section 5.

DEFINITION 9 [UTS]. Given a term ϕ, a configuration $\langle U, UR\rangle$, and a userset X, the *Userset-Term Satisfaction* (UTS) *problem* determines whether X satisfies ϕ under $\langle U, UR\rangle$.

It is worth mentioning that UTS and TCSAT are related problems: given a configuration and a term, UTS is a decisional problem which asks whether a given userset satisfies the term, while TCSAT can be solved by searching for a userset in the configuration that satisfies the term.

If a high-level security policy passes all the tests, we need to enforce the policy correctly. A high-level security policy can be enforced statically or dynamically. In static enforcement, one ensures that in a configuration, any set of users who together have enough permissions to perform the task satisfy the high-level policy. In dynamic enforcement, one records the history of who performs which steps in a task instance and determines whether the set of users involved in the task instance satisfies the policy. In the rest of this section, we discuss these two enforcement approaches.

3.1 Static Enforcement

Static enforcement can be achieved either directly or indirectly. In direct static enforcement, one verifies whether an access control state is safe with respect to a high-level security policy. In indirect static enforcement, one specifies constraints so that any access control state satisfying the constraints is safe with respect to the policy.

Direct static enforcement of SoD policies, which are a subclass of the policies that can be specified in the algebra, has been studied in [Li et al. 2007]. It has been shown that checking whether an access control state statically satisfies an SoD policy, i.e., whether every set of users who together have all the permissions for the task contains at least k users, is $\mathbf{coNP}$-complete [Li et al. 2007]. As SoD policies can be specified in the algebra, direct statement enforcement of policies in the algebra requires solving an intractable problem. Computationally expense notwithstanding, we argue that the study of direct enforcement of static high-level policies is necessary for the following reasons. First, direct static enforcement is the most simple and straightforward enforcement mechanism for high-level security policies. Its performance will be used as a benchmark for comparison when evaluating other enforcement mechanisms. Second, even though direct static enforcement is computationally intractable in theory, it is interesting and necessary to study its performance for instances that are likely to occur in practice. Third, direct enforcement cannot be entirely replaced by indirect enforcement. It is oftentimes difficult or even impossible to create efficiently-verifiable constraints to precisely capture a high-level policy. For example, Li et al. studied indirect enforcement by using Static Mutually Exclusive Roles (SMER) to enforce SoD policies in the context of role-based access control (RBAC), and showed that there exist SoD policies such that no set of SMER constraints can *precisely* capture them [Li et al. 2007]. Most of the time, the set of constraints created for a security policy is more restrictive than the policy itself. That is to say, some access control states that are safe with respect to the security policy will be ruled out by the constraints. In situations where precise enforcement is desired, direct enforcement may be the only option.

Direct static enforcement requires solving the Static Safety Checking (SSC) problem, which we formally define through the following definitions.

DEFINITION 10 [STATE]. An access control system *state* is given by a triple $\langle U, UR, UP\rangle$, where $UR \subseteq U \times \mathcal{R}$ determines user-role memberships and $UP \subseteq U \times \mathcal{P}$ determines user-permission assignment, where $\mathcal{P}$ is the set of all permissions.

Note that a state $\langle U, UR, UP\rangle$ uniquely determines a configuration $\langle U, UR\rangle$ used by term satisfaction. Hence, we may discuss term satisfaction in a state without explicitly mentioning the corresponding configuration. Note also that a user may be assigned a permission directly or indirectly (e.g. via role membership), and the relation UP has taken both ways into consideration.

We say that a userset X *covers* a set P of permissions if and only if the following holds: $\{\, p \mid \exists u \in X[(u, p) \in UP] \,\} \supseteq P$.

Next, we define the notion of safety in direct static enforcement. As we mentioned earlier, the idea of static enforcement is that, by careful design of access control states, one can guarantee that every set of users who together have enough permissions to complete a task satisfies the security policy of the task, and thus runtime checking is unnecessary. While introducing no runtime overhead, static enforcement has a limitation, that is, only monotonic security policies can be enforced statically. The reason is that permission coverage is monotonic with respect to usersets. In other words, if X covers P, then any superset of X also covers P. However, as we emphasized in Section 2.5, term satisfaction does not have the monotonicity property. In order to specify monotonic policies, we may use terms in the form of $(\phi \odot \mathsf{All}^+)$. A userset U satisfies $(\phi \odot \mathsf{All}^+)$ if and only if U contains a subset that satisfies ϕ.

When static enforcement is the only enforcement approach, all policies need to be implicitly monotonic to be enforceable. We thus introduce the notion of static safety, which implicitly assumes each term means its monotonic closure.

DEFINITION 11 [STATIC SAFETY]. A high-level security policy is given as a pair $\mathsf{sp}\langle P, \phi\rangle$, where $P \subseteq \mathcal{P}$ is a set of permissions and ϕ is a term in the algebra. An access control state $\langle U, UR, UP\rangle$ is *statically safe* with respect to $\mathsf{sp}\langle P, \phi\rangle$, if and only if, for every userset X that covers P, X satisfies the monotonic closure of ϕ (i.e. X satisfies $(\phi \odot \mathsf{All}^+)$). If a state is statically safe with respect to a policy, we say that it *satisfies* the policy.

Note that in the above definition, we require that, for each userset X that covers P, X satisfies the monotonic closure of ϕ rather than ϕ itself. Equivalently, an access control state is statically safe with respect to $\mathsf{sp}\langle P, \phi\rangle$ if and only if for every userset X that covers P, there exists $X' \subseteq X$, such that X' satisfies ϕ.

The problem of checking static safety is defined as follows; its computational complexity will be studied in Section 6.1.

DEFINITION 12 [SSC]. Given a static safety policy $\mathsf{sp}\langle P, \phi\rangle$, the problem of determining whether a given state $\langle U, UR, UP\rangle$ is statically safe with respect to $\mathsf{sp}\langle P, \phi\rangle$ is called the *Static Safety Checking (*SSC*) problem*.

Note also that Definition 11 does not require $\langle U, UR, UP\rangle$ to contain a userset that covers P in $\mathsf{sp}\langle P, \phi\rangle$. If a state does not contain any userset that covers P, then it trivially satisfies $\mathsf{sp}\langle P, \phi\rangle$. Checking whether there exists a userset in $\langle U, UR, UP\rangle$ that covers P can be done in linear time with respect to the size of UP.

To check static safety, one needs to determine whether a set of users contains a subset that satisfy a term. This problem is defined as follows.

DEFINITION 13 [SAFE]. Given a term ϕ, a configuration $\langle U, UR\rangle$, and a userset X, a userset X is *safe* with respect to a term ϕ under configuration $\langle U, UR\rangle$, if and only if there exists $X' \subseteq X$ such that X' satisfies ϕ under $\langle U, UR\rangle$.

The *Userset-Term Safety (*SAFE*) problem* determines whether X is *safe* with respect to a term ϕ under configuration $\langle U, UR\rangle$.

SAFE can be viewed as a special case of UTS, because X is safe with respect to ϕ if and only if X satisfies $(\phi \odot \mathsf{All}^+)$; however, it may be solved more efficiently when treated as a separate problem. The computational complexity of SAFE will be studied in Section 6.

We point out that SAFE is technically the same problem as TCSAT, even though they are motivated by different purposes. In SAFE, we ask whether a userset X contains a subset that satisfies ϕ under $\langle U, UR\rangle$, where $X \subseteq U$. Since users in U/X are irrelevant in answering such a question, the problem is equivalent to whether X contains a subset that satisfies ϕ under $\langle X, UR\rangle$, which is the same as whether there is a userset in the configuration $\langle X, UR\rangle$ that satisfies ϕ.

As we mentioned earlier, static enforcement can only enforce security policies with the monotonicity property. To enforce non-monotonic policies, we may use a dynamic enforcement scheme.

3.2 Dynamic Enforcement

Similar to static enforcement, dynamic enforcement can be achieved either directly or indirectly as well.

To directly enforce a policy $\langle task, \phi \rangle$, one identifies the steps in performing the task. The system maintains a history of each instance of the task, which includes information on who have performed which steps. For any task instance, one can compute the set of users (denoted as U_{past}) who have performed at least one step of the instance. Before a user u performs a step of the instance, the system checks to ensure that there exists a superset of $U_{past} \cup \{u\}$ that can satisfy ϕ upon finishing all steps of the task. In particular, if u is about to perform the last step of the task instance, it is required by the policy that $U_{past} \cup \{u\}$ satisfies ϕ. As we will see in Section 5, checking whether a userset satisfies a term is computationally expensive. In practice, people usually use workflows with security constraints to indirectly enforce high-level security policies.

In the rest of this section, we give an example of the secure task design process. The example demonstrates how to use a workflow as an indirect dynamic enforcement scheme for a high-level security policy specified in the algebra. We would like to point out that in the design of workflows, a designer may take efficiency, quality of service, and other practical restrictions into account in addition to security requirements.

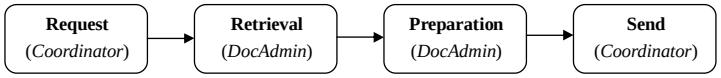
EXAMPLE 2. *Company* XYZ *newly established a plan to share some of its classified documents with its business partners. As the task (denoted as T_s) involves disclosure of classified documents, it is considered to be sensitive by* XYZ *and has to go through a security design procedure. The first step is the high-level policy design, which is performed by a security officer $Alice$. After evaluating the risks and effects of T_s, $Alice$ decides that at least two* `Managers` *must be involved in the task. She then creates a high-level security policy* $(\texttt{Manager} \otimes \texttt{Manager}) \odot \mathsf{All}^+$ *for T_s.*

The second step is to design a workflow to model T_s in compliance with the high-level security policy. This is performed by a system designer Bob. T_s consists of four physical steps: 1) a business partner coordinator (denoted as `Coordinator`*) receives a request from a business partner; 2) a document administrator (denoted as* `DocAdmin`*) retrieves the document from company archives; 3) a* `DocAdmin` *performs pre-releasing preparation on the document, such as anonymizing certain items; 4) a* `Coordinator` *sends the post-preparation document to the business partner. To begin with, Bob creates a workflow W_1 with the four physical steps of T_s, which is shown in Figure 2-a. He then introduces two additional steps into W_1 so as to comply with the security policy* $(\texttt{Manager} \otimes \texttt{Manager}) \odot \mathsf{All}^+$. *He adds two steps to W_1 so that a classified document will not be retrieved until two* `Managers` *have approved the request on disclosure. Furthermore, in order to provide better quality of service, Bob adds a binding of duty constraint to the workflow so that the coordinator who received the request is responsible to send the document to the business partner. The final workflow W_2 modeling T_s is shown in Figure 2-b. It can be verified that any team of users who completes W_2 must satisfy* $(\texttt{Manager} \otimes \texttt{Manager}) \odot \mathsf{All}^+$.

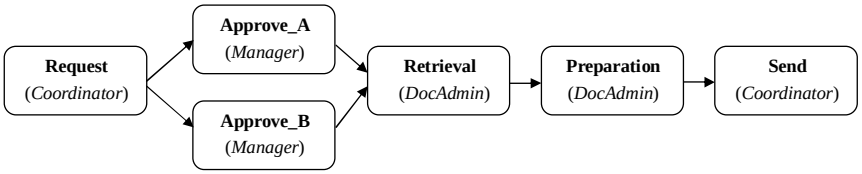
It is interesting future work to study how to verify whether a workflow is compliant with a high-level security policy specified in the algebra. In the upcoming sections, we will study the computational problems (i.e. TSAT, TCSAT, UTS, SAFE and SSC) defined in this section. As we have seen, these problems are important in the analysis and enforcement of high-level security policies, and are of both theoretical and practical interest.

4. TWO TERM SATISFIABILITY PROBLEMS

In this section, we study the computational complexities of TSAT and TCSAT.



a. A workflow (W_1) consisting of physical steps of the task on releasing classified documents to corporate partners. **Request** and **Send** are authorized to the role *Coordinator*, while **Retrieval** and **Preparation** are authorized to *DocAdmin*.



Constraint 1: **Approve_A** and **Approve_B** must be performed by different users

Constraint 2: **Request** and **Send** must be performed by the same user

b. A workflow (W_2) consisting of physical steps, security-oriented steps and constraints in compliance with a high-level security policy. **Approve_A** and **Approve_B** are authorized to *Manager* and may be performed in parallel. Constraint 2 is specified for the purpose of quality of service.

Fig. 2. Workflows in Example 2

4.1 The Term Satisfiability (TSAT) Problem

As the algebra supports negation, it is not surprising that unsatisfiable terms exist. A simple example of a term that is not satisfiable is $(r \sqcap \neg r)$. Another source of unsatisfiable terms is the use of explicit sets of users in a term. For example, the term $(\{Alice, Bob\} \sqcap \{Carl\})$ is not satisfiable. However, even if a term does not contain negation or explicit sets of users, it may still be unsatisfiable. An example of such a term is $\phi = (r_1 \sqcap (r_2 \otimes r_3))$, where r_1, r_2 and r_3 are roles. In the example, r_1 is satisfiable only by a singleton userset, and $(r_2 \otimes r_3)$ is satisfiable only by a userset of cardinality 2. Therefore, there does not exist a userset that satisfies ϕ.

We now show that TSAT is $\mathbf{NP}$-complete in general. We identify the source of intractability by identifying two special cases that are $\mathbf{NP}$-hard. One special case (Lemma 4 below) involves the negation operator, and the other (Lemma 5 below) involves explicit sets of users. In Section 4.2, we show that for terms that are free of negation and explicit sets of users, TSAT can be efficiently solved.

LEMMA 4. TSAT over terms built using only roles and the operators $\neg$, $\sqcap$, and $\sqcup$ is $\mathbf{NP}$-hard.

LEMMA 5. TSAT over terms built using only explicit sets of users and the operators $\sqcap$, $\sqcup$, and $\odot$ is $\mathbf{NP}$-hard.

To show that TSAT is in $\mathbf{NP}$, we need the following lemma, which shows that if a term is satisfiable, then there exists an evidence of polynomial size.

LEMMA 6. If a term ϕ is satisfiable, then there exists a userset U and a configuration $\langle U, UR \rangle$, such that U satisfies ϕ under $\langle U, UR \rangle$, $|U| \leq |\phi|$ and $|UR| \leq |\phi|^2$, where $|\phi|$ is the number of occurrences of atomic terms in ϕ.

THEOREM 7. TSAT is $\mathbf{NP}$-complete.

Please refer to Appendix B.1 for the proofs of Lemmas 4, 5, 6 and Theorem 7.

4.2 TSAT for the Sub-Algebra Free of Negation and Explicit Sets of Users

Lemmas 4 and 5 show that if a term involves negation or explicit sets of users, then determining whether it is satisfiable or not may be intractable. We now study the term satisfiability problem for terms that are free of explicit sets of users and negation. For convenience, we call such terms *SNF (Set-and-Negation Free) terms*. The following lemma states an important property of terms that are free of negation.

LEMMA 8. *Let ϕ be a term that does not contain the operator $\neg$. If userset X satisfies ϕ under configuration $\langle U, UR\rangle$, then X satisfies ϕ under configuration $\langle U, UR'\rangle$, where $UR \subset UR'$.*

Lemma 8 essentially states that, for terms that are free of negation, satisfaction is monotonic with respect to user-role assignment. The proof of the lemma is straightforward and is omitted from the paper.

We have the following theorem.

THEOREM 9. *Checking whether an SNF term is satisfiable is in* $\mathbf{P}$.

To prove Theorem 9, we first introduce the notion of characteristic sets for SNF terms in Definition 14. Definition 14 essentially gives an algorithm to compute the characteristic set of a given SNF term. Then, we show that the algorithm given in Definition 14 is a polynomial time algorithm. Finally, we prove an important property of characteristic set, that is, an SNF term is satisfiable if and only if its characteristic set is non-empty. To determine whether an SNF term is satisfiable, we can run a polynomial-time algorithm to compute its characteristic set and check whether the characteristic set is empty or not.

To begin with, we introduce the notion of characteristic sets. A key observation is that, in order to satisfy a term, a userset must be of certain size. For example, $(r_1 \odot (r_2 \otimes r_3))$ can be satisfied by a set of 2 or 3 users, but not by a set containing 1 or 4 or any other number of users. We thus call $\{2, 3\}$ the characteristic set of the term $(r_1 \odot (r_2 \otimes r_3))$.

DEFINITION 14 [CHARACTERISTIC SET]. The *characteristic set* of an SNF term ϕ, which is denoted as $C(\phi)$, is a set of natural numbers computed as follows:

—$C(\mathsf{All}) = C(r) = \{1\}$, where r is a role
—$C(\phi_1 \sqcup \phi_2) = C(\phi_1) \cup C(\phi_2)$
—$C(\phi_1 \sqcap \phi_2) = C(\phi_1) \cap C(\phi_2)$
—$C(\phi^+) = \{i \mid i \in [1, \infty)\}$, where ϕ is a unit term free of explicit sets of users and negations
—$C(\phi_1 \odot \phi_2) = \{i \mid \exists\, c_1 \in C(\phi_1)\ \exists\, c_2 \in C(\phi_2)\ [\, max(c_1, c_2) \leq i \leq c_1 + c_2\,]\}$
—$C(\phi_1 \otimes \phi_2) = \{\, c_1 + c_2 \mid c_1 \in C(\phi_1)\ \wedge\ c_2 \in C(\phi_2)\, \}$

An integer k is called a *characteristic number* of ϕ if and only if $k \in C(\phi)$.

Note that the above definition states how to compute the characteristic set of a given SNF term. As examples, we give the characteristic sets of some terms in below.

—$C(\mathsf{All} \otimes \mathsf{All} \otimes \mathsf{All}) = \{3\}$

—$C(\texttt{Manager} \odot \texttt{Accountant}) \otimes \texttt{Treasurer}) = \{2, 3\}$
The term $(\texttt{Manager} \odot \texttt{Accountant})$ can be satisfied by two users as well as by a single user who is both a `Manager` and an `Accountant`. An additional user is needed to satisfy `Treasurer`.

—$C((\texttt{Clerk} \sqcup \texttt{Accountant}) \otimes (\texttt{Clerk} \sqcap \texttt{Manager})) = \{2\}$
One user is required for $(\texttt{Clerk} \sqcup \texttt{Accountant})$, and for $(\texttt{Clerk} \sqcap \texttt{Manager})$, and the $\otimes$ mandates that the two terms be satisfied by different users.

—$C((\texttt{Manager} \odot \texttt{Accountant} \odot \texttt{Treasurer}) \sqcap \texttt{Clerk}^{+}) = \{1, 2, 3\} \cap \{i | i \in [1, \infty)\} = \{1, 2, 3\}$

Given a term ϕ, computing $C(\phi)$ requires at most $2|\phi| - 1$ steps according to the algorithm in Definition 14, where $|\phi|$ is the number of occurrences of atomic terms in ϕ and ϕ contains $|\phi| - 1$ binary operators. A step in the algorithm may require such operations: set union, set intersection, computing the sums of all pairs of elements from two different sets. If the size of intermediate results (which are sets) is bounded by $|\phi|$, then each step can be performed in polynomial time, and thus the algorithm finishes in polynomial time. However, when a term contains $^{+}$, its characteristic set could be an infinite set. Fortunately, the following Lemma 10 shows that if $C(\phi)$ is an infinite set, it must always contain all the numbers that are greater than $|\phi|$. In this case, we do not have to deal with infinitely many elements in a characteristic set individually, as $\{|\phi| + 1, |\phi| + 2, \dots\}$ can be treated as one unit during computation.

LEMMA 10. Let ϕ be an SNF term and $|\phi|$ be the number of occurrences of atomic terms in ϕ. One of the following two cases holds:

—$C(\phi) \subseteq \{1, 2, \dots, |\phi|\}$
—$C(\phi) = W \cup \{|\phi| + 1, |\phi| + 2, \dots\}$, where $W \subseteq \{1, 2, \dots, |\phi|\}$

With Lemma 10, we can prove the following lemma. The proofs of Lemmas 10 and 11 are given in Appendix B.2.

LEMMA 11. Given an SNF term ϕ, $C(\phi)$ can be computed in polynomial time with respect to $|\phi|$.

The following theorem states an important property of characteristic sets.

THEOREM 12. Given an SNF term ϕ and a positive integer k, there exists a userset U of size k and a configuration such that U satisfies ϕ under the configuration, if and only if k is a characteristic number of ϕ (i.e. $k \in C(\phi)$).

The proof of Theorem 12 is given in Appendix B.2.

COROLLARY 13. An SNF term ϕ is satisfiable if and only if $C(\phi) \neq \emptyset$

With Lemma 11 and the above corollary, we can see that TSAT over SNF terms is in $\mathbf{P}$.

Another usage of characteristic set is to determine whether a term satisfies some minimal SoD requirements. If the smallest characteristic number of the term is k, then no $k - 1$ users can satisfy the term.

Finally, we can extend the notion of characteristic set to non-SNF terms by defining $C(\neg\phi) = \{1\}$, where ϕ is a unit term, and $C(S) = \{1\}$, where S is an explicit set of users. But in that case, it is no longer true that for every integer $k \in C(\phi)$, there is a userset of

size k that satisfies ϕ. For example, $C(\{Alice, Bob\} \sqcap \{Carl\}) = C(\{Alice, Bob\}) \cap C(\{Carl\}) = \{1\}$, even though the term $(\{Alice, Bob\} \sqcap \{Carl\})$ is not satisfiable. But it remains true that for any userset X that satisfies a term ϕ, $|X| \in C(\phi)$.

4.3 The Term-Configuration Satisfiability (TCSAT) Problem

We have discussed the TSAT problem, which asks whether a term is satisfiable at all. We now examine the TCSAT problem, which asks whether a term is satisfiable under a certain configuration. When a security officer comes up with a term for a high-level security policy of a task, he/she may want to know whether there exists a set of users that satisfies the term and hence is able to perform the task under the current configuration.

Observe that TCSAT is equivalent to TSAT for terms using only explicit sets of users but not roles or the keyword All. Given an instance of TCSAT, which consists of a term ϕ and a configuration $\langle U, UR\rangle$, one can replace each role (or the keyword All) in ϕ with the corresponding set of users in the configuration, which results in a new term ϕ'. In this case, ϕ' is independent of configuration, and ϕ is satisfiable under $\langle U, UR\rangle$ if and only if ϕ' is satisfiable. Therefore, it follows from Lemma 5 and Theorem 7 that TCSAT is $\mathbf{NP}$-complete; this is stated in the following theorem.

THEOREM 14. TCSAT is $\mathbf{NP}$-complete.

We mentioned earlier that TCSAT is equivalent to SAFE. In Section 6 we will examine the computational complexities of SAFE when only some subsets of operators are allowed. Those results for SAFE apply to TCSAT as well.

5. THE USERSET-TERM SATISFACTION (UTS) PROBLEM

In this section, we study the computational complexities of the Userset-Term Satisfaction (UTS) problem, which asks: Given a configuration $\langle U, UR\rangle$, a userset X, and a term ϕ, whether X satisfies ϕ under $\langle U, UR\rangle$? We will show that UTS in the most general case (i.e., arbitrary terms in which all operators are allowed) is $\mathbf{NP}$-complete. In order to understand how the operators affect the computational complexities, we consider sub-algebras in which only some subset of the six operators $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed. For example, UTS$\langle \neg, +, \sqcup, \sqcap\rangle$ denotes the sub-case of UTS where ϕ does not contain operators $\odot$ or $\otimes$, while UTS$\langle\otimes\rangle$ denotes the sub-case of UTS where $\otimes$ is the only kind of operator in ϕ. UTS$\langle \neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ denotes the general case. Observe that unlike in the case of TSAT, whether to allow explicit sets of users in a term or not does not affect the computational complexities of UTS, because a fixed configuration is given in UTS, and one can thus replace each occurrence of a role in the term with the explicit set of the role's members.

THEOREM 15. The computational complexities of UTS and its subcases are given in Table I.

The proof of Theorem 15 is done in two parts. First, in Appendix C.1, we prove that the five cases UTS$\langle \sqcup, \odot\rangle$, UTS$\langle \sqcap, \odot\rangle$, UTS$\langle \sqcup, \otimes\rangle$, UTS$\langle \sqcap, \otimes\rangle$, and UTS$\langle \odot, \otimes\rangle$ are $\mathbf{NP}$-hard by reducing the $\mathbf{NP}$-complete problems SET COVERING, DOMATIC NUMBER, and SET PACKING to them. Second, in Appendix C.2, we prove that the general case UTS$\langle \neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{NP}$. In Section 5.1, we identify a wide class of syntactically restricted terms for which the UTS problem is tractable. The class of restricted terms subsumes all the cases listed as in $\mathbf{P}$ in Table I.

$\neg$	$+$	$\sqcup$	$\sqcap$	$\odot$	$\otimes$	**Complexity**	**Reduction**
✓	✓	✓	✓	✓	✓	**NP**-complete	
		✓		✓		**NP**-complete	Set Covering
		✓			✓	**NP**-complete	Set Packing
			✓	✓		**NP**-complete	Set Covering
			✓		✓	**NP**-complete	Set Covering
				✓	✓	**NP**-complete	Domatic Number
✓	✓	✓	✓			**P**	
✓	✓			✓		**P**	
✓	✓				✓	**P**	

Table I. Various sub-cases of the Userset Term Satisfaction (UTS) problem and the corresponding time-complexity. Time-complexity of all other subcases can be deduced from the subcases shown in the table.

5.1 UTS is Tractable for Terms in Canonical Forms

From Table I, UTS is **NP**-complete in all but one sub-algebras that contain at least two binary operators; however, using any one binary operator by itself remains tractable. In this subsection, we show that if a term satisfies certain syntactic restrictions, then even if all operators appear in the term, one can still efficiently determine whether a userset satisfies the term.

DEFINITION 15 [CANONICAL FORMS FOR TERMS]. The canonical forms for terms are defined as follows:

—A term is *in level-1 canonical form* (called a 1CF term) if it is t or t^+, where t is a unit term. Recall that a unit term can use the operators $\neg$, $\sqcap$, and $\sqcup$. We call t the *base* of the 1CF term.

—A term is *in level-2 canonical form* (called a 2CF term) if it consists of one or more sub-terms that are 1CF terms, and these sub-terms are connected only by the operator $\sqcap$.

—A term is *in level-3 canonical form* (called a 3CF term) if it consists of one or more sub-terms that are 2CF terms, and these sub-terms are connected only by the operator $\otimes$.

—A term is *in level-4 canonical form* (called a 4CF term) if it consists of one or more sub-terms that are 3CF terms, and these sub-terms are connected only by the operator $\odot$.

—A term is *in level-5 canonical form* (called a 5CF term) if it consists of one or more sub-terms that are 4CF terms, and these sub-terms are connected only by operators in the set $\{\sqcup, \sqcap\}$.

We say that a term is *in canonical form* if it is in level-5 canonical form. Observe that any term that is in level-i canonical form is also in level-$(i+1)$ canonical form for any $i \in [1, 4]$.

To check whether a term ϕ is in canonical form, one parses ϕ into a syntax tree and then traverses the tree in a depth-first manner to see if any syntactical restriction described in Definition 15 is violated. This can be done in polynomial time in the size of ϕ.

THEOREM 16. Given a term ϕ in canonical form, a set X of users, and a configuration $\langle U, UR\rangle$, checking whether X satisfies ϕ under $\langle U, UR\rangle$ can be done in polynomial time.

PROOF. Recall that, by definition, X satisfies $\phi_1 \sqcap \phi_2$ if and only if X satisfies both ϕ_1 and ϕ_2, and X satisfies $\phi_1 \sqcup \phi_2$ if and only if X satisfies either ϕ_1 or ϕ_2. Therefore, to determine whether X satisfies a 5CF term, one can first determine whether X satisfies each of the 4CF sub-terms, and then combine these results using logical conjunction and disjunction.

For a 1CF term ϕ, if it is a unit term, then it is straightforward to determine whether X satisfies ϕ, because a unit term can be satisfied only by a singleton set, and because of the definitions of $\sqcap$ and $\sqcup$. If ϕ is of the form t^+, where t is a unit term, then one just needs to determine whether each user in X satisfies t. Therefore, one can efficiently check whether X satisfies a 1CF term.

Given a 2CF term, if at least one sub-term is a unit term, then one can get an equivalent 1CF term by removing all occurrences of $^+$. For example, $(t_1 \sqcap t_2^+)$ is equivalent to $t_1 \sqcap t_2$. Given a 2CF term where all sub-terms have $^+$, it may be rewritten as an equivalent 1CF term, according to algebraic properties. For example, $(t_1^+ \sqcap t_2^+)$ is equivalent to $(t_1 \sqcap t_2)^+$. Hence, any 2CF term can be transformed into an equivalent 1CF term. We assume that the transformation is performed whenever applicable so that we don't need to consider 2CF terms explicitly.

Given a 3CF term $P = (\phi_1 \otimes \cdots \otimes \phi_m)$, where each ϕ_i is a 1CF term. Let us first consider a special case that each ϕ_i is a unit term t_i. In this case, one can determine whether X satisfies ϕ_i by solving the following bipartite graph maximal matching problem. One constructs a bipartite graph such that one set of nodes consists of users in X and the other consists of the m unit terms $t_1, t_2, \ldots, t_m$; and there is an edge between $u \in X$ and t_i if and only if $\{u\}$ satisfies t_i. One then computes a maximal matching of the graph (which can be done in polynomial time); if the size of the matching is $\max(|X|, m)$, then X satisfies P; otherwise, X does not satisfy P.

The case that a 3CF term contains $+$ is more complicated, as is the case for a 4CF term. The proof for the 4CF case (which subsumes the 3CF case) is long and offers limited new insights. We thus leave the proof in Appendix C.3. □

Terms in canonical forms appear to be general enough to specify many high-level security policies in practice. We arrive at these canonical forms by excluding the intractable cases used in the **NP**-hardness proofs, and by studying how to efficiently handle terms involving the binary operators.

6. THE USERSET-TERM SAFETY (SAFE) PROBLEM AND THE STATIC SAFETY CHECKING (SSC) PROBLEM

In this section, we study the Userset-Term Safety (SAFE) problem and the Static Safety Checking (SSC) problem.

As we have pointed out in Section 4.3, SAFE is technically equivalent to TCSAT, even though the two problems are motivated by different purposes. Since TCSAT is **NP**-complete, SAFE is **NP**-complete in general.

Also, SAFE is related to yet different from UTS. SAFE asks whether X is safe with respect to a term ϕ under a configuration; this is monotonic in that if X is safe, then any superset of X is also safe. However, UTS is not monotonic. This difference has subtle but important effects. For example, under SAFE, the operator $\odot$ is equivalent to logical

$\neg$	$+$	$\sqcup$	$\sqcap$	$\odot$	$\otimes$	**Complexity**	**Reduction**
✓	✓	✓	✓	✓	✓	**NP**-complete	
		✓			✓	**NP**-complete	Set Packing
			✓	✓		**NP**-complete	Set Covering
			✓		✓	**NP**-complete	Set Covering
				✓	✓	**NP**-complete	Domatic Number
✓	✓	✓	✓			**P**	
✓	✓	✓		✓		**P**	
✓	✓				✓	**P**	

Table II. Various sub-cases of the Userset-Term Safety (SAFE) problem and the corresponding time-complexity. Time-complexity of all other subcases can be deduced from the subcases shown in the table. The complexity results in the table also apply to TCSAT, as TCSAT is technically equivalent to SAFE.

conjunction, that is, X is safe with respect to $\phi_1 \odot \phi_2$ if and only if X is safe with respect to both ϕ_1 and ϕ_2. This is because X is safe with respect to $\phi_1 \odot \phi_2$ if and only if X contains a subset X_0 that is the union of two subsets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. This is equivalent to X containing two subsets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. In contrast, the operator $\odot$ is different from logical conjunction under UTS. That X satisfies $\phi_1 \odot \phi_2$ does not imply X satisfies both ϕ_1 and ϕ_2. For example, $\{u_1, u_2\}$ satisfies All $\odot$ All, but does not satisfy All, because term satisfaction is not monotonic. Another difference regards the operator $\sqcap$. The operator $\sqcap$ is equivalent to logical conjunction under UTS, by definition of term satisfaction. However, $\sqcap$ is stronger than logical conjunction under SAFE. That X is safe with respect to $\phi_1 \sqcap \phi_2$ implies that X is safe with respect to both ϕ_1 and ϕ_2, but the other direction is not true. For example, given $UR = \{(u_1, r_1), (u_2, r_2)\}$, $X = \{u_1, u_2\}$ is safe with respect to both r_1 and r_2, but is not safe with respect to $r_1 \sqcap r_2$.

Because of these and other differences, the computational complexity results about UTS do not imply computational complexity results for SAFE. In the rest of this section, we give the computational complexities of SAFE and its subcases, and compare them with those of UTS. Similar to the discussion of UTS in Section 5, we consider all sub-algebras in which only some subset of the six operators in $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed.

THEOREM 17. The computational complexities of SAFE and its subcases are given in Table II.

Please refer to Appendix D for proofs of the above theorem. In the appendix, we first prove that the three cases SAFE$\langle \neg, +, \sqcap, \sqcup \rangle$, SAFE$\langle \neg, +, \sqcup, \odot \rangle$, and SAFE$\langle \neg, +, \otimes \rangle$ are in $\mathbf{P}$. As we mentioned at the beginning of the section, SAFE is $\mathbf{NP}$-complete in general, which implies that all of its subcases are in $\mathbf{NP}$. Hence, to prove all the $\mathbf{NP}$-completeness results, it suffices to prove that the four cases SAFE$\langle \sqcap, \odot \rangle$, SAFE$\langle \sqcup, \otimes \rangle$, SAFE$\langle \sqcap, \otimes \rangle$, and SAFE$\langle \odot, \otimes \rangle$ are $\mathbf{NP}$-hard.

Comparing Table II with Table I, we found that the computational complexities of all subcases of SAFE are the same as those of UTS except for the subcase in which only operators in $\{\neg, +, \sqcup, \odot\}$ are allowed. SAFE$\langle \neg, +, \sqcup, \odot \rangle$ is in $\mathbf{P}$, while UTS$\langle \sqcup, \odot \rangle$ is $\mathbf{NP}$-hard. Intuitively, UTS$\langle \sqcup, \odot \rangle$ is computationally more expensive than SAFE $\{\sqcup, \odot\}$

$\neg$	$+$	$\sqcup$	$\sqcap$	$\odot$	$\otimes$	**Complexity**	**Reduction**
✓	✓	✓	✓	✓	✓	**NP**-hard, **coNP**-hard, in $\mathbf{coNP^{NP}}$	
		✓		✓		**coNP**-hard	Validity
			✓	✓		**NP**-hard	SAFE$\langle\sqcap, \odot\rangle$
					✓	**coNP**-complete	Set Covering
✓	✓	✓	✓			**P**	
✓	✓			✓		**P**	

Table III. Various sub-cases of the Static Safety Checking (SSC) problem and the corresponding time-complexity. Time-complexity of all other subcases can be deduced from the subcases shown in the table.

for the following reason: given a term $\phi = (\phi_1 \odot \cdots \odot \phi_m)$ and a userset U, U is safe with respect to ϕ if and only if U is safe with respect to ϕ_i for every $i \in [1, m]$. In other words, for SAFE, one may check whether U is safe with respect to ϕ_i independently from ϕ_j $(i \neq j)$. However, when it comes to UTS, such independency no longer exists and one has to take into account whether every user in U is used to satisfy some ϕ_i in the term ϕ.

6.1 Static Safety Checking (SSC) Problem

Given a high-level security policy $\mathsf{sp}\langle P, \phi\rangle$, the Static Safety Checking (SSC) problem asks whether a given state $\langle U, UR, UP\rangle$ is statically safe with respect to $\mathsf{sp}\langle P, \phi\rangle$. We study the computational complexities of SSC, and consider all subcases where only some subset of the operators in $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed. We show that the general case of SSC is both **NP**-hard and **coNP**-hard and is in $\mathbf{coNP^{NP}}$, which is a complexity class in Polynomial Hierarchy. The proof of the following theorem is given in Appendix E.

THEOREM 18. The computational complexities of SSC and its subcases are given in Table III.

7. DISCUSSIONS

In this section we discuss potential extensions to the syntax of the algebra, the relationship between the algebra and regular expressions, and the limitations of the algebra's expressive power.

7.1 Extensions to the Syntax of the Algebra

In this paper, we have defined the basic operators in the algebra and examined their properties. We now discuss some additional operators that could be added to the algebra as syntactic sugars.

As discussed in Section 2.5, SoD policies are monotonic, as are policies in McLean's formulation of N-person policies [McLean 1988]; our algebra supports both monotonic policies and policies that are not monotonic. To express a monotonic policy that requires a task to be performed by a userset that either satisfies a term ϕ or contains a subset that satisfies ϕ, one can use $(\phi \odot \mathsf{All}^+)$. As monotonic policies may be quite common, we introduce a unary operator $\bigtriangledown$ as a syntactic sugar. That is, $\bigtriangledown\phi$ is defined to be $(\phi \odot \mathsf{All}^+)$.

Besides monotonic policies, another type of policy mentioned in Section 2.5 states that every user involved in a task must satisfy certain requirements and there need to be at least

a certain number of users involved. Let ϕ be a unit term that expresses the requirements. A policy that requires two or more users that satisfy ϕ can be expressed as $((\phi \otimes \phi) \odot \phi^+)$. To simplify the expression of these policies, we define ϕ^{2+} as a syntactic sugar for $((\phi \otimes \phi) \odot \phi^+)$. In general, ϕ^{k+} means that at least k ($k \geq 2$) users are required and every user involved must satisfy ϕ.

Similar to the above, ϕ^k is a syntactic sugar for a term using operator $\otimes$ to connect k unit terms ϕ. For instance, $\texttt{Accountant}^3$ is defined as ($\texttt{Accountant} \otimes \texttt{Accountant} \otimes \texttt{Accountant}$). More generally, ϕ^k states that exactly k users are required and every user involved must satisfy ϕ. Writing a term in ϕ^k rather than $(\phi \otimes \cdots \otimes \phi)$ explicitly states that all the k sub-terms connected together by $\otimes$ are the same. This makes the policy more succinct and easier to process.

7.2 Relationship with Regular Expressions

The syntax of terms in our algebra may remind readers of regular expressions. A regular expression is a string that describes or matches a set of strings, while a term in the algebra is a string that describes or matches a set of sets. Given an alphabet, a regular expression evaluates to *a set of strings*. Given a configuration, a term in our algebra evaluates to *a set of sets*. In the following, we compare our algebra with regular expressions.

For example, the regular expression "$a(b|c)[\hat{}abc]^+$" matches all strings that start with the letter a, followed by either b or c, and then by one or more symbols that are not in $\{a, b, c\}$. A term that is close in spirit to the regular expression is $\{a\} \otimes (\{b\} \sqcup \{c\}) \otimes (\neg\{a, b, c\})^+$, which is satisfied by all sets that contain a, either b or c, and one or more symbols that are not in $\{a, b, c\}$.

From the example, one can draw some analogies between the operators in regular expressions and the ones in our algebra. The operator | in regular expressions is similar to $\sqcup$. Concatenation in regular expression may seem to be related to $\otimes$. One clear difference is that concatenation is order sensitive, whereas $\otimes$ is not, because a string is order sensitive but a set is not. A more subtle difference comes from the property that $\otimes$ requires the two sub-terms be satisfied by disjoint sets. For instance, $\{a\} \otimes \{a\}$ cannot be satisfied by any set. The usage of negation in regular expressions is similar to negation in the algebra; in both cases, negation can be applied only to an expression corresponding to a single element. In regular expression, the closure operator ($*$ or $+$) can be applied to arbitrary sub-expressions. Our algebra requires that repetition (using operator $^+$) can only be applied to unit terms. As we discussed in Section 2.5, since the algebra is proposed for security policy specification, we impose such restriction so as to clearly capture real-word security requirements. If the algebra is used in areas other than security policy specification, it is certainly possible to release such restriction so that the algebra can define a wider range of sets. The remaining binary operators $\odot$ and $\sqcap$ have the flavor of set intersection, which does not have counterparts in regular expressions.

Observe that determining whether a string satisfies a regular expression is in $\mathbf{NL}$-complete, where $\mathbf{NL}$ stands for Nondeterministic Logarithmic-Space, and is contained in $\mathbf{P}$. On the other hand, determining whether a userset satisfies a term is $\mathbf{NP}$-complete, even if the term uses only $\sqcup$ and $\otimes$ or only $\sqcup$ and $\odot$. It appears that this increase in complexity is due to the unordered nature of sets. Checking a string against a regular expression can be performed from the beginning of a string to its end; on the other hand, there is no such order in checking a set against a term in the algebra.

As a fundamental tool for defining sets of strings, regular expression is used in many areas. Analogically, because our algebra is about the fundamental concept of defining sets of sets, we conjecture that, besides expression of security policies, the algebra could be used in other areas where set specification is desired. For example, we may use the algebra to specify some sorts of reaction formula, in which each element must have certain properties and in some cases we may be able to choose among several properties. For another example, our algebra could be used to specify digital-right-management licenses that entitle users to play a set of songs. An example of such licenses is, Alice can play a song in Album A once, and two other songs in either Album B or Album C. Barth and Mitchell studied how to specify such licenses using linear logic in [Barth and Mitchell 2006].

7.3 Limitations of the Algebra's Expressive Power

It is well-known that using regular expression, one cannot express languages that require counting to an unbounded number; for example, one cannot express all strings over the alphabet $\{a, b\}$ that contain the same number of a's as of b's.

Similarly, the algebra as defined in Section 2.1 cannot express a policy that requires a set of users in which the number of members of r_1 equals the number of members of r_2. The proof is similar to that of the Pumping Lemma in regular language. We illustrate the sketch of the proof here. Assume, for the purpose of contradiction, that there exists a term ϕ in the algebra that is satisfied only by usersets with an equal number of members of r_1 and members of r_2. Let X_1 be a userset consisting of n users who are members of r_1 but not r_2, and X_2 be another userset consisting of n users who are members of r_2 but not r_1, where $n > |\phi|$. Let $X = X_1 \cup X_2$. By assumption, X satisfies ϕ. Let T be the satisfaction tree of ϕ, whose root is labeled with X. Since $|X| > |\phi|$, T must have leaves corresponding to sub-terms in the form of ϕ_0^+. Also, since $n > |\phi|$, there must exist a leaf N_1 with $\phi_1 = \phi_2^+$ and $L_T(N_1)$ contains a user $u \in X_1$ that does not appear in usersets labeling leaves without $^+$. We may now "pump" (i.e. add) another m copies of u to every node in T whose associated userset contains u. By following the rules in Definition 4, it can be proved that the tree T', which is acquired from T after pumping, is a satisfaction tree of ϕ. Note that the root of T' is labeled with X', which contains $n + m$ members of r_1 and n members of r_2. According to Theorem 2, X' satisfies ϕ, which is a contradiction to the assumption.

If we allow the application of $^+$ to non-unit terms and define it as follows:

$$\phi^+ \stackrel{def}{=} \phi \sqcup (\phi \otimes \phi) \sqcup (\phi \otimes \phi \otimes \phi) \sqcup \dots$$

then we can express the policy that requires an equal number of members of r_1 and members of r_2 using the term

$$[(r_1 \sqcap r_2) \sqcup ((r_1 \sqcap \neg r_2) \otimes (r_2 \sqcap \neg r_1)) \sqcup (\neg r_1 \sqcap \neg r_2)]^+$$

Note that the subterm $((r_1 \sqcap \neg r_2) \otimes (r_2 \sqcap \neg r_1))$ matches one user who is a member of r_1 but not r_2 with a user who is a member of r_2 but not r_1.

Even with the extension, however, there are sets of usersets that cannot be expressed. For example, one cannot express a policy that requires that the number of users who are r_1 equals the square of the number of users who are r_2. [2] Further discussions of expressive

[2] Intuitively, since $^+$ does not record the number of users, there is no way for a term to compute the square of the

power and more general algebras are interesting future research topics and are beyond the scope of this paper.

8. RELATED WORK

The concept of SoD has long existed in the physical world, sometimes under the name "the two-man rule", for example, in the banking industry and the military. To our knowledge, in the information security literature the notion of SoD first appeared in Saltzer and Schroeder [Saltzer and Schroeder 1975] under the name "separation of privilege." Clark and Wilson's commercial security policy for integrity [Clark and Wilson 1987] identified SoD along with well-formed transactions as two major mechanisms of fraud and error control. Nash and Poland [Nash and Poland 1990] explained the difference between dynamic and static enforcement of SoD policies. In the former, a user may perform any step in a sensitive task provided that the user does not also perform another step on that task. In the latter, users are constrained a-priori from performing certain steps.

Sandhu [Sandhu 1990; 1988] presented Transaction Control Expressions, a history-based mechanism for dynamically enforcing SoD policies. A transaction control expression associates each step in the transaction with a role. By default, the requirement is such that each step must be performed by a different user. One can also specify that two steps must be performed by the same user. In Transaction Control Expressions, user qualification requirements are associated with individual steps in a transaction, rather than a transaction as a whole.

Li et al [Li et al. 2007] studied both direct and indirect enforcement of static separation of duty (SSoD) policies. They showed that directly enforcing SSoD policies is intractable ($\mathbf{NP}$-complete). They also discussed using static mutually exclusive roles (SMER) constraints to indirectly enforce SSoD policies. They defined what it means for a set of SMER constraints to precisely enforce an SSoD policy, characterize the policies for which such constraints exist, and show how they are generated. In Section 3, we study the enforcement of policies specified in our algebra, which include SoD policies as a sub-class; however, our computational results (those on SSC) are on direct static enforcement only.

There exists a wealth of literature on constraints in the context of RBAC [Ahn and Sandhu 1999; 2000; Crampton 2003; Gligor et al. 1998; Jaeger 1999; Jaeger and Tidswell 2001; Simon and Zurko 1997; Tidswell and Jaeger 2000]. They either proposed and classified new kinds of constraints [Gligor et al. 1998; Simon and Zurko 1997] or proposed new languages for specifying sophisticated constraints [Ahn and Sandhu 1999; 2000; Crampton 2003; Jaeger and Tidswell 2001; Tidswell and Jaeger 2000]. Most of these constraints are motivated by SoD and are variants of role mutual exclusion constraints, which may declare two roles to be mutually exclusive so that no user can be a member of both roles.

Workflow systems have been widely studied in the literature as well. Atluri and Huang [Atluri and Huang 1996] proposed an access control model and temporal constraints for workflow environments. Bertino et al. [Bertino et al. 1999] proposed a language for specifying static and dynamic constraints for separation of duty in role-based workflow systems. Other workflow models have been proposed in [Atluri and Warner 2005; Crampton 2005; Tan et al. 2004; Wang and Li 2007]. In these workflow models, steps are authorized to roles and security requirements are enforced by inter-step constraints. Example constraints are "Step 1 and Step 2 must be performed by different users" and "Step

number of users in a userset.

2 must be performed by the manager of the person who performed Step 1". These models contribute to the study of low-level enforcement mechanism in security design of business process, while our paper focuses on high-level security policy design.

McLean [McLean 1988] introduced a framework that includes various mandatory access control models. Security models are instances of the framework; and they differ in which users are allowed to change the security levels. These models form a boolean algebra. McLean also looked at the issue of N-person policies, where a policy may allow multiple subjects acting together to perform some action. McLean adopted the monotonicity requirement in such N-person policies. McLean [McLean 1988] does not discuss how to specify N-person policies, and the examples in the paper list explicitly the usersets that are allowed access. Our algebra, on the other hand, is about how to define policies that require multiple users with qualification requirements.

Abadi et al. [Abadi et al. 1993] developed a calculus for access control in distributed systems. The calculus allows compound principals to be formed from basic ones using two operations $\wedge$ (and) and $|$ (quoting). Some principals are groups, when a principal u is a member of a group g, then u speaks for g. One can express multi-user policies in this calculus. An access control policy is specified as an access control list (ACL), where each entry is an expression in the calculus. The $\wedge$ corresponds to $\odot$ in our algebra. That is, if an ACL entry contains $g_1 \wedge g_2$, then a single user that is a member of both g_1 and g_2 is allowed access, as are two users such that one is a member of g_1 and the other is a member of g_2. The $\sqcup$ operator in our algebra can be partially supported in the calculus by having multiple ACL entries, which has the effect of supporting logical OR, but only at the top level. The other operators $\neg$, $^+$, $\sqcap$ and $\otimes$ cannot be expressed in the calculus.

Several algebras have been proposed for combining security policies. These include the work by Bonatti et al. [Bonatti et al. 2000; 2002], Wijesekera and Jajodia [Wijesekera and Jajodia 2003], Pincus and Wing [Pincus and Wing 2005]. These algebras are designed for purpose that are different from ours; therefore, they are quite different from our algebra. Each element in their algebra is a policy that specifies what subjects are allowed to access which resources, whereas each element in our algebra maps to a user.

The two operators $\odot$ and $\otimes$ in our algebra are taken from the RT family of role-based trust-management languages designed by Li et al. [Li et al. 2002]. In [Li et al. 2002], the notion of manifold roles was introduced, which are roles that have usersets, rather than individual users, as their members. The two operators $\otimes$ and $\odot$ are used to define manifold roles. This paper differs in that we propose to combine these two operators together with four other operators $\sqcup$, $\sqcap$, $\neg$, and $+$ (which are not in RT) in an algebra for specifying high-level security policies. In addition, we also study the algebraic properties of these operators, the satisfaction problems, and the term satisfiability problem related to the algebra.

Readers who are familiar with description logic (DL) may find similarities between the algebra and DL. However, there is a fundamental difference between the two: a term in DL describes *a set of individuals*, while a term in the algebra describes *a set of sets of individuals*. A concept in DL defines a set of individuals, which corresponds to a role in the algebra; a "role" in DL defines a binary relation between individuals. DL supports operators $\neg, \sqcap$ and $\sqcup$, which stand for complement of concepts, intersection of concepts and union of concepts, respectively. If we interpret a unit term in our algebra as a set of

individuals[3], then unit terms may be viewed as a strict subset of terms in DL. But in general case, there is no operator in DL that corresponds to operators $^+$, $\odot$ and $\otimes$ in our algebra. Hence, computational complexity problems studied in this paper are not directly related to those in DL.

9. SUMMARY

While separation of duty policies are extremely important and widely used, they state only quantity requirements and cannot capture qualification requirements on users involved in the task. We have introduced a novel algebra that enables the specification of high-level policies that combine qualification requirements with quantity requirements motivated by separation of duty considerations. Our algebra has two unary and four binary operators, and is expressive enough to specify a large number of diverse policies. We have studied algebraic properties of these operators and discussed low-level enforcement mechanisms for high-level policies, such as static enforcement and dynamic enforcement. Furthermore, several computational problems related to the algebra have been studied. Finally, as our algebra is about the general concept of sets of sets, we conjecture that it will prove to be useful in other contexts as well.

REFERENCES

ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. 1993. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems 15,* 4 (Oct.), 706–734.

AHN, G.-J. AND SANDHU, R. S. 1999. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the 4th Workshop on Role-Based Access Control*. 43–54.

AHN, G.-J. AND SANDHU, R. S. 2000. Role-based authorization constraints specification. *ACM Transactions on Information and System Security 3,* 4 (Nov.), 207–226.

ATLURI, V. AND HUANG, W. 1996. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS)*. 44–64.

ATLURI, V. AND WARNER, J. 2005. Supporting conditional delegation in secure workflow management systems. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*. ACM Press, New York, NY, USA, 49–58.

BERTINO, E., FERRARI, E., AND ATLURI, V. 1999. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security 2,* 1 (Feb.), 65–104.

BARTH, A. AND MITCHELL, J. 2006. Managing Digital Rights using Linear Logic. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS)*. 127–136.

BONATTI, P., DE CAPITANI DI VIMERCATI, S., AND SAMARATI, P. 2000. A modular approach to composing access control policies. In *Proc. ACM Conference on Computer and Communications Security (CCS)*. 164–173.

BONATTI, P., DE CAPITANI DI VIMERCATI, S., AND SAMARATI, P. 2002. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC) 5,* 1 (Feb.), 1–35.

CLARK, D. D. AND WILSON, D. R. 1987. A comparision of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 184–194.

CRAMPTON, J. 2003. Specifying and enforcing constraints in role-based access control. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*. Como, Italy, 43–50.

CRAMPTON, J. 2005. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies (SACMAT 2005)*. Stockholm, Sweden, 38–47.

[3]A unit term in our algebra describes a set of singletons.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers And Intractability*. W. H. Freeman.

GLIGOR, V. D., GAVRILA, S. I., AND FERRAIOLO, D. F. 1998. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of IEEE Symposium on Research in Security and Privacy*. 172–183.

JAEGER, T. 1999. On the increasing importance of constraints. In *Proc. ACM Workshop on Role-Based Access Control (RBAC)*. 33–42.

JAEGER, T. AND TIDSWELL, J. E. 2001. Practical safety in flexible access control models. *ACM Transactions on Information and System Security 4,* 2 (May), 158–190.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2002. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 114–130.

LI, N., TRIPUNITARA, M. V., AND BIZRI, Z. 2007. On mutually exclusive roles and separation-of-duty. *ACM Transactions on Information and Systems Security (TISSEC) 10,* 2, 5.

MCLEAN, J. 1988. The algebra of security. In *Proceedings of IEEE Symposium on Security and Privacy*. 2–7.

NASH, M. J. AND POLAND, K. R. 1990. Some conundrums concerning separation of duty. In *Proceedings of IEEE Symposium on Research in Security and Privacy*. 201–209.

PAPADIMITTIOU, C. H. AND STEIGLITZ, K. 1982. *Combinatorial Optimization*. Prentice-Hall.

PINCUS, J. AND WING, J. M. 2005. Towards an algebra for security policies (extended abstract). In *Proceedings of ICATPN 2005*. Number 3536 in LNCS. Springer, 17–25.

SALTZER, J. H. AND SCHROEDER, M. D. 1975. The protection of information in computer systems. *Proceedings of the IEEE 63,* 9 (September), 1278–1308.

SANDHU, R. 1990. Separation of duties in computerized information systems. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*.

SANDHU, R. S. 1988. Transaction control expressions for separation of duties. In *Proceedings of the Fourth Annual Computer Security Applications Conference (ACSAC'88)*.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Computer 29,* 2 (February), 38–47.

SIMON, T. T. AND ZURKO, M. E. 1997. Separation of duty in role-based environments. In *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 183–194.

TAN, K., CRAMPTON, J., AND GUNTER, C. 2004. The consistency of task-based authorization constraints in workflow systems. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW)*. 155–169.

TIDSWELL, J. AND JAEGER, T. 2000. An access control model for simplifying constraint expression. In *Proceedings of ACM Conference on Computer and Communications Security*. 154–163.

WANG, Q. AND LI, N. 2007. Satisfiability and resiliency in workflow systems. In *Proceedings of the 12th European Symposium on Research in Computer Security (ESORICS)*. 90–105.

WIJESEKERA, D. AND JAJODIA, S. 2003. A propositional policy algebra for access control. *ACM Transactions on Information and Systems Security (TISSEC) 6,* 2 (May), 286–325.

A. PROOFS OF THEOREMS IN SECTION 2

Proof of Lemma 1

Given a node N in T, let $\Phi(N)$ be the sub-term of ϕ represented by the sub-tree rooted at N. In the following, we prove by induction that for every node N, if $L_T(N) \neq \emptyset$, then $L_T(N)$ satisfies $\Phi(N)$.

Base case: When N is a leaf node, by Definition 4, $L_T(N)$ is either $\emptyset$ or it satisfies $\Phi(N)$. Since $L_T(N) \neq \emptyset$, $L_T(N)$ satisfies $\Phi(N)$.

Inductive case: Assume that the statement holds for both children N_1 and N_2 of N and $L_T(N) \neq \emptyset$.

—When N represents $\sqcap$: According to Definition 4, $L_T(N) = L_T(N_1) = L_T(N_2)$. Since $L_T(N) \neq \emptyset$, $L_T(N_1)$ and $L_T(N_2)$ are non-empty. By inductive assumption, $L_T(N_1)$

and $L_T(N_2)$ satisfy $\Phi(N_1)$ and $\Phi(N_2)$, respectively. Since $\Phi(N) = \Phi(N_1) \sqcap \Phi(N_2)$, by Definition 3, $L_T(N)$ satisfies $\Phi(N)$.

—When N represents $\sqcup$: According to Definition 4, $L_T(N) = L_T(N_1)$ or $L_T(N) = L_T(N_2)$. Without loss of generality, assume that $L_T(N) = L_T(N_1)$. Since $L_T(N) \neq \emptyset$, $L_T(N_1)$ is non-empty. By inductive assumption, $L_T(N_1)$ satisfies $\Phi(N_1)$. Since $\Phi(N) = \Phi(N_1) \sqcup \Phi(N_2)$, by Definition 3, $L_T(N)$ satisfies $\Phi(N)$.

—When N represents $\odot$: According to Definition 4, $L_T(N) = L_T(N_1) \cup L_T(N_2)$, and since $L_T(N) \neq \emptyset$, $L_T(N_1)$ and $L_T(N_2)$ are also non-empty. By inductive assumption, $L_T(N_1)$ and $L_T(N_2)$ satisfy $\Phi(N_1)$ and $\Phi(N_2)$, respectively. Since $\Phi(N) = \Phi(N_1) \sqcap \Phi(N_2)$, by Definition 3, $L_T(N)$ satisfies $\Phi(N)$.

—When N represents $\otimes$: This is very similar to the above case.

Proof of Theorem 2

Clearly, if there exists a satisfaction tree of ϕ with root labeled X, then X satisfies ϕ. Now we show the other direction. If a userset X satisfies ϕ under $\langle U, UR \rangle$, we construct a satisfaction tree for ϕ. First of all, we construct the syntax tree T of ϕ and label its root with X. We then recursively label other nodes in T in a top-down manner. Let N be an inner node labeled with a non-empty userset. We label the children N_1 and N_2 of N in the following manner.

—When N represents $\sqcap$: We label N_1 and N_2 with $L_T(N)$. This satisfies the rules specified in Definition 4. Since $L_T(N)$ satisfies $\Phi(N)$ and $\Phi(N) = \Phi(N_1) \sqcap \Phi(N_2)$, we have $L_T(N_1) = L_T(N)$ satisfies $\Phi(N_1)$ and $L_T(N_2) = L_T(N)$ satisfies $\Phi(N_2)$.

—When N represents $\sqcup$: Since $L_T(N)$ satisfies $\Phi(N)$ and $\Phi(N) = \Phi(N_1) \sqcup \Phi(N_2)$, either $L_T(N)$ satisfies $\Phi(N_1)$ or $L_T(N)$ satisfies $\Phi(N_2)$. Without loss of generality, assume that $L_T(N)$ satisfies $\Phi(N_1)$. We label N_1 with $L_T(N)$ and N_2 with $\emptyset$. We also label all the nodes in the sub-tree rooted at N_2 with $\emptyset$. This satisfies the rules specified in Definition 4.

—When N represents $\odot$: Since $L_T(N)$ satisfies $\Phi(N)$ and $\Phi(N) = \Phi(N_1) \odot \Phi(N_2)$, according to Definition 3, we have non-empty sets X_1 and X_2 such that $L_T(N) = X_1 \cup X_2$ and X_1 satisfies $\Phi(N_1)$ and X_2 satisfies $\Phi(N_2)$. We label N_1 with X_1 and N_2 with X_2. Since $L_T(N) = X_1 \cup X_2$, the labeling satisfies the rules specified in Definition 4.

—When N represents $\otimes$: we label it in ways similar to the above case.

According to the above, when X satisfies ϕ, we can construct a satisfaction tree whose root is labeled with X.

Proof of Theorem 3 on Algebraic Properties

(1) The operators $\sqcup, \sqcap, \otimes, \odot$ are commutative and associative.
This is straightforward from Definition 3.

(2) The operator $\sqcup$ distributes over $\sqcap$.
If a userset X satisfies $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3))$, then either X satisfies ϕ_1, or X satisfies both ϕ_2 and ϕ_3. It follows that X satisfies $((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$.
If X satisfies $((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$, then X satisfies $(\phi_1 \sqcup \phi_2)$ and $(\phi_1 \sqcup \phi_3)$. There are only two cases: (1) X satisfies ϕ_1; and (2) X satisfies both ϕ_2 and ϕ_3. In either case, X satisfies $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3))$.

The operator $\sqcap$ distributes over $\sqcup$.
If X satisfies $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3))$, then X satisfies both ϕ_1 and $(\phi_2 \sqcup \phi_3)$, which means X satisfies either ϕ_2 or ϕ_3. It follows that X satisfies $((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$.
If X satisfies $((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$, then either (1) X satisfies $(\phi_1 \sqcap \phi_2)$ or (2) X satisfies $(\phi_1 \sqcap \phi_3)$. In both cases, X satisfies ϕ_1; furthermore, X satisfies either ϕ_2 or ϕ_3. It follows that X satisfies $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3))$.

(3) The operator $\odot$ distributes over $\sqcup$.
If X satisfies $(\phi_1 \odot (\phi_2 \sqcup \phi_3))$, then there exist X_1 and X_2 such that $X_1 \cup X_2 = X$, X_1 satisfies ϕ_1, and X_2 satisfies $(\phi_2 \sqcup \phi_3)$. By Definition 3, X_2 satisfies ϕ_2 or ϕ_3. In the former case, X satisfies $(\phi_1 \odot \phi_2)$, which implies that X satisfies $((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$, as desired. The argument is analogous if X_2 satisfies ϕ_3 but not ϕ_2.
If X satisfies $((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$, then either X satisfies $(\phi_1 \odot \phi_2)$ or X satisfies $(\phi_1 \odot \phi_3)$. Without loss of generality, assume that X satisfies $(\phi_1 \odot \phi_2)$, then there exist X_1, X_2 such that $X_1 \cup X_2 = X$, X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. Therefore, X_2 satisfies $(\phi_2 \sqcup \phi_3)$, and consequently, X satisfies $(\phi_1 \odot (\phi_2 \sqcup \phi_3))$ as desired.

(4) The operator $\otimes$ distributes over $\sqcup$.
If X satisfies $(\phi_1 \otimes (\phi_2 \sqcup \phi_3))$, X can be partitioned into two disjoint sets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2 or ϕ_3. In this case, by definition, X satisfies $(\phi_1 \otimes \phi_2)$ or $(\phi_1 \otimes \phi_3)$, which means X satisfies $((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$.
For the other direction, if X satisfies $((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$, it satisfies either $(\phi_1 \otimes \phi_2)$ or $(\phi_1 \otimes \phi_3)$. Without loss of generality, assume that X satisfies $(\phi_1 \otimes \phi_2)$. Then, X can be partitioned into two disjoint sets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. By definition, X_2 satisfies $(\phi_2 \sqcup \phi_3)$. Therefore, X satisfies $(\phi_1 \otimes (\phi_2 \sqcup \phi_3))$.

(5) No other ordered pair of operators have the distributive property.
We show a counterexample for each case. In the following, $U_r = \{u | (u, r) \in UR\}$.

(a) The operator $\odot$ does not distribute over $\sqcap$.
If X satisfies $(\phi_1 \odot (\phi_2 \sqcap \phi_3))$, then X also satisfies $((\phi_1 \odot \phi_2) \sqcap (\phi_1 \odot \phi_3))$.
However, the other direction of implication does not hold. Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$, and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \odot r_2) \sqcap (r_1 \odot r_3))$, but does not satisfy $(r_1 \odot (r_2 \sqcap r_3))$.

(b) The operator $\sqcap$ does not distribute over $\odot$. Neither direction holds.
Counterexample: Let $U_{r_1} = U_{r_3} = \{u_1\}$ and $U_{r_2} = U_{r_4} = \{u_2\}$, let $\phi_1 = (r_1 \odot r_2)$, then $\{u_1, u_2\}$ satisfies $(\phi_1 \sqcap (r_3 \odot r_4))$, but does not satisfy $((\phi_1 \sqcap r_3) \odot (\phi_1 \sqcap r_4))$.
Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$, and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcap r_2) \odot (r_1 \sqcap r_3))$, but does not satisfy $(r_1 \sqcap (r_2 \odot r_3))$.

(c) The operator $\sqcup$ does not distribute over $\odot$.
If X satisfies $(\phi_1 \sqcup (\phi_2 \odot \phi_3))$, then X satisfies $((\phi_1 \sqcup \phi_2) \odot (\phi_1 \sqcup \phi_3))$.
However, the other direction of implication does not hold. Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_1\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcup r_2) \odot (r_1 \sqcup r_3))$, but does not satisfy $(r_1 \sqcup (r_2 \odot r_3))$.

(d) The operator $\sqcup$ does not distribute over $\otimes$. Neither direction holds.
Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_1\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcup r_2) \otimes (r_1 \sqcup r_3))$, but does not satisfy $(r_1 \sqcup (r_2 \otimes r_3))$.

Counterexample: Let $U_{r_1} = U_{r_2} = U_{r_3} = \{u_1\}$, then $\{u_1\}$ satisfies $(r_1 \sqcup (r_2 \otimes r_3))$, but does not satisfy $((r_1 \sqcup r_2) \otimes (r_1 \sqcup r_3))$.

(e) The operator $\otimes$ does not distribute over $\sqcap$.
If X satisfies $(\phi_1 \otimes (\phi_2 \sqcap \phi_3))$, then X satisfies $((\phi_1 \otimes \phi_2) \sqcap (\phi_1 \otimes \phi_3))$.
However, the other direction of implication does not hold. Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \otimes r_2) \sqcap (r_1 \otimes r_3))$, but does not satisfy $(r_1 \otimes (r_2 \sqcap r_3))$.

(f) The operator $\sqcap$ does not distribute over $\otimes$. Neither direction holds.
Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcap r_2) \otimes (r_1 \sqcap r_3))$, but does not satisfy $(r_1 \otimes (r_2 \sqcap r_3))$.
Counterexample: Let $U_{r_1} = U_{r_3} = \{u_1\}$ and $U_{r_2} = U_{r_4} = \{u_2\}$, and let $\phi_1 = (r_1 \odot r_2)$, then $\{u_1, u_2\}$ satisfies $(\phi_1 \sqcap (r_3 \otimes r_4))$, but does not satisfy $((\phi_1 \sqcap r_3) \otimes (\phi_1 \sqcap r_4))$.

(g) The operator $\odot$ does not distribute over $\otimes$. Neither direction holds.
Counterexample: Let $U_{r_1} = \{u_1, u_4\}$, $U_{r_2} = \{u_2\}$ and $U_{r_3} = \{u_3\}$, then $\{u_1, u_2, u_3, u_4\}$ satisfies $((r_1 \odot r_2) \otimes (r_1 \odot r_3))$, but does not satisfies $(r_1 \odot (r_2 \otimes r_3))$.
Counterexample: Let $U_{r_1} = \{u_1\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $(r_1 \odot (r_2 \otimes r_3))$, but does not satisfy $((r_1 \odot r_2) \otimes (r_1 \odot r_3))$.

(h) The operator $\otimes$ does not distribute over $\odot$.
If X satisfies $(\phi_1 \otimes (\phi_2 \odot \phi_3))$, then X satisfies $((\phi_1 \otimes \phi_2) \odot (\phi_1 \otimes \phi_3))$.
However, the other direction of implication does not hold. Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_2\}$ and $U_{r_3} = \{u_1\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \otimes r_2) \odot (r_1 \otimes r_3))$, but does not satisfy $(r_1 \otimes (r_2 \odot r_3))$.

(6) $(\phi_1 \sqcap \phi_2)^+ \equiv (\phi_1^+ \sqcap \phi_2^+)$.
If a userset X satisfies $(\phi_1 \sqcap \phi_2)^+$, then for every $u \in X$, $\{u\}$ satisfies $(\phi_1 \sqcap \phi_2)$ and thus satisfies ϕ_1 and ϕ_2. Hence, X satisfies ϕ_1^+ and ϕ_2^+, which means that X satisfies $(\phi_1^+ \sqcap \phi_2^+)$.
If X satisfies $(\phi_1^+ \sqcap \phi_2^+)$, then X satisfies both ϕ_1^+ and ϕ_2^+. For every $u \in X$, $\{u\}$ satisfies both ϕ_1 and ϕ_2. Hence, X satisfies $(\phi_1 \sqcap \phi_2)^+$.

(7) DeMorgan's Law: $\neg(\phi_1 \sqcap \phi_2) \equiv (\neg\phi_1 \sqcup \neg\phi_2)$, $\neg(\phi_1 \sqcup \phi_2) \equiv (\neg\phi_1 \sqcap \neg\phi_2)$
The proof is straightforward by definition of $\neg$, $\sqcap$ and $\sqcup$.

B. PROOFS OF THEOREMS IN SECTION 4

In the following proofs, $(\mathsf{op}_k \phi)$ denotes k copies of ϕ connected together by operator op and $(\mathsf{op}_{i=1}^{n} r_i)$ denotes $(r_1\ \mathsf{op} \ldots \mathsf{op}\ r_n)$. Given $R = \{r_1, \ldots, r_m\}$, $(\mathsf{op} R)$ denotes $(r_1\ \mathsf{op} \ldots \mathsf{op}\ r_m)$.

B.1 Proof of Lemma 4, Lemma 5, and Theorem 7

Proof of Lemma 4

To prove that TSAT over terms built using only roles, $\neg$, $\sqcap$, and $\sqcup$ is NP-hard, we reduce the **NP**-complete SAT problem to it. Given a propositional logic formula e, let $\{v_1, \ldots, v_n\}$ be the set of propositional variables that appear in e. Construct a term ϕ by substituting every occurrence of v_i $(i \in [1, n])$ in e with the atomic term r_i, every occurrence of $\neg v_i$ $(i \in [1, n])$ with $\neg r_i$, and replacing logical AND with $\sqcap$ and logical OR with $\sqcup$. The result is a unit term. By Definition 3, a term without $\odot, \otimes$ and $^+$ can be

satisfied by singletons only. If ϕ is satisfiable, then there exists a configuration $\langle U, UR\rangle$ and a user u such that $\{u\}$ satisfies ϕ. We can construct a truth assignment T in which v_i is TRUE if and only if $(u, r_i) \in UR$. It is clear that e evaluates to TRUE under T. Similarly, if there exists a truth assignment T such that e evaluates to TRUE under T, we can construct UR in which u is a member of r_i if and only if v_i is TRUE in T. In that case, $\{u\}$ satisfies ϕ under $\langle U, UR\rangle$. Therefore, e is satisfiable if and only if ϕ is satisfiable.

Proof of Lemma 5

To prove that TSAT over terms built using only explicit sets of users, $\sqcap$, $\sqcup$, and $\odot$ is **NP**-hard, we reduce the **NP**-complete SET COVERING problem to it. In the SET COVERING problem, we are given a finite set $U = \{u_1, \ldots, u_n\}$, a family $F = \{U_1, \ldots, U_m\}$ of subsets of U, and an integer k no larger than m, and we ask whether there is a sub-family $F' \subseteq F$ of sets whose union is U and $|F'| \leq k$.

We view each element in U as a user. For every $j \in [1, m]$, we construct a term $\phi_j = \bigodot\{\{u_i\} \mid u_i \in U_j\}$; that is, $\phi_j = \{u_{j_1}\} \odot \{u_{j_2}\} \odot \cdots \odot \{u_{j_x}\}$, where $U_j = \{u_{j_1}, u_{j_2}, \ldots, u_{j_x}\}$. It is clear that ϕ_j can only be satisfied by U_j. Finally, we construct a term $\phi = ((\bigodot_k(\bigsqcup_{i=1}^m \phi_i)) \sqcap (\bigodot_{i=1}^n \{u_i\}))$. Since $(\bigodot_{i=1}^n \{u_i\})$ can be satisfied only by U, U is the only userset that may satisfy ϕ.

We now demonstrate that ϕ is satisfiable if and only if there are no more than k sets in family F whose union is U. On the one hand, if ϕ is satisfiable, then it must be satisfied by U. In this case, U satisfies $(\bigodot_k(\bigsqcup_{i=1}^m \phi_i))$, which means that there exist k sets $U'_1, \ldots, U'_k$ such that $\bigcup_{i=1}^k U'_i = U$ and each U'_i satisfies $(\bigsqcup_{i=1}^m \phi_i)$. Since ϕ_i can be satisfied only by $U_i \in F$, we have $U'_j \in F$ for every $j \in [1, k]$. The answer to the SET COVERING problem is thus "yes". On the other hand, without loss of generality, assume that $\bigcup_{i=1}^k U_i = U$. We have, for every $i \in [1, k]$, U_i satisfies ϕ_i and thus satisfies $(\bigsqcup_{i=1}^m \phi_i)$. Therefore, U satisfies $(\bigodot_k(\bigsqcup_{i=1}^m \phi_i))$. Since U also satisfies $(\bigodot_{i=1}^n \{u_i\})$, U satisfies $((\bigodot_k(\bigsqcup_{i=1}^m \phi_i)) \sqcap (\bigodot_{i=1}^n \{u_i\}))$.

Proof of Lemma 6

First, assume that a userset X satisfies ϕ under $\langle U, UR\rangle$. According to Theorem 2, there exists a satisfaction tree T of ϕ under $\langle U, UR\rangle$ and $L_T(N_r) = X$. Now, we show that if $|X| > |\phi|$, then there must exist $X' \subseteq X$ such that X' satisfies ϕ under $\langle U, UR\rangle$ and $|X'| \leq |\phi|$. In the following, we construct a satisfaction tree T' of ϕ based on T.

Initially, $X' = \emptyset$. For every leaf node N_i of T, if $L_T(N_i) \neq \emptyset$, then we arbitrarily select $u \in L_T(N_i)$ and add u to X'. Since the number of leaves in T is no larger than $|\phi|$, we have $|X'| \leq |\phi|$. Also, $X' \subseteq X$ because $L_T(N_i) \subseteq L_T(N_r) = X$ according to Definition 4. Next, for every node N in T, we relabel N with $L_{T'}(N)$ such that $L_{T'}(N) = L_T(N) \cap X'$. When the relabeling is done, we acquire a new tree T'. In particular, the root of T' is labeled with $X \cap X' = X'$. Now, we show that T' is a satisfaction tree by proving that it satisfies the conditions in Definition 4. Given a node N in T, we denote $\Phi(N)$ as the sub-term of ϕ that is represented by the sub-tree rooted at N. When $L_T(N) = \emptyset$, $L_{T'}(N) = \emptyset$. In the following, we only discuss the cases when $L_T(N) \neq \emptyset$.

—When N is a leaf node: If $\Phi(N)$ is a unit term, then $L_T(N)$ must be a singleton and the only user in $L_T(N)$ must have been added to X'. Thus, we have $L_{T'}(N) = L_T(N)$ which satisfies $\Phi(N)$. Otherwise, $\Phi(N)$ is in the form of ϕ_1^+. $L_T(N)$ satisfying ϕ_1^+ indicates that every user in $L_T(N)$ satisfies ϕ_1. Since at least one user in $L_T(N)$ has

been added to X', $L_{T'}(N) = L_T(N) \cap X'$ is a non-empty subset of $L_T(N)$. Therefore, $L_{T'}(N)$ satisfies ϕ_1^+.

—When N represents $\sqcap$: Because $L_T(N) = L_T(N_1)$, we have $L_{T'}(N) = L_T(N) \cap X' = L_T(N_1) \cap X' = L_{T'}(N_1)$. Similarly, $L_T(N) = L_T(N_2)$ implies that $L_{T'}(N) = L_{T'}(N_2)$.

—When N represents $\sqcup$: If $L_T(N) = L_T(N_1)$, we have $L_{T'}(N) = L_T(N) \cap X' = L_T(N_1) \cap X' = L_{T'}(N_1)$. Otherwise, if $L_T(N) = L_T(N_2)$, we can prove similarly that $L_{T'}(N) = L_{T'}(N_2)$. Therefore, $L_{T'}(N) = L_{T'}(N_1)$ or $L_{T'}(N) = L_{T'}(N_2)$.

—When N represents $\odot$: Because $L_T(N) = L_T(N_1) \cup L_T(N_2)$, we have $L_{T'}(N) = L_T(N) \cap X' = (L_T(N_1) \cup L_T(N_2)) \cap X' = (L_T(N_1) \cap X') \cup (L_T(N_2) \cap X') = L_{T'}(N_1) \cup L_{T'}(N_2)$.

—When N represents $\otimes$: Similar to the above, we have $L_T(N) = L_{T'}(N_1) \cup L_{T'}(N_2)$. Also, $L_T(N_1) \cap L_T(N_2) = \emptyset$ indicates that $L_{T'}(N_1) \cap L_{T'}(N_2) = \emptyset$.

Therefore, T' is a satisfaction tree for ϕ. And since the root of T is labeled with X', X' satisfies ϕ according to Theorem 2.

According to the above argument, if ϕ is satisfiable, then there exists a set X' of no more than $|\phi|$ users and a configuration $\langle U, UR\rangle$, such that X' satisfies ϕ under $\langle U, UR\rangle$. Users not in X' can be removed from the configuration without affecting the satisfaction of ϕ. Also, those roles in UR that do not appear in ϕ can be removed too. Since there are no more than $|\phi|$ roles in ϕ and there are no more than $|\phi|$ users in X', we have $|UR| \leq |\phi|^2$. Therefore, the lemma holds.

Proof of Theorem 7

Since we have already proved that certain subcases of TSAT are $\mathbf{NP}$-hard, to prove the theorem, we just need to show that the problem is in $\mathbf{NP}$. Given a term ϕ, a nondeterministic Turing machine may guess a configuration $\langle U, UR\rangle$, a userset X, and a satisfaction tree T whose root is labeled with X. According to Lemma 6, the size of X and $\langle U, UR\rangle$ is bounded by $|\phi|^2$. Also, according to Theorem 2, X satisfies ϕ if and only if there is a satisfaction tree of ϕ whose root is labeled with X. There are no more than $2|\phi| - 1$ nodes in T and the size of the set labeling a node is bounded by $|X|$. Therefore, the size of T is polynomial in the size of input. The Turing machine may verify whether T is a satisfaction tree by following the rules specified in Definition 4. It is clear that the verification can be done in polynomial time by following the structure of T. Therefore, TSAT is in $\mathbf{NP}$.

B.2 Proof of Lemma 10, Lemma 11, and Theorem 12

Proof of Lemma 10

Proof by induction on the structure of term ϕ.

Base case: When $\phi = r$ or $\phi = \mathsf{All}$, we have $C(\phi) = \{1\} \subseteq \{1, 2, \ldots, |\phi|\}$. Otherwise, when ϕ is in the form of ϕ_1^+ where ϕ_1 is a unit term, according to Definition 14, we have $C(\phi) = \{i | i \in [1, \infty)\} = W \cup \{|\phi| + 1, |\phi| + 2, \ldots\}$, where $W = \{1, 2, \ldots, |\phi|\}$.

Inductive case: When ϕ is in the form of $(\phi_1 \ \mathsf{op} \ \phi_2)$, assume that the lemma holds for ϕ_1 and ϕ_2. Let W_1 denote a subset of $\{1, 2, \ldots, |\phi_1|\}$ and W_2 denote a subset of $\{1, 2, \ldots, |\phi_2|\}$. We have the following three cases:

Case 1: Both $C(\phi_1)$ and $C(\phi_2)$ are finite. Let $C(\phi_1) = W_1$ and $C(\phi_2) = W_2$. Since $|\phi| = |\phi_1| + |\phi_2|$, it follows from Definition 14 that $C(\phi) \subseteq \{1, 2, \ldots, |\phi|\}$, because for

any $c_1 \in C(\phi_1)$ and $c_2 \in C(\phi_2)$, $c_1 + c_2 \leq |\phi_1| + |\phi_2| = |\phi|$.

Case 2: Exactly one of $C(\phi_1)$ and $C(\phi_2)$ is an infinite set. Without loss of generality, assume that $C(\phi_1) = W_1$ and $C(\phi_2) = W_2 \cup \{|\phi_2|+1, |\phi_2|+2, \dots\}$. We compute $C(\phi)$ according to op:

— $\mathsf{op} = \sqcup$: $C(\phi) = C(\phi_1) \cup C(\phi_2) = W_1 \cup W_2 \cup \{|\phi_2|+1, |\phi_2|+2, \dots\} = W_1 \cup W_2 \cup \{|\phi_2|+1, \dots, |\phi|\} \cup \{|\phi|+1, |\phi|+2, \dots\}$, in which $W_1 \cup W_2 \cup \{|\phi_2|, \dots, |\phi|\}$ is a subset of $\{1, 2, \dots, |\phi|\}$.

— $\mathsf{op} = \sqcap$: $C(\phi) = C(\phi_1) \cap C(\phi_2)$ is a subset of W_1, which is a subset of $\{1, 2, \dots, |\phi|\}$.

— $\mathsf{op} = \odot$:

$$\begin{aligned} C(\phi) =& \{i \mid \exists c_1 \in W_1\ \exists c_2 \in W_2\ [max(c_1, c_2) \leq i \leq c_1 + c_2]\} \\ & \cup \{max(min(W_1), |\phi_2|+1), max(min(W_1), |\phi_2|+1)+1, \dots\} \\ =& \{i \mid \exists c_1 \in W_1\ \exists c_2 \in W_2\ [max(c_1, c_2) \leq i \leq c_1 + c_2]\} \\ & \cup \{max(min(W_1), |\phi_2|+2, \dots, |\phi|\} \cup \{|\phi|+1, |\phi|+2, \dots\} \end{aligned}$$

Note that $\{i \mid \exists c_1 \in W_1\ \exists c_2 \in W_2\ [max(c_1, c_2) \leq i \leq c_1 + c_2]\} \cup \{max(min(W_1), |\phi_2|+1), \dots, |\phi|\}$ is a subset of $\{1, 2, \dots, |\phi|\}$, as $c_1 + c_2 \leq |\phi_1| + |\phi_2| = |\phi|$.

— $\mathsf{op} = \otimes$:

$$\begin{aligned} C(\phi) &= \{c_1 + c_2 | c_1 \in W_1 \wedge (c_2 \in W_2 \vee c_2 \in [|\phi_2|, \infty))\} \\ &= \{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \cup \{min(W_1) + |\phi_2| + 1, min(W_1) + |\phi_2| + 2, \dots\} \\ &= \{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \cup \{min(W_1) + |\phi_2| + 1, \dots, |\phi|\} \cup \{|\phi|+1, |\phi|+2, \dots\} \end{aligned}$$

Note that $\{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \cup \{min(W_1) + |\phi_2|, \dots, |\phi|]\}$ is a subset of $\{1, 2, \dots, |\phi|\}$.

Case 3: Both $C(\phi_1)$ and $C(\phi_2)$ are infinite sets, where $C(\phi_1) = W_1 \cup \{i | i \in [|\phi_1|, \infty)\}$ and $C(\phi_2) = W_2 \cup \{i | i \in [|\phi_2|, \infty)\}$. The argument is similar to Case 2. We omit the details here.

Proof of Lemma 11

When $\phi = \mathsf{All}$ or $\phi = r$ or $\phi = \phi_1^+$, $C(\phi)$ can be computed in constant time according to Definition 14.

There are $|\phi| - 1$ binary operators in ϕ. Hence, to prove the lemma, we just need to prove that, given $C(\phi_1)$ and $C(\phi_2)$, $C(\phi)$ can be computed in time polynomial in the size of $|\phi|$, where $\phi = (\phi_1\ \mathsf{op}\ \phi_2)$.

According to Lemma 10, we may represent the characteristic set of a term ϕ as a tuple. Let $W \subseteq \{1, \dots, |\phi|\}$. When $C(\phi) = W$, we represent $C(\phi)$ as a tuple $\langle |\phi|, W, 0\rangle$; when $C(\phi) = W \cup \{|\phi|+1, |\phi|+2, \dots\}$, we represent $C(\phi)$ as a tuple $\langle |\phi|, W, 1\rangle$. In other words, the last element (either 0 or 1) of the tuple indicates whether $C(\phi)$ contains $\{|\phi|+1, |\phi|+2, \dots\}$ or not.

Given $C(\phi_1)$ and $C(\phi_2)$, we represent them as tuples $\langle |\phi_1|, W_1, f_1\rangle$ and $\langle |\phi_2|, W_2, f_2\rangle$, where $f_1, f_2 \in \{0, 1\}$. Now, we show that computing the tuple-representation $\langle |\phi|, W, f\rangle$ of $C(\phi)$ can be done in polynomial time. Note that $|\phi| = |\phi_1| + |\phi_2|$. We just need to determine W and f.

—Case $f_1 = f_2 = 0$: According to Definition 14, it is clear that $f = 0$. Computing W from W_1 and W_2, by following Definition 14, involves set union/intersection or computing the sums of pairs of elements, which can be done in $O(|\phi_1| \cdot |\phi_2|)$.

—Case $f_1 = 0$ and $f_2 = 1$: According to Definition 14, if op $= \sqcap$, then $f = 0$; otherwise, $f = 1$. Computing W from W_1 and $W_2 \cup \{|\phi_2|+1, \ldots, |\phi|\}$ can be done in $O(|\phi_1| \cdot |\phi|)$.

—Case $f_1 = 1$ and $f_2 = 0$: Similar to the above.

—Case $f_1 = 1$ and $f_2 = 1$: According to Definition 14, we have $f = 1$. Computing W from $W_1 \cup \{|\phi_1| + 1, \ldots, |\phi|\}$ and $W_2 \cup \{|\phi_2| + 1, \ldots, |\phi|\}$ can be done in $O(|\phi|^2)$.

In summary, computing $C(\phi)$ takes polynomial time.

Proof of Theorem 12

Given a term ϕ, let $C'(\phi)$ be the set of all integers k's such that there is a userset of size k that satisfies ϕ under some configuration. We would like to prove that $C'(\phi) \equiv C(\phi)$. We prove this by induction on the structure of ϕ.

Base case: when $\phi = \mathsf{All}$, ϕ is satisfied by any userset that is singleton; when $\phi = r$, ϕ is satisfied by a singleton containing a user who is a member of r. Hence, we have $C'(\mathsf{All}) = C'(r) = \{1\}$. According to Definition 14, $C'(\phi) \equiv C(\phi)$.

Inductive case: assume that $C'(\phi) \equiv C(\phi)$ when $|\phi| < k$, where $|\phi|$ is the number of atomic terms in ϕ. When $|\phi| = k$, we have:

—Case $\phi = \phi_1 \sqcup \phi_2$: It follows from the definition of satisfaction (Definition 3) that $C'(\phi_1 \sqcup \phi_2) = C'(\phi_1) \cup C'(\phi_2)$. By inductive assumption, $C'(\phi_1) \equiv C(\phi_1)$ and $C'(\phi_2) \equiv C(\phi_2)$. According to Definition 14, we have $C'(\phi) \equiv C(\phi)$.

—Case $\phi = \phi_1 \sqcap \phi_2$: It follows from Definition 3 that $C'(\phi_1 \sqcap \phi_2) \subseteq C'(\phi_1) \cap C'(\phi_2)$. In the following, we prove that $C'(\phi_1 \sqcap \phi_2) \supseteq C'(\phi_1) \cap C'(\phi_2)$, where ϕ_1 and ϕ_2 are free of negation and explicit sets of users.
Assume that X_1 is a size-k userset that satisfies ϕ_1 under configuration $\langle U, UR_1 \rangle$ and X_2 is a size-k userset that satisfies ϕ_2 under configuration $\langle U, UR_2 \rangle$. Since ϕ_1 and ϕ_2 do not contain explicit sets of users, the names of users are not important. Hence, we can assume that $X_1 = X_2$. Also, since ϕ_1 does not contain negation, X_1 still satisfies ϕ_1 even if we assign more roles to users in X_1. Therefore, X_1 satisfies ϕ_1 under $\langle U, UR_1 \cup UR_2 \rangle$. Also, X_1 (which is equivalent to X_2) satisfies ϕ_2 under $\langle U, UR_1 \cup UR_2 \rangle$. Therefore, X_1 satisfies $\phi_1 \sqcap \phi_2$. Since $|X_1| = k$, we have $k \in C'(\phi_1 \sqcap \phi_2)$. Hence, $C'(\phi_1 \sqcap \phi_2) \supseteq C'(\phi_1) \cap C'(\phi_2)$.
In summary, we have $C'(\phi_1 \sqcap \phi_2) = C'(\phi_1) \cap C'(\phi_2)$. By inductive assumption, $C'(\phi_1) \equiv C(\phi_1)$ and $C'(\phi_2) \equiv C(\phi_2)$. According to Definition 14, we have $C'(\phi) \equiv C(\phi)$.

—Case $\phi = \phi_0^+$: It follows from the computation of $C'(\mathsf{All}), C'(r), C'(\phi_1 \sqcup \phi_2)$ and $C'(\phi_1 \sqcap \phi_2)$ that $C'(\phi_0) = \{1\}$, where ϕ_0 is a unit term free of explicit sets of users and negation. Given a configuration $\langle U, UR \rangle$ and a singleton $\{u_1\}$ such that $\{u_1\}$ satisfies ϕ_0, we create $u_2, \ldots, u_n$ such that u_i ($i \in [2, n]$) is assigned to precisely the same set of roles as u_1. In this case, $\{u_1, \ldots, u_n\}$ satisfies ϕ_0^+. In other words, ϕ_0^+ may be satisfied by n users for any $n \geq 1$. That is to say, $C'(\phi_0^+) = \{i \mid i \in [1, \infty)\}$. According to Definition 14, we have $C'(\phi) \equiv C(\phi)$.

—Case $\phi = \phi_1 \odot \phi_2$: Let X be a userset that satisfies $(\phi_1 \odot \phi_2)$. There exist X_1 and X_2 such that X_1 satisfies ϕ_1, X_2 satisfies ϕ_2, and $X_1 \cup X_2 = X$. By the definition of C',

there exist $c_1 \in C'(\phi_1)$ and $c_2 \in C'(\phi_2)$ such that $|X_1| = c_1$ and $|X_2| = c_2$. Hence, $max(c_1, c_2) \leq |X| \leq c_1 + c_2$.
Given $c_1 \in C'(\phi_1)$ and $c_2 \in C'(\phi_2)$, there exist X_1 and X_2 such that X_1 satisfies ϕ_1 under $\langle U_1, UR_1\rangle$, X_2 satisfies ϕ_2 under $\langle U_2, UR_2\rangle$, $|X_1| = c_1$ and $|X_2| = c_2$. For any integer $k \in [max(c_1, c_2), c_1 + c_2]$, we may name users in such a way that $|X_1 \cap X_2| = c_1 + c_2 - k$. In this case, $X = X_1 \cup X_2$ satisfies $(\phi_1 \odot \phi_2)$ under $\langle U_1 \cup U_2, UR_1 \cup UR_2\rangle$ and $|X| = k$.
In summary, $C'(\phi_1 \odot \phi_2) = \{\ i \mid \exists\ c_1 \in C'(\phi_1) \quad \exists\ c_2 \in C'(\phi_2)\ [\ max(c_1, c_2) \leq i \leq c_1 + c_2\ ]\}$. By inductive assumption, $C'(\phi_1) \equiv C(\phi_1)$ and $C'(\phi_2) \equiv C(\phi_2)$. According to Definition 14, we have $C'(\phi) \equiv C(\phi)$.

—Case $\phi = \phi_1 \otimes \phi_2$: On the one hand, userset X satisfies $(\phi_1 \otimes \phi_2)$ if and only if there exist X_1 and X_2 such that $X_1 \cup X_2 = X$, $X_1 \cap X_2 = \emptyset$ and X_1, X_2 satisfy ϕ_1, ϕ_2 respectively. By definition of C', we have $|X_1| \in C'(\phi_1)$ and $|X_2| \in C'(\phi_2)$. Therefore, $|X| = (|X_1| + |X_2|) \in \{\ c_1 + c_2 \mid c_1 \in C'(\phi_1)\ \wedge\ c_2 \in C'(\phi_2)\ \}$.
On the other hand, given any $c_1 \in C'(\phi_1)$ and $c_2 \in C'(\phi_2)$, by definition of C', there exist X_1 and X_2 that satisfy ϕ_1 and ϕ_2 under $\langle U_1, UR_1\rangle$ and $\langle U_1, UR_1\rangle$ respectively, such that $|X_1| = c_1$ and $|X_2| = c_2$. Name the users in such a way that $X_1 \cap X_2 = \emptyset$. We have $X = X_1 \cup X_2$ satisfies $(\phi_1 \otimes \phi_2)$ under $\langle U_1 \cup U_2, UR_1 \cup UR_2\rangle$, where $|X| = |X_1| + |X_2| = c_1 + c_2$.
In summary, $C'(\phi_1 \otimes \phi_2) = \{\ c_1 + c_2 \mid c_1 \in C'(\phi_1)\ \wedge\ c_2 \in C'(\phi_2)\ \}$. By inductive assumption, $C'(\phi_1) \equiv C(\phi_1)$ and $C'(\phi_2) \equiv C(\phi_2)$. According to Definition 14, we have $C'(\phi) \equiv C(\phi)$.

In conclusion, we have $C'(\phi) \equiv C(\phi)$ and Theorem 12 holds.

C. PROOFS OF THEOREMS IN SECTION 5

In the following proofs, $(\mathsf{op}_k\phi)$ denotes k copies of ϕ connected together by operator op and $(\mathsf{op}_{i=1}^n r_i)$ denotes $(r_1\ \mathsf{op} \ldots \mathsf{op}\ r_n)$. Given $R = \{r_1, \ldots, r_m\}$, $(\mathsf{op}R)$ denotes $(r_1\ \mathsf{op} \ldots \mathsf{op}\ r_m)$.

C.1 The five intractability subcases of UTS

LEMMA 19. UTS $\langle \sqcup, \odot\rangle$ *is* **NP***-hard.*

PROOF. We use a reduction from the **NP**-complete SET COVERING problem [Garey and Johnson 1979]. In the SET COVERING problem, we are given a finite set $S = \{e_1, \ldots, e_n\}$, a family of S's subsets $F = \{S_1, \ldots, S_m\}$, and an integer $k < m$, and we ask whether there exists a sub-family of sets $F' \subseteq F$ whose union is S and $|F'| \leq k$. Given such an instance, our reduction maps each element in S to a user and to a role. We construct a configuration $\langle U, UR\rangle$ such that $U = \{u_1, \ldots, u_n\}$ and $UR = \{(u_i, r_i) \mid i \in [1, n]\}$, and a term $\phi = (\bigodot_k(\bigsqcup_{i=1}^m(\bigodot R_i)))$, where R_i is a set of roles such that $r_j \in R_i$ if and only if $e_j \in S_i$.

We now demonstrate that U satisfies ϕ under $\langle U, UR\rangle$ if and only if there exist k sets in F whose union is S. On the one hand, assume that U satisfies ϕ, by definition. U has k subsets $U_1, \ldots, U_k$ such that $\bigcup_{i=1}^k U_i = U$ and every U_i satisfies $(\bigsqcup_{i=1}^m(\bigodot R_i))$. U_i satisfies $(\bigsqcup_{i=1}^m(\bigodot R_i))$ if and only if U_i satisfies a certain $(\bigodot R_{x_i})$, where $x_i \in [1, m]$. From the construction of R_{x_i}, U_i satisfies $(\bigodot R_{x_i})$ if and only if $U_i = \{u_a \mid e_a \in S_{x_i}\}$. Since $\bigcup_{i=1}^k U_i = U$, we have $\bigcup_{i=1}^k S_{x_i} = S$. The answer to the set covering problem is "yes".

On the other hand, assume that there are k sets in F whose union is S. Without loss of generality, we assume that $\bigcup_{i=1}^{k} S_i = S$. In this case, we divide U into k sets $U_1, \ldots, U_k$ such that $U_i = \{u_j \mid e_j \in S_i\}$. Since $\bigcup_{i=1}^{k} S_i = S$, we have $\bigcup_{i=1}^{k} U_i = U$. Furthermore, since $U_i = \{u_j \mid e_j \in S_i\}$, from the construction of R_i, we have U_i satisfies $(\bigodot R_i)$ for every $i \in [1, k]$. Therefore, U satisfies $\phi = (\bigodot_k (\bigsqcup_{i=1}^{m} (\bigodot R_i)))$. □

LEMMA 20. UTS $\langle \sqcap, \odot \rangle$ *is* **NP**-*hard.*

PROOF. We use a reduction from the **NP**-complete SET COVERING problem [Garey and Johnson 1979]. Given $S = \{e_1, \ldots, e_n\}$, a family of S's subsets $F = \{S_1, \ldots, S_m\}$, and an integer $k < m$, our reduction maps each element $e_j \in S$ to a role r_j and each $S_i \in F$ to a user u_i. We construct a configuration $\langle U, UR \rangle$ such that $U = \{u_1, \ldots, u_m\}$ and $UR = \{(u_i, r_j) \mid e_j \in S_i\}$, and a term $\phi = (((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i)) \odot (\bigodot_m \mathsf{All}))$.

We now demonstrate that U satisfies ϕ under $\langle U, UR \rangle$ if and only if there exist k sets in family F whose union is S. On the one hand, assume that U satisfies ϕ. Since $(\bigodot_m \mathsf{All})$ can be satisfied by any nonempty userset with no more than m users, U always satisfies $(\bigodot_m \mathsf{All})$ and it satisfies ϕ if and only if there is $U' \subseteq U$ such that U' satisfies $((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i))$. U' satisfying $(\bigodot_k \mathsf{All})$ indicates that $|U'| \leq k$, while U' satisfying $(\bigodot_{i=1}^{n} r_i)$ indicates that users in U' together have membership of all roles in $\{r_1, \ldots, r_n\}$. Without loss of generality, suppose $U' = \{u_1, \ldots, u_t\}$, where $t \leq k$. Because $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_1, \ldots, S_t\}$ is S. The answer to the SET COVERING problem is "yes".

On the other hand, assume that k subsets in F cover S. Without loss of generality, we assume that $\bigcup_{i=1}^{k} S_i = S$. From the construction of UR, users $u_1, \ldots, u_k$ together have membership of all roles in $\{r_1, \ldots, r_n\}$. In this case, $\{u_1, \ldots, u_k\}$ satisfies $(\bigodot_{i=1}^{n} r_i)$. Also, $\{u_1, \ldots, u_k\}$ satisfies $(\bigodot_k \mathsf{All})$. Hence, $\{u_1, \ldots, u_k\}$ satisfies $((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i))$. $(\bigodot_m \mathsf{All})$ is also satisfied by U. Therefore, U satisfies ϕ. □

LEMMA 21. UTS $\langle \odot, \otimes \rangle$ *is* **NP**-*hard.*

PROOF. We use a reduction from the **NP**-complete DOMATIC NUMBER problem [Garey and Johnson 1979]. Given a graph $G(V, E)$, the Domatic Number problem asks whether V can be partitioned into k disjoint nonempty sets $V_1, V_2, \ldots, V_k$, such that each V_i is a dominating set for G. V' is a dominating set for $G = (V, E)$ if for every node u in $V - V'$, there is a node v in V' such that $(u, v) \in E$.

Given a graph $G = (V, E)$ and a threshold k, let $U = \{u_1, u_2, \ldots, u_n\}$ and $R = \{r_1, r_2, \ldots, r_n\}$, where n is the number of nodes in V. Each user in U corresponds to a node in G, and $v(u_i)$ denotes the node corresponding to user u_i. $UR = \{(u_i, r_j) \mid i = j \text{ or } (v(u_i), v(u_j)) \in E\}$. Let $\phi = (\bigotimes_k (\bigodot_{i=1}^{n} r_i))$.

A dominating set in G corresponds to a set of users that together have membership of all the n roles. U satisfies ϕ under $\langle U, UR \rangle$ if and only if U can be divided into k pairwise disjoint sets, each of which has role membership of $r_1, r_2, \ldots, r_n$. Therefore, the answer to the Domatic Number problem is "yes" if and only if U satisfies ϕ under $\langle U, UR \rangle$. □

LEMMA 22. UTS $\langle \otimes, \sqcup \rangle$ *is* **NP**-*hard.*

PROOF. We use a reduction from the **NP**-complete SET PACKING problem [Garey and Johnson 1979], which asks: Given a finite set $S = \{e_1, \ldots, e_n\}$, a family of S's subsets $F = \{S_1, \ldots, S_m\}$, and an integer k, whether there are k pairwise disjoint elements (which are sets) in F? Without loss of generality, we assume that $S_i \not\subseteq S_j$ when $i \neq j$. (If

$S_i \subseteq S_j$, one can remove S_j without affecting the answer.) Let $U = \{u_0, u_1, \ldots, u_n\}$, $R = \{r_1, \ldots, r_n\}$ and $UR = \{(u_i, r_i) \mid 1 \leq i \leq n\}$. Note that u_0 is a user that is not assigned to any role. We then construct a term $\phi = ((\bigotimes_k (\bigsqcup_{i=1}^m (\bigotimes R_j))) \otimes \phi_{nonempty})$, where $R_j = \{r_i \mid e_i \in S_j\}$ and $\phi_{nonempty} = (\mathsf{All} \sqcup (\mathsf{All} \otimes \mathsf{All}) \sqcup \cdots \sqcup (\bigotimes_m \mathsf{All}))$.

We show that U satisfies ϕ under $\langle U, UR\rangle$ if and only if there are k pairwise disjoint elements in family F. As the only member of r_i is u_i, the only userset that satisfies $\phi_i = (\bigotimes R_j)$ is $U_j = \{u_i \mid e_i \in S_j\}$. Hence, a userset X satisfies $\phi' = (\bigsqcup_{i=1}^m \phi_i)$ if and only if X equals to some U_j.

Without loss of generality, assume that $S_1, \ldots, S_k$ are k pairwise disjoint sets. Then, $U_1, \ldots, U_k$ are k pairwise disjoint sets of users. U_1 satisfies ϕ_1, and thus satisfies ϕ'. Similarly, we have U_i satisfies ϕ' for every i from 1 to k. Furthermore, since $u_0 \notin U_i$ for any $i \in [1, k]$, we have $\bigcup_{i=1}^k U_i \subset U$. Hence, U can be divided into two nonempty subset $\bigcup_{i=1}^k U_i$ and $U' = U - \bigcup_{i=1}^k U_i$ such that $\bigcup_{i=1}^k U_i$ satisfies $(\bigotimes_k (\bigsqcup_{i=1}^m (\bigotimes R_j)))$ and U' satisfies $\phi_{nonempty}$. In other words, U satisfies ϕ.

On the other hand, suppose that U satisfies ϕ. Then, U has a strict subset U' with $u_0 \notin U'$, such that U' can be divided into k pairwise disjoint sets $\hat{U_1}, \ldots, \hat{U_k}$, such that each $\hat{U_i}$ satisfies ϕ'. In order to satisfy ϕ', $\hat{U_i}$ must satisfy a certain ϕ_{a_i} and hence be equivalent to U_{a_i}, where $a_i \in [1, m]$. The assumption that $\hat{U_1}, \ldots, \hat{U_k}$ are pairwise disjoint indicates that $U_{a_1}, \ldots, U_{a_k}$ are also pairwise disjoint. Therefore, their corresponding sets $S_{a_1}, \ldots, S_{a_k}$ are pairwise disjoint. The answer to the SET PACKING problem is "yes". □

LEMMA 23. UTS $\langle \sqcap, \otimes \rangle$ *is* **NP***-hard.*

PROOF. We use a reduction from the NP-complete SET COVERING problem, which asks: Given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an integer k no larger than m, whether there is a subfamily of sets $F' \leq F$ whose union is S and $|F'| \leq k$?

Given S and F, let $U = \{u_1, u_2, \ldots, u_m\}$, $R = \{r_1, r_2, \ldots, r_n\}$ and $UR = \{(u_i, r_j) \mid e_j \in S_i\}$. Let $\phi = ((\bigsqcap_{i=1}^n (r_i \otimes (\bigotimes_{k-1} \mathsf{All}))) \otimes (\bigotimes_{m-k} \mathsf{All}))$. We now demonstrate that U satisfies ϕ under $\langle U, UR\rangle$ if and only if there are k sets in family F whose union is S. Without loss of generality, assume that $k < m$.

First, assume that U satisfies ϕ. Since $(\bigotimes_{m-k} \mathsf{All})$ can be satisfied by any userset with $m - k$ users, U satisfies ϕ if and only if there is a size-k subset U' of U that satisfies $(r_i \otimes (\bigotimes_{k-1} \mathsf{All}))$ for every $i \in [1, n]$. This means that users in U' together have membership of all roles in $\{r_1, \ldots, r_n\}$. Suppose $U' = \{u_{a_1}, \ldots, u_{a_k}\}$, where $a_i \in [1, m]$. Because $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_{a_1}, \ldots, S_{a_k}\}$ is S. The answer to the Set Covering problem is "yes".

Second, without loss of generality, assume that $\bigcup_{i=1}^k S_i = S$. From the construction of UR, users $u_1, \ldots, u_k$ together have membership of $r_1, \ldots, r_n$. In this case, $\{u_1, \ldots, u_k\}$ satisfies $(r_i \otimes (\bigotimes_{k-1} \mathsf{All}))$ for every $i \in [1, n]$. Since $k < m$, $\{u_1, \ldots, u_k\}$ is a strict subset of U. Therefore, U can be divided into two nonempty subset $\{u_1, \ldots, u_k\}$ and $U - \{u_1, \ldots, u_k\}$ such that $\{u_1, \ldots, u_k\}$ satisfies $(\bigsqcap_{i=1}^n (r_i \otimes (\bigotimes_{k-1} \mathsf{All})))$ and $U - \{u_1, \ldots, u_k\}$ satisfies $(\bigotimes_{m-k} \mathsf{All})$. In other words, U satisfies ϕ. □

C.2 Proof that UTS is in NP

LEMMA 24. UTS $\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ *is in* **NP**.

PROOF. Given a term ϕ, a configuration $\langle U, UR\rangle$ and a userset X, according to Theo-

rem 2, X satisfies ϕ if and only if there exists a satisfaction tree of ϕ whose root is labeled with X. A non-deterministic Turing machine may guess a satisfaction tree T of ϕ such that the root of T is labeled with X. From the proof of Theorem 7, the size of T is polynomial in the size of ϕ and verifying whether T is a satisfaction tree can be done in polynomial time by following the rules in Definition 4. Therefore, UTS $\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP}$. □

C.3 The tractable cases

LEMMA 25. UTS *for 4CF terms is in* $\mathbf{P}$.

PROOF. Given a 4CF term $\phi = (P_1 \odot \cdots \odot P_n)$, where for each k such that $1 \leq k \leq n$, P_k is a 3CF term of the form $(\phi_{k,1} \otimes \phi_{k,2} \otimes \cdots \otimes \phi_{k,m_k})$, and each $\phi_{k,j}$ is a 1CF term. Let $t_{k,j}$ be the base (which is a unit term) of $\phi_{k,j}$. $T_k = \{t_{k,1}, t_{k,2}, \ldots, t_{k,m_k}\}$ is a multiset of the base of the 1CF terms in P_k.

Given a userset $X = \{u_1, \ldots, u_n\}$ and configuration $\langle U, UR \rangle$, we present an algorithm that determines whether X satisfies ϕ under $\langle U, UR \rangle$.

Step 1 The first step checks that each P_k is satisfied by some subset of X. For each k such that $1 \leq k \leq n$, do the following. Construct a bipartite graph $G(X, T_k)$, in which one partition consists of users in X and the other consists of all the $t_{k,j}$'s in T_k; and there is an edge between $u \in X$ and $t_{k,j}$ if and only if $\{u\}$ satisfies $t_{k,j}$. Compute a maximal matching of the graph $G(X, T_k)$, if the size of the matching is less than m_k, returns "no", as this means that X does not contain a subset that satisfies P_k; thus X does not satisfy ϕ.

Step 2 The second step checks that each user in X can be "consumed" by some unit term in ϕ. Let $G(A, B)$ denote the bipartite graph in which one partition, A, consists of users in X, and the other partition, B, consists of all the $t_{k,j}$'s in $T_1 \cup T_2 \cup \cdots \cup T_n$. Furthermore, for any unit term t that occurs as t^+ in ϕ, we make sure that B has at least $|X|$ copies of t by adding additional copies of t if necessary. There is an edge between $u \in A$ and $t \in B$ if and only if $\{u\}$ satisfies t. Compute a maximal matching of the graph $G(X, T)$, if the matching has size less than $|X|$, returns "no".

Step 3 Return "yes".

It is not difficult to see that if the algorithm returns "no", then X does not satisfy ϕ. We now show that if the algorithm returns "yes", then X satisfies ϕ. If the algorithm returns "yes", then for each k, the graph $G(X, T_k)$ has a matching of size m_k. Let X_k be the set of users involved in the matching. X_k satisfies P_k. Let $X' = X_1 \cup X_2 \cup \cdots \cup X_n$. If $X' = X$, then clearly X satisfies ϕ. If $X' \subset X$, then find a user u in $X \setminus X'$, and do the following: Find the term t that is matched with u in the maximal matching computed in step 2. Such a term must exist, since the matching has size $|X|$. Without loss of generality, assume that t appears in P_1, and X_1 contains a user w that is matched with t; then change X_1 by replacing w with u. Clearly, the new X_1 still satisfies P_1. Compute X' again, and if $X' \subset X$, find another user in $X \setminus X'$ and repeat the previous process. Note that X' will grow if w appears in some other X_k. Also observe that, the newly added matching between u and t will never be removed again in future, because no other user is matched with t in the maximal matching computed in step 2; as a result, u will always remain in X'. Therefore, after each step, one new user will be added to X' and will never be removed. After at most $|X|$ steps, we will have $X' = X$. □

D. PROOF OF THEOREM 17

Proofs of the P results in Theorem 17

We first prove the following lemma, which will be useful.

LEMMA 26. The following properties hold.

(1) A userset X satisfies a unit term t if and only if X is a singleton and the only user in X satisfies t.
(2) A userset X satisfies a term t^+, where t is a unit term, if and only if every user in X satisfies t.
(3) If a userset X satisfies a term ϕ that is built using only $\neg, +, \sqcap, \sqcup$, then every user in X satisfies ϕ.
(4) A userset X is safe with respect to a 1CF term ϕ if and only if there exists a user in X that satisfies t.

PROOF. Properties 1 and 2 follow from the definition of term satisfaction. Observe that a unit term can be satisfied only by a singleton.

Property 3. The term ϕ can be decomposed into subterms in 1CF form, connected using $\sqcap$ and $\sqcup$. By definition, X satisfies $\phi_1 \sqcap \phi_2$ if and only if X satisfies both ϕ_1 and ϕ_2, and X satisfies $\phi_1 \sqcup \phi_2$ if and only if X satisfies either ϕ_1 or ϕ_2. Identify all 1CF subterms that X satisfies, it follows from Properties 1 and 2 that each user in X satisfies all these subterms. Therefore, each user satisfies ϕ.

Property 4. For the "if" direction, if X contains a user u that satisfies t, then $\{u\}$ satisfies the term ϕ, and thus X is safe with respect to ϕ. For the "only if" direction, if X is safe with respect to ϕ, then X contains a subset X_0 that satisfies ϕ. Any user in X_0 must satisfy t according to Properties 1 and 2. □

LEMMA 27. SAFE $\langle \neg, +, \sqcup, \odot \rangle$ is in $\mathbf{P}$.

PROOF. A userset X is safe with respect to $(\phi_1 \sqcup \phi_2)$ if and only if either X is safe with respect to ϕ_1 or X is safe with respect to ϕ_2. Furthermore, X is safe with respect to $(\phi_1 \odot \phi_2)$ if and only if X is safe with respect to both ϕ_1 and ϕ_2. Therefore, one can determine whether U is safe with respect to ϕ, which is built using only the operators in $\{\neg, +, \sqcup, \odot\}$, by following the structure of the term until reaching subterms in 1CF. From Property 4 of Lemma 26, checking whether U is safe with respect to such a term amounts to checking whether there exists a user in U that satisfies t, which can be done in polynomial time. □

LEMMA 28. SAFE $\langle \neg, +, \sqcup, \sqcap \rangle$ is in $\mathbf{P}$.

PROOF. Given a term ϕ which is built using only operators in $\{\neg, +, \sqcup, \sqcap\}$, we prove that a userset X is safe with respect to ϕ if and only if there exists a user $u \in X$ such that u satisfies ϕ. The "if" direction follows by definition. For the "only if" direction: Suppose that X contains a nonempty subset X_0 that satisfies ϕ, then by Property 3 of Lemma 26, every user in X_0 satisfies ϕ; thus X must contain a user that satisfies ϕ. Therefore, to determine whether X is safe with respect to ϕ, one can, for each user in X, check whether the user satisfies ϕ. Checking whether one user satisfies a term using only operators in $\{\neg, +, \sqcup, \sqcap\}$ can be done in polynomial time. □

LEMMA 29. SAFE $\langle \neg, +, \otimes \rangle$ is in $\mathbf{P}$.

PROOF. Given a term ϕ which does not contain any binary operator but $\otimes$, we show that determining whether a userset X is safe with respect to ϕ under a configuration $\langle U, UR\rangle$ can be reduced to the maximum matching problem on bipartite graphs, which can be solved in $O(MN)$ time, where M is the number of edges and N is the number of nodes in G [Papadimittiou and Steiglitz 1982].

Let s be the number of 1CF terms in ϕ and $t = |X|$. Since $\otimes$ is associative, ϕ can be equivalently expressed as $(\phi_1 \otimes \phi_2 \otimes \cdots \otimes \phi_s)$, where each ϕ_i is a 1CF term . Let $X = \{u_1, \ldots, u_t\}$. We construct a bipartite graph $G(V_1 \cup V_2, E)$, where each node in V_1 corresponds to a 1CF term in ϕ and each node in V_2 corresponds to a user in X. More precisely, $V_1 = \{a_1, \ldots, a_s\}$, $V_2 = \{b_1, \ldots, b_t\}$, and $(a_i, b_j) \in E$ if and only if $\{u_j\}$ satisfies ϕ_i. The resulting graph G has $s+t$ nodes and $O(st)$ edges, and can be constructed in time polynomial in the size of G. Solving the maximal matching problem for G takes time $O((s+t)st)$.

We now show that X is safe with respect to ϕ if and only if the maximal matching in the graph G has size s. If the maximal matching has size s, then each node in V_1 matches to a certain node in V_2, which means that the s 1CF terms in ϕ are satisfied by s distinct users in X; thus X contains a subset that satisfies ϕ. If X is safe with respect to ϕ, by definition, there exist s disjoint subsets $X_1, \ldots, X_s$ such that X_i $(i \in [1, s])$ satisfies ϕ_i and $\bigcup_{j=1}^{s} X_j \subseteq X$. From our construction of G, we may match a node corresponding to a user in X_i to the node corresponding to ϕ_i. In this case, a maximal matching of size s exists. □

Proving the NP-completeness results in Table II

LEMMA 30. $\mathsf{SAFE}\langle\sqcap, \odot\rangle$ is **NP**-hard.

PROOF. We use a reduction from the **NP**-complete SET COVERING problem [Garey and Johnson 1979]. In the SET COVERING problem, we are given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an integer k no larger than m, and we ask whether there is a subfamily of sets $F' \subseteq F$ whose union is S and $|F'| \leq k$.

Given S and F, we construct a configuration $\langle U, UR\rangle$ such that $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$. Let $U = \{u_1, \ldots, u_m\}$ and $\phi = ((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i))$.

We now demonstrate that U is safe with respect to ϕ under $\langle U, UR\rangle$ if and only if there are no more than k sets in family F whose union is S.

First, if U is safe with respect to ϕ, by definition, a subset U' of U satisfies both $(\bigodot_k \mathsf{All})$ and $(\bigodot_{i=1}^{n} r_i)$. U' satisfying $(\bigodot_k \mathsf{All})$ indicates that $|U'| \leq k$, while U' satisfying $(\bigodot_{i=1}^{n} r_i)$ indicates that users in U' together have membership of r_i for every $i \in [1, n]$. Without loss of generality, suppose $U' = \{u_1, \ldots, u_t\}$, where $t \leq k$. Since $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_1, \ldots, S_t\}$ is S. The answer to the SET COVERING problem is "yes".

Second, without loss of generality, assume that $\bigcup_{i=1}^{k} S_i = S$. From the construction of UR, users $u_1, \ldots, u_k$ together have membership of r_i for every $i \in [1, n]$, which indicates that $\{u_1, \ldots, u_k\}$ is safe with respect to $(\bigodot_{i=1}^{n} r_i)$. Also, any non-empty subset of $\{u_1, \ldots, u_k\}$ satisfies $(\bigodot_k \mathsf{All})$. Hence, U is safe with respect to ϕ. □

LEMMA 31. $\mathsf{SAFE}\langle\odot, \otimes\rangle$ is **NP**-hard.

PROOF. We use a reduction from the **NP**-complete DOMATIC NUMBER problem [Garey and Johnson 1979]. Given a graph $G(V, E)$, the Domatic Number problem

asks whether V can be partitioned into k disjoint sets $V_1, V_2, \ldots, V_k$, such that each V_i is a dominating set for G. V' is a dominating set for $G = (V, E)$ if for every node u in $V - V'$, there is a node v in V' such that $(u, v) \in E$.

Given a graph $G = (V, E)$ and an integer k, let $U = \{u_1, u_2, \ldots, u_n\}$ and $R = \{r_1, r_2, \ldots, r_n\}$, where n is the number of nodes in V. Each user in U corresponds to a node in G, and $v(u_i)$ denotes the node corresponding to user u_i. Let $UR = \{(u_i, r_j) \mid i = j \text{ or } (v(u_i), v(u_j)) \in E\}$ and $\phi = (\bigotimes_k (\bigodot_{i=1}^{n} r_i))$.

A dominating set in G corresponds to a set of users who together have membership of all the n roles. U is safe with respect to ϕ if and only if U has a subset U' that can be divided into k pairwise disjoint sets, each of which have role membership of $r_1, r_2, \ldots, r_n$. Therefore, the answer to the Domatic Number problem is "yes" if and only if U is safe with respect to ϕ. □

LEMMA 32. SAFE$\langle \otimes, \sqcup \rangle$ is **NP**-hard.

PROOF. We use a reduction from the **NP**-complete SET PACKING problem [Garey and Johnson 1979], which asks, given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an integer k, whether there are k pairwise disjoint sets in family F. Without loss of generality, we assume that $S_i \not\subseteq S_j$ if $i \neq j$.

Given S and F, let $U = \{u_1, \ldots, u_n\}$, $R = \{r_1, \ldots, r_n\}$ and $UR = \{(u_i, r_i) \mid 1 \leq i \leq n\}$. We then construct a term $\phi = (\bigotimes_k (\bigsqcup_{i=1}^{m} (\bigotimes R_j)))$, where $R_j = \{r_i \mid e_i \in S_j\}$. We show that U is safe with respect to ϕ under $\langle U, UR \rangle$ if and only if there are k pairwise disjoint sets in family F.

As the only member of r_i is u_i, the only userset that satisfies $\phi_i = (\bigotimes R_j)$ is $U_j = \{u_i \mid e_i \in S_j\}$. A userset X satisfies $\phi' = (\bigsqcup_{i=1}^{m} \phi_i)$ if and only if X equals to some U_j.

First, without loss of generality, assume that $S_1, \ldots, S_k$ are k pairwise disjoint sets. Then, $U_1, \ldots, U_k$ are k pairwise disjoint sets of users. U_1 satisfies ϕ_1, and thus satisfies ϕ'. Similarly, U_i satisfies ϕ' for every i from 1 to k. Since $U_i \subseteq U$, U is safe with respect to ϕ.

Second, suppose U is safe with respect to ϕ. Then, U has a subset U' that can be divided into k pairwise disjoint sets $\hat{U}_1, \ldots, \hat{U}_k$, such that $\hat{U}_i$ satisfies ϕ_i. In order to satisfy ϕ', $\hat{U}_i$ must satisfy a certain ϕ_{a_i} and hence be equivalent to U_{a_i}. The assumption that $\hat{U}_1, \ldots, \hat{U}_k$ are pairwise disjoint indicates that $U_{a_1}, \ldots, U_{a_k}$ are also pairwise disjoint. Therefore, their corresponding sets $S_{a_1}, \ldots, S_{a_k}$ are pairwise disjoint. The answer to the Set Packing problem is "yes". □

LEMMA 33. SAFE $\langle \sqcap, \otimes \rangle$ *is* **NP**-*hard.*

PROOF. We use a reduction from the NP-complete SET COVERING problem, which asks, given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an integer k no larger than m, whether there is a subfamily of sets $F' \subseteq F$ whose union is S and $|F'| \leq k$.

Given S and F, let $U = \{u_1, u_2, \ldots, u_m\}$, $R = \{r_1, r_2, \ldots, r_n\}$ and $UR = \{(u_i, r_j) \mid e_j \in S_i\}$. Let $\phi = (\bigsqcap_{i=1}^{n} (r_i \otimes (\bigotimes_{k-1} \mathsf{All})))$. We now demonstrate that U satisfies ϕ under $\langle U, UR \rangle$ if and only if there are k sets in family F whose union is S.

If U is safe with respect to ϕ, by definition, a subset U' of U satisfies $(r_i \otimes (\bigotimes_{k-1} \mathsf{All}))$ for every $i \in [1, n]$, which indicates users in U' together have membership of r_i for every $i \in [1, n]$. For any $i \in [1, n]$, U' satisfying $(r_i \otimes (\bigotimes_{k-1} \mathsf{All}))$ indicates that $|U'| = k$.

Suppose $U' = \{u_{a_1}, \ldots, u_{a_k}\}$. Because $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_{a_1}, \ldots, S_{a_k}\}$ is S. The answer to the SET COVERING problem is "yes".

On the other hand, without loss of generality, assume that $\bigcup_{i=1}^{k} S_i = S$. From the construction of UR, users $u_1, \ldots, u_k$ together have membership of r_i for every $i \in [1, n]$, which indicates that $\{u_1, \ldots, u_k\}$ satisfies ϕ_i for every $i \in [1, n]$. Hence, $\{u_1, \ldots, u_k\}$ satisfies ϕ and U is safe with respect to ϕ. □

E. PROOF OF THEOREM 18

LEMMA 34. $\mathsf{SSC}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{coNP^{NP}}$.

PROOF. We show that the complement of $\mathsf{SSC}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{NP^{NP}}$. Because SAFE is in $\mathbf{NP}$ (see Table II), an $\mathbf{NP}$ oracle can decide whether a userset is safe with respect to a term. We construct a nondeterministic Oracle Turing Machine M that accepts an input consisting of a state $\langle U, UR, UP\rangle$ and a policy $\mathsf{sp}\langle P, \phi\rangle$ if and only if $\langle U, UR, UP\rangle$ is not safe with respect to $\mathsf{sp}\langle P, \phi\rangle$. M nondeterministically selects a set U of users in $\langle U, UR, UP\rangle$. If U does not cover P, then M rejects. Otherwise, M invokes the $\mathbf{NP}$ oracle to check whether U is safe with respect to ϕ. If the oracle answers "yes", then M rejects; otherwise, M accepts, as it has found a userset that covers P but is not safe with respect to ϕ, which violates the static safety policy. The construction of M shows that the complement of $\mathsf{SSC}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{NP^{NP}}$. Hence, $\mathsf{SSC}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{coNP^{NP}}$. □

LEMMA 35. $\mathsf{SSC}\langle\sqcup, \odot\rangle$ *is* $\mathbf{coNP}$*-hard.*

PROOF. We reduce the $\mathbf{coNP}$-complete VALIDITY problem for propositional logic to $\mathsf{SSC}\langle\sqcup, \odot\rangle$. Given a propositional logic formula φ in disjunctive normal form, let $\{v_1, \ldots, v_n\}$ be the set of propositional variables in φ.

We create a state $\langle U, UR, UP\rangle$ with n permissions $p_1, p_2, \ldots, p_n$, $2n$ users $u_1, u_1', u_2, u_2', \ldots, u_n, u_n'$, and $2n$ roles $r_1, r_1', r_2, r_2', \ldots, r_n, r_n'$. We have $UP = \{(u_i, p_i), (u_i', p_i) \mid 1 \leq i \leq n\}$ and $UR = \{(u_i, r_i), (u_i', r_i') \mid 1 \leq i \leq n\}$. We also construct a term ϕ from the formula φ by replacing each literal v_i with r_i, each literal $\neg v_i$ with r_i', each occurrence of $\wedge$ with $\odot$ and each occurrence of $\vee$ with $\sqcup$.

Note that X is safe with respect to $\phi_1 \sqcup \phi_2$ if and only if X is safe respect to either ϕ_1 or ϕ_2, and X is safe with respect to $\phi_1 \odot \phi_2$ if and only if X is safe respect to both ϕ_1 and ϕ_2. Thus the logical structure of ϕ follows that of φ.

We now show that the formula φ is valid if and only if $\langle U, UR, UP\rangle$ is safe with respect to the policy $\mathsf{sp}\langle\{p_1, p_2, \ldots, p_n\}, \phi\rangle$. On the one hand, if the formula φ is not valid, then there is an assignment I that makes it false. Using that assignment, we construct a userset $X = \{u_i \mid I(v_i) = \text{true}\} \cup \{u_i' \mid I(v_i) = \text{false}\}$. X covers all permissions in P, but X is not safe with respect to ϕ. On the other hand, if $\langle U, UR, UP\rangle$ is not safe with respect to $\mathsf{sp}\langle\{p_1, p_2, \ldots, p_n\}, \phi\rangle$, then there exists a set X of users that covers P but X is not safe with respect to ϕ. In order to cover all permissions in P, for each $i \in [1, n]$, at least one of u_i, u_i' is in X. Without loss of generality, assume that for each i, exactly one of u_i, u_i' is in X. (If both u_i, u_i' are in X, we can remove either one, the resulting set is a subset of X and still covers P.) Then we can derive a truth assignment I from X by setting v_i to true if $u_i \in X$ and to false if $u_i' \in X$. Then the formula evaluates to false, because X is not safe with respect to ϕ. □

LEMMA 36. $\mathsf{SSC}\langle\sqcap, \odot\rangle$ is $\mathbf{NP}$-hard.

PROOF. There is a straightforward reduction from $\mathsf{SAFE}\langle\sqcap,\odot\rangle$ to $\mathsf{SSC}\langle\sqcap,\odot\rangle$. Given a term ϕ using only operators $\sqcap$ or $\odot$, in order to check whether a userset X is safe with respect to ϕ, we can construct a policy $\mathsf{sp}\langle P,\phi\rangle$ and a state $\langle U, UR, UP\rangle$ such that X is the only set of users in the state that covers P. In this case, X is safe with respect to ϕ if and only if the state we constructed satisfies $\mathsf{sp}\langle P,\phi\rangle$. Since $\mathsf{SAFE}\langle\sqcap,\odot\rangle$ is **NP**-hard (see Table II), $\mathsf{SSC}\langle\sqcap,\odot\rangle$ is **NP**-hard. □

LEMMA 37. $\mathsf{SSC}\langle\otimes\rangle$ *is* **coNP***-hard.*

PROOF. We can reduce the **NP**-complete SET COVERING problem to the complement of $\mathsf{SSC}\langle\otimes\rangle$. In SET COVERING, we are given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an integer k, where k is an integer smaller than m and n. We are asking whether there is a subfamily of sets $F' \subseteq F$ whose union is S and $|F'| \leq k$.

Given an instance of the Set Covering problem, construct a state $\langle U, UR, UP\rangle$ such that $UR = \{(u_i, r_i) \mid i \in [1, m]\}$ and $UP = \{(u_i, p_j) \mid e_j \in S_i\}$. Construct a safety policy $\mathsf{sp}\langle P,\phi\rangle$, where $P = \{p_1, \ldots, p_n\}$ and $\phi = (\bigotimes_{k+1} \mathsf{All})$. ϕ is satisfied by any set of no less than $k+1$ users.

First, if $\langle U, UR, UP\rangle$ is safe, no k users together have all permissions in P. In this case, since u_i corresponds to S_i, there does not exist k sets in family F whose union is S. The answer to the Set Covering problem is "no".

Second, if $\langle U, UR, UP\rangle$ is not safe, there exist a set of no more than k users together have all permissions in P. Accordingly, the answer to the Set Covering problem is "yes".

Since the SET COVERING problem is **NP**-complete, we conclude that the complement of $\mathsf{SSC}\langle\otimes\rangle$ is **NP**-hard. Hence, $\mathsf{SSC}\langle\otimes\rangle$ is **coNP**-hard. □

Tractable cases of SSC:

LEMMA 38. $\mathsf{SSC}\langle\neg, +, \sqcap, \sqcup\rangle$ is in **P**.

PROOF. Given a term ϕ with operators $\neg, +, \sqcap$ and $\sqcup$, construct another term ϕ' by removing $+$ in ϕ. For example, if $\phi = ((r_1 \sqcap r_2)^+ \sqcup r_3^+)$, then $\phi' = ((r_1 \sqcap r_2) \sqcup r_3)$. When only operators $\neg, +, \sqcap$ and $\sqcup$ are allowed, if a set U of users satisfies ϕ, then there exists $U' \subseteq U$ such that U' satisfies ϕ'. This indicates that U is safe with respect to ϕ if and only if U is safe with respect to ϕ'. Therefore, in order to show that $\mathsf{SSC}\langle\neg, +, \sqcap, \sqcup\rangle$ is tractable, it suffices to prove that $\mathsf{SSC}\langle\neg, \sqcap, \sqcup\rangle$ is in **P**.

A term ϕ' with operators $\neg, \sqcap$ and $\sqcup$ may be satisfied only by singleton. A state $\langle U, UR, UP\rangle$ is safe with respect to $\mathsf{sp}\langle\{p_1, \ldots, p_m\}, \phi'\rangle$, if and only if for any set U of users who together have all permissions in $\{p_1, \ldots, p_m\}$, there exists a user $u \in U$ such that $\{u\}$ satisfies ϕ'. This is equivalent to checking whether there exists a permission p_i ($i \in [1, m]$) such that for every user u having p_i, $\{u\}$ satisfies ϕ'. The following algorithm performs such a check.

```
isSafe(P, φ', UR, UP)
begin
   For each p_i in {p_1,...,p_m} do
      flag = true;
      For each u such that (u, p_i) ∈ UP do
         If u does not satisfy φ' then
            flag = false;
            break;
         EndIf;
```

```
        EndFor;
        If flag then return true;
      EndFor;
      return false;
   end
```

The worst-case time complexity of the above algorithm is $O(m \times |U| \times t)$, where t is the time taken to check whether a singleton satisfies a term with operators $\neg, \sqcap$ and $\sqcup$, which is polynomial in the size of input according to Theorem 15. □

LEMMA 39. SSC $\langle \neg, +, \odot \rangle$ *is in* $\mathbf{P}$.

PROOF. The general form of terms built using only $\neg, +$ and $\odot$ is $(\gamma_1 \odot \cdots \odot \gamma_n)$, where γ_i is of the form r, $\neg r$, r^+ or $(\neg r)^+$, where r is a role. Given a term ϕ with operators $\neg, +$ and $\odot$, construct another term ϕ' by removing $+$ in ϕ. It is clear that if a set U of users satisfies ϕ, then there exists $U' \subseteq U$ such that U' satisfies ϕ'. This indicates that U is safe with respect to ϕ if and only if U is safe with respect to ϕ'. Therefore, in order to show that SSC $\langle \neg, +, \odot \rangle$ is tractable, it suffices to prove that SSC $\langle \neg, \odot \rangle$ is in $\mathbf{P}$.

Given a policy $\mathsf{sp}\langle\{p_1, \ldots, p_m\}\rangle$, without loss of generality, assume that $\phi' = (\gamma_1 \odot \cdots \odot \gamma_n)$, where $\gamma_i = r$ or $\neg r$. The following algorithm checks whether $\langle U, UR, UP \rangle$ is safe with respect to ϕ'.

```
isSafe(P, φ', UR, UP)
begin
   Γ = {γ1, ... γn};
   For each pi in {p1,...,pm} do
      G_pi = ∅
      For each u such that (u,pi) ∈ UP do
         G_pi = G_pi ∪
         { γi ∈ φ' | u does not satisfy γi;}
      EndFor;
      Γ = Γ ∩ G_pi
   EndFor;
   if (Γ == ∅) return true
   else return false
end
```

In the above algorithm, G_{p_i} stores the set of sub-terms in ϕ' such that, for every $\gamma_j \in G_{p_i}$, there exists a user who has p_i but does not satisfy γ_j. At the end of the algorithm, on the one hand, if Γ contains a sub-term γ_i, it means that for every permissions p_j in $\{p_1, \ldots, p_n\}$, there exists a user u_{p_j} such that u_{p_j} has permission p_j but does not satisfy γ_i. In this case, the set of users $\{u_{p_1}, \ldots, u_{p_n}\}$ have all permissions in $\{p_1, \ldots, p_n\}$ but does not satisfy γ_i, and hence does not satisfy ϕ'. On the other hand, $\Gamma = \emptyset$ indicates that if users in U have all permissions in $\{p_1, \ldots, p_n\}$ then every sub-term γ_i in ϕ' is satisfied by a certain user in U. Therefore, there exists $U' \subseteq U$ such that U' satisfies ϕ'.

The worst-case time complexity of the above algorithm is $O(m \times |U| \times t)$, where t is the time taken to check whether a singleton satisfies a term with operators $\neg$ and $\odot$, which is polynomial according to Theorem 15. □