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

Ninghui Li
ninghui@cs.purdue.edu

Qihua Wang
wangq@cs.purdue.edu

Center for Education and Research in Information Assurance and Security
and Department of Computer Science
Purdue University

ABSTRACT

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 sensitive task to be performed by a team of at least k users. It states a high-level requirement about the task without the need to refer to individual steps in the task. While extremely important and widely used, separation of duty policies state only quantity requirements and do not capture qualification requirements on users involved in the task. In this paper, we introduce a novel algebra that enables the specification of high-level policies that combine qualification requirements with quantity requirements motivated by separation of duty considerations. A high-level policy associates a task with a term in the algebra and requires that all sets of users that perform the task satisfy the term. We give the syntax and semantics of the algebra and study algebraic properties of its operators. We also study several computational problems related to the algebra.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Complexity of proof procedures*

General Terms

Security, Theory, Languages

Keywords

Access Control, Separation of Duty, Policy Design

1. INTRODUCTION

Separation of Duty (SoD) is widely recognized as a fundamental principle in computer security [7, 18]. 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 the task. SoD has been identified as a high-level mechanism that is "at the heart of fraud and error control" [7]. An SoD policy is a high-level policy in the sense that it does not restrict which users are allowed to carry out the individual steps in a sensitive task, but rather states an overall requirement that must be satisfied by any set of users that together complete a task. In many situations, 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 that are involved to be physicians, certified nurses, certified accountants, or directors of a company. Because a high-level SoD policy states only a quantity requirement and does not express such qualification requirements, existing work addresses this by specifying such requirements at individual steps of a task. For example, if a policy requires a manager and two clerks to be involved in a task, one may divide the task into three steps and require two clerks to each perform step 1 and step 3, and a manager to perform step 2.

Specifying such requirements at the lower level of steps, however, results in the loss of the following important advantages offered by a higher-level policy. First, as the specification abstracts away details of how a task is implemented, a higher-level policy is likely to be closer to organizational policy guidelines. It would thus be easier for administrators to specify and understand such policies. Second, a high-level policy can be specified even before the actual steps involved in a task are designed. In fact, a formal specification of task-level policies may help in the process of designing steps to implement the task. Third, a task-level policy is often more flexible than a corresponding step-level policy, which can be more restrictive than necessary. For example, to enforce a task-level policy that requires a manager and two clerks, a step-level policy may require a manager to execute a particular step, which is too restrictive. Finally, a higher-level policy specification allows flexibility in the choice of the mechanism for enforcing the policy. For example, one can use either static enforcement or dynamic enforcement. In static enforcement, one ensures that any set of users that 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. When policies have to be associated with steps in a task, all the advantages discussed above are lost.

In this paper we introduce a novel algebra that enables the specification of high-level policies that combine qualification require-

CCS'06, October 30–November 3, 2006, Alexandria, Virginia, USA.

ments 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, rather than referring to the steps, simply associates a task with a term in the algebra. This policy 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 $(\texttt{Manager} \sqcup (\texttt{Clerk} \otimes \texttt{Clerk}))$.

We define the syntax and semantics of terms in the algebra, and study the algebraic properties of the operators. We show that all four binary operators are commutative and associative. We also show that $\sqcap$ and $\sqcup$ distribute over each other and both $\odot$ and $\otimes$ distribute over $\sqcup$. The four binary operators result in 12 ordered pair of operators. For the eight pairs other than the four mentioned above, distributivity does not hold.

We then study the Term Satisfiability (TSAT) problem and the Userset-Term Satisfaction (UTS) problem. The TSAT problem asks whether a given term is satisfiable at all. We show that TSAT is **NP**-complete in general and remains **NP**-complete in certain sub-algebras with only a subset of the operators. We also identify a sub-algebra whose satisfiability problem is efficiently solvable.

The UTS problem asks whether a userset satisfies a term. We show that the UTS problem is **NP**-complete in general. To better understand the properties of the operators, we also study computational complexities for the UTS problem in all sub-algebras with only a subset of the operators. We identify syntactic restrictions so that even for terms with all six operators, as long as they satisfy these restrictions, UTS can be solved efficiently. We also present a heuristic algorithm for solving UTS in the general case, and show that for terms whose size is not very large, even if they do not satisfy the syntactic restrictions, UTS can be solved in reasonable amount of time.

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. The relationships between the two are discussed.

The remainder of the paper is organized as follows. We introduce the syntax and semantics of the algebra in Section 2. We study the TSAT problem in Sections 3 and the UTS problem in Section 4. In Section 5, we discuss limitations of the algebra and extensions to it, as well as the relationship with regular expressions. We discuss related work in Section 6 and conclude with Section 7. Proofs not included in the main body are included in the appendices.

2. THE ALGEBRA

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

2.1 Syntax and Semantics

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 [21]. 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 *term*; furthermore, if ϕ_1 and ϕ_2 are terms, then $\neg\phi_1$, ϕ_1^+, $(\phi_1 \sqcup \phi_2)$, $(\phi_1 \sqcap \phi_2)$, $(\phi_1 \otimes \phi_2)$, and $(\phi_1 \odot \phi_2)$ are also terms, with the following restriction: For $\neg\phi_1$ or ϕ_1^+ to be a term, ϕ_1 must be a *unit term*, that is, it must not contain $^+$, $\otimes$, or $\odot$.

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 "$(\texttt{Manager} \sqcap \texttt{Accountant})$" requires a user that is both a Manager and an Accountant. The term "$(\texttt{Manager} \sqcap \neg\{Alice, Bob\})$" 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 "$(\texttt{Physician} \sqcup \texttt{Nurse})$" requires a user that is either a Physician or a Nurse. The term "$(\texttt{Manager} \odot \texttt{Clerk})$" requires a user who is a Manager and a user who is a Clerk; however, when one user is both a Manager and a Clerk, that user by itself also satisfies the requirement. The term "$((\mathsf{All} \otimes \mathsf{All}) \otimes \mathsf{All})$" requires 3 different users. The keyword All allows us to refer to the set of all users. The term "$\texttt{Accountant}^+$" 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 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.

Definition 3 (Satisfaction of a Term)**.** Given a configuration $\langle U, UR\rangle$, we say that a userset X *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.

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

- 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 $\langle U = \{Alice, Bob, Carl\},\ UR\rangle$, in which UR is such that: $U_{\texttt{Manager}} = \{Alice\}$ and $U_{\texttt{Clerk}} = \{Alice, Carl\}$, where $U_r = \{u \mid (u, r) \in UR\}$, we have $\{Alice\}$ satisfies the term, $\{Alice, Carl\}$ also satisfies the term, but $\{Alice,\ Bob\}$ and $\{Bob, Carl\}$ do not satisfy the term.

2.2 Examples

The following examples help illustrate the expressive power of the algebra.

- $\{Alice, Bob, Carl\} \otimes \{Alice, Bob, Carl\}$

  This term requires 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.
- $((\texttt{Manager} \odot \texttt{Accountant}) \otimes \texttt{Treasurer})$

  This term requires a `Manager`, an `Accountant`, and a `Treasurer`; the first two requirements can be satisfied by a single user.
- $((\texttt{Physician} \sqcup \texttt{Nurse}) \otimes (\texttt{Manager} \sqcap \neg\texttt{Accountant}))$

  This term requires two different users, one of which 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 requires a `Manager`, an `Accountant` and a `Treasurer`. In addition, everybody involved must be a `Clerk` and must not be $Alice$ or Bob.

Definition 4 (Value of a term)**.** Given a configuration $\langle U, UR\rangle$ and a term ϕ, we use $S_{\langle U, UR\rangle}(\phi)$ to denote the set of all usersets that satisfy ϕ under $\langle U, UR\rangle$, and call this the *value* of term ϕ under the configuration.

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 $(\texttt{Clerk} \sqcap \neg\{Alice, Bob\})^+$ means 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.

2.3 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 5 (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 $\forall\langle U, UR\rangle\ \ S_{\langle U, UR\rangle}(\phi_1) = S_{\langle U, UR\rangle}(\phi_2)$.

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 1. *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 have 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 counter example 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.

  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 $\phi_1^+ \sqcup \phi_2^+$.

2.4 Rationale of the Design of the Algebra

We now discuss the rationale for some of the design decisions for 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 [15] 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 monotonicity property, in which we call the notion of satisfaction in Definition 3 "strict satisfaction" and define a userset X satisfies a term ϕ if and only if X contains a subset that strictly satisfies ϕ. The monotonicity property follows directly. We chose our current design over the one that has monotonicity because it 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 who 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 satisfy certain 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 "$\neg$" and "$+$" The syntax of our algebra (Definition 1) restricts that the two operators "$\neg$" 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 [18]. 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 for making 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 one 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 unclear. 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 *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 the need for this. For simplicity and usability, we chose to allow $+$ only be applied on unit terms. The algebra can be extended to have two closure operators that can be applied to non-unit terms, if the need for them arises in the future.

2.5 Enforcing Policies Specified in the Algebra

To use the algebra to specify high-level security policies, the administrators first identify sensitive tasks and then for each sensitive task, specify a term such that every set of users that together perform the task must satisfy the term. For instance, a simple SoD policy that requires at least two users to perform the task can be specified as $(\mathsf{All} \otimes \mathsf{All}^+)$. Our algebra also allows the specification of more sophisticated policies based on user qualifications. In summary, a security policy is a pair $\langle task, \phi \rangle$, where ϕ is a term; it means that only usersets that satisfy ϕ can perform $task$.

Once a policy is specified, it needs to be enforced. Enforcement techniques for policies in this algebra can benefit from research in enforcement techniques for Separation of Duty policies [8, 13, 16, 20, 22]. We say that a policy $\langle task, \phi \rangle$ is monotonic if the term ϕ is monotonic, i.e., if a userset X satisfies ϕ, then any superset of X also satisfies ϕ. A monotonic policy can be enforced either statically or dynamically, whereas a policy that is not monotonic cannot be enforced statically.

To statically enforce a policy $\langle task, \phi \rangle$, one identifies the set of all permissions that are needed to perform the task, and requires that any userset such that users in the set together possess all these permissions satisfies the term ϕ. That is, one defines *static safety policies*, each of which takes the form $\mathsf{sp}\langle P, \phi \rangle$, where $P = \{p_1, \cdots, p_n\}$ is a set of permissions. Such a policy means that any userset such that all users in the set together have all permissions in P must satisfy ϕ. Note that if a userset has all permissions in P, then its superset also has all permissions in P; therefore, static enforcement is only for monotonic policies.

To dynamically enforce a policy $\langle task, \phi \rangle$, one identifies the steps in performing the task. And the system maintains a history of each instance of a task, which includes information on who has performed which step. For any task instance, one can compute the set of users (denoted as U_{past}) who have performed at least one step on the instance. Before a user u performs a step on 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. An example on dynamic enforcement of policy $\langle task, \phi \rangle$ is given as follow:

A company has a high-level policy $\langle Purchase, (\texttt{Manager} \odot (\texttt{Clerk} \otimes \texttt{Clerk})) \rangle$ which states that a `Manager` and two `Clerks` are required to purchase supplies for the company. The task $Purchase$ consists of three steps which are $MakeOrder$, $PrepareCheck$ and $SignCheck$. Step-specific requirements state that $MakeOrder$ must be performed by `Manager`, while $PrepareCheck$ and $SignCheck$ may be performed by either `Manager` or `Clerk`. Suppose $Alice$ is a `Manager` who made an order and now tries to prepare a check for the order. If $Alice$ is a `Clerk`, then the system will allow her to do so. The reason is that as long as a `Clerk` different from $Alice$ (say Bob) signs the check later, $\{Alice, Bob\}$ satisfies the high-level requirement $(\texttt{Manager} \odot (\texttt{Clerk} \otimes \texttt{Clerk}))$. If $Alice$ is not a `Clerk`, then she is not allowed to sign the check she prepared. The reason is that no matter who signs the check in future, $Alice$ plus that person cannot satisfy the high-level requirement $(\texttt{Manager} \odot (\texttt{Clerk} \otimes \texttt{Clerk}))$. Note that if $Alice$ performed both the first step and the second step, then she is precluded from performing the last step as two different `Clerks` are required to complete the task.

Dynamic enforcement can enforce both monotonic policies and policies that are not monotonic. It is also more flexible than static enforcement. Enforcement of high-level policies specified in the algebra generates many interesting open technical problems.

3. TWO TERM SATISFIABILITY PROBLEMS

Given the definitions of terms and term satisfaction, the following problems naturally arise.

The Term Satisfiability (TSAT) Problem: Given a term ϕ, determine whether there exists a configuration $\langle U, UR \rangle$ and a userset X such that X satisfies ϕ under $\langle U, UR \rangle$.

This asks whether the term ϕ is satisfiable at all. This provides a basic level of sanity check, as a term that cannot be satisfied in any configuration is probably not what a policy author intended.

The Term-Configuration Satisfiability (TCSAT) Problem: Given a term ϕ and a configuration $\langle U, UR \rangle$, determine whether there exists a userset X that satisfies ϕ under $\langle U, UR \rangle$.

This asks whether a term ϕ is satisfiable under a given configuration. This is useful when determining whether a term is meaningful in the current configuration.

The Userset-Term Satisfaction (UTS) Problem: Given a term ϕ, a configuration $\langle U, UR \rangle$, and a userset X, determine whether X satisfies ϕ under $\langle U, UR \rangle$.

This is probably the most fundamental problem related to the algebra. When an administrator specifies a policy that associates a sensitive task with a term; this means that every set of users that together perform an instance of the task must satisfy the term. To enforce this policy, one needs to check whether a userset satisfies the term and to forbid users in the set to finish the task if it does not satisfy the term. This requires solving the UTS problem.

In this section we will study TSAT and TCSAT. The UTS problem will be studied in Section 4.

3.1 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 any userset that satisfies ϕ.

In this section, we show that TSAT is **NP**-complete in general. We identify the source of intractability by identifying two special cases that are **NP**-hard. One special case involves the negation operator, and the other involves explicit sets of users. In the next section, we show that for terms that are free of negation and explicit sets of users, TSAT can be efficiently solved.

Lemma 2. TSAT *for unit terms using only roles and the operators* $\neg$, $\sqcap$, *and* $\sqcup$ *is* **NP**-*hard.*

Lemma 3. TSAT *for terms using only usersets and the operators* $\sqcap$, $\sqcup$, *and* $\odot$ *is* **NP**-*hard.*

Theorem 4. TSAT *is* **NP**-*complete.*

See Appendix B.1 for the proofs of Lemmas 2 and 3 and Theorem 4.

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

Lemmas 2 and 3 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 *UNF (Userset-and-Negation Free) terms*.

One property of UNF terms is that if a userset X satisfies a term ϕ under configuration $\langle U, UP \rangle$, then X also satisfies ϕ under $\langle U', UP' \rangle$, where $U \subseteq U'$ and $UP \subseteq UP'$. That is, because a UNF term does not have the negation operator, then if X satisfies the term under a configuration, enlarging the configuration will not make X fail to satisfy the term.

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 of 1 or 4 users. This observation leads us to introduce the notion of characteristic numbers of a UNF term.

Definition 6 (Characteristic Numbers)**.** Given a UNF term ϕ and a positive integer k, we say that k is a *characteristic number* of ϕ when there exists a userset of size k that satisfies ϕ under some configuration. A term ϕ may have more than one characteristic numbers. We use $C(\phi)$ to denote the set of all characteristic numbers of ϕ and call it the *characteristic set* of ϕ.

It follows from the definition that a UNF term ϕ is satisfiable if and only if $C(\phi) \neq \emptyset$.

Theorem 5. *The characteristic set of a UNF term can be 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 usersets and negations*
- $C(\phi_1 \odot \phi_2) = \{i \mid \exists\, c_1 \in C(\phi_1)\ \exists\, c_2 \in C(\phi_2)$ $[\, max(c_1, c_2) \leq i \leq c_1 + c_2\,]\}$
- $C(\phi_1 \otimes \phi_2) = \{\, c_1 + c_2 \mid c_1 \in C(\phi_1)\ \wedge\ c_2 \in C(\phi_2)\, \}$

The proof for the theorem is in Appendix B.2. We now illustrate the computation of characteristic set according to the theorem using some examples.

- $C(\mathsf{All} \otimes \mathsf{All} \otimes \mathsf{All}) = \{3\}$
- $C(\texttt{Manager} \odot \texttt{Accountant}) \otimes \texttt{Treasurer}) = \{2, 3\}$

  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`.
- $C((\texttt{Clerk} \sqcup \texttt{Accountant}) \otimes (\texttt{Clerk} \sqcap \texttt{Manager})) = \{2\}$

  Only one user is required for both ($\texttt{Clerk} \sqcup \texttt{Accountant}$) and ($\texttt{Clerk} \sqcap \texttt{Manager}$), and the $\otimes$ mandates that these users must be different from one another.
- $C((\texttt{Manager} \odot \texttt{Accountant} \odot \texttt{Treasurer}) \sqcap \texttt{Clerk}^+) = \{1, 2, 3\} \cap \{i | i \in [1, \infty)\} = \{1, 2, 3\}$

For the unsatisfiable term considered earlier in this section, namely $(r_1 \sqcap (r_2 \otimes r_3))$, we observe that $C(r_1 \sqcap (r_2 \otimes r_3)) = \{1\} \cap \{2\} = \emptyset$.

In Appendix B.3 we show that computing $C(\phi)$ using a straightforward algorithm that follows Theorem 5 takes at most quadratic time.

One can use $C(\phi)$ to determine whether ϕ satisfies some minimal SoD requirements. If the smallest characteristic number of a term is at least k, then we know that no $k-1$ users can satisfy the term.

We can extend the method of calculating the characteristic set stated in Theorem 5 to non-UNF terms as well, by defining $C(\neg\phi) = \{1\}$, where ϕ is a unit term, and $C(S) = 1$, where S is a 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. It remains true that for any userset X that satisfies a term ϕ, $|X| \in C(\phi)$. In other words, $C(\phi)$ gives an upperbound on the actual set of characteristic numbers of ϕ.

3.3 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 company comes up with a new security requirement for a task, it may want to know whether there exists a set of users that satisfies the new requirement and hence can perform the task under the current configuration of the company.

Observe that TCSAT is equivalent to TSAT for the terms using 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 3 and Theorem 4 that TCSAT is **NP**-complete; this is stated in the following theorem.

Theorem 6. TCSAT *is* **NP***-complete.*

4. THE USERSET-TERM SATISFACTION (UTS) PROBLEM

In Section 3, we have studied the problems of determining whether a term is satisfiable at all, as well as whether a term is satisfiable under a given configuration. In this section, we study 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 **NP**-complete. In order to understand how the operators affect the computational complexity, we consider all sub-algebras in which only some subset of the six operators $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed. For example, $\mathsf{UTS}\langle\neg, +, \sqcup, \sqcap\rangle$ denotes the sub-case of UTS where ϕ does not contain operators $\odot$ or $\otimes$, while $\mathsf{UTS}\langle\otimes\rangle$ denotes the sub-case of UTS where $\otimes$ is the only kind of operator in ϕ. $\mathsf{UTS}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ denotes the general case. Observe that unlike in the case of TSAT, whether or not to allow explicit sets of users in a term does not affect the computational complexity of UTS, because a fixed configuration is given in UTS and one can always replace each role with the corresponding set of users.

Theorem 7. *The computational complexities for* UTS *and its subcases are given in Figure 1.*

The proof of Theorem 7 is done in three parts. First, in Appendix C.1, we prove that the five cases $\mathsf{UTS}\langle\sqcup, \odot\rangle$, $\mathsf{UTS}\langle\sqcap, \odot\rangle$, $\mathsf{UTS}\langle\sqcup, \otimes\rangle$, $\mathsf{UTS}\langle\sqcap, \otimes\rangle$, and $\mathsf{UTS}\langle\odot, \otimes\rangle$ are **NP**-hard by reducing the **NP**-complete problems SET COVERING, DOMATIC NUMBER, and SET PACKING to them. Second, in Appendix C.2, we prove that the general case $\mathsf{UTS}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in **NP**. Finally, the tractable cases are discussed in Section 4.1, where 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 **P** in Figure 1.

4.1 UTS is Tractable for Terms in Canonical Forms

From Figure 1, UTS is **NP**-complete in general 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 7 (Canonical Forms for Terms)**.** The canonical forms for terms are defined as follows:

- A term is *in level-1 canonical form* (called an 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 (when there are two or more sub-terms) these sub-terms are connected only by the operator $\sqcap$.

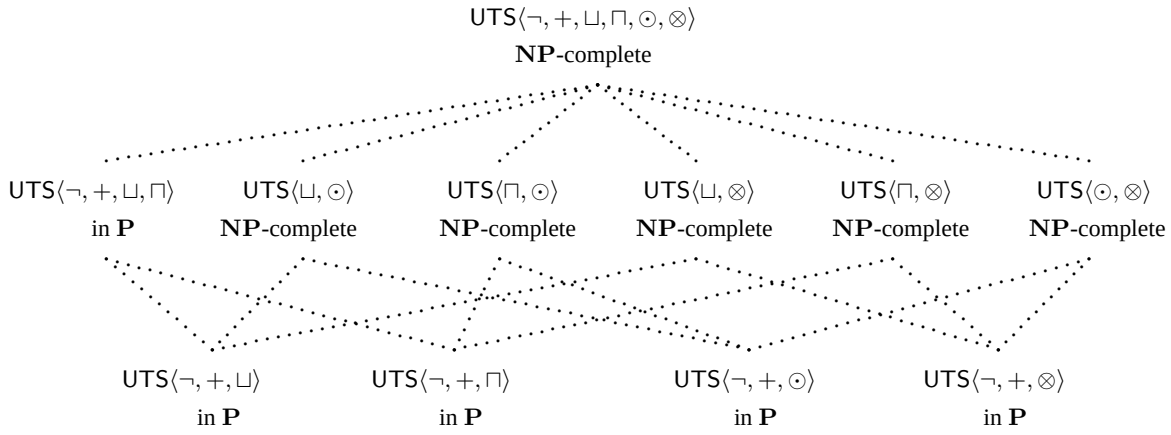

Figure 1: Various sub-cases of the Userset Term Satisfaction (UTS) problem and the corresponding time-complexity. Some sub-cases are omitted from the figure, as their time-complexities are implied from what are in the figure.

- 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 (when there are two or more sub-terms) 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 $i \in [1, 4]$.

Theorem 8. *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. We first 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 an 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 $^+$, from the algebraic property, it is equivalent a 1CF term. For example, $(t_1^+ \sqcap t_2^+)$ is equivalent to $(\phi_1 \sqcap \phi_2)^+$. Hence, any 2CF term can be revised to an equivalent 1CF term. We assume that the revision 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 an 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 $t_1, t_2, \cdots, t_m$; 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 it has size the same as $\max(|X|, m)$, then X satisfies P; otherwise, X does not satisfy P.

The case that a 3CF term contains $+$ is more complicated, so is the case for a 4CF term. Because of space limitation, we give the proof for the case of a 4CF term (which subsumes the case for a 3CF term) in Appendix C.3.

Terms in canonical form 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 involving combinations of multiple operators in different ordering, and by studying how to handle each binary operator individually and examining how combinations of them can still be handled efficiently.

4.2 An Algorithm for UTS

We have shown that for terms that are in canonical forms, it is efficient to check whether a userset satisfies them or not. However, it is not necessary to restrict the use of the algebra to only such terms. Even if one writes policies that use terms that are not in canonical forms, these terms may not be in the pathological cases that lead to intractability, or they may not be very large. We now present a heuristic algorithm for UTS that works for all terms. We show through experimental results that this algorithm works well for instances of UTS with complicated terms and relative large usersets.

To determine whether a userset X satisfies a term ϕ under a configuration $\langle U, UR\rangle$, our algorithm first computes the syntax tree T of ϕ. Given the syntax tree, there are two approaches. The first one is top-down processing. One starts with X and the root of the syntax tree; if the root is the operator $\odot$, then for each subset $X_1 \subseteq X$, one recursively checks whether X_1 satisfies the left child, and if it does, one tries all $X_2 \subseteq X$ such that $X_1 \cup X_2 = X$ and checks whether X_2 satisfies the right child. Other operators can be handled similarly. The second approach is bottom-up processing. One starts with unit terms. For each unit term, one calculates all subsets of X that satisfy the term. One then goes bottom-up to calculate those for each node in the syntax tree.

Our algorithm combines top-down processing with bottom-up processing. It first performs bottom-up processing until encounter-

ing nodes such that bottom-up processing becomes too expensive. For example, if each user in a set Y can satisfy t, then node t^+ can be satisfied by the $2^{|Y|} - 1$ non-empty subsets of Y; so we avoid bottom-up processing for t^+. After the bottom-up phase, the algorithm performs top-down search. When the search encounters a node that has been bottom-up processed, it can perform a lookup. Our algorithm also includes several optimizations to improve top-down search. For example, it computes the characteristic set for each sub-term so as to prune the subsets of X that need to be checked; it also sorts sub-terms of ϕ according to the size of their characteristic sets so that sub-terms that are "harder to be satisfied" are processed first.

We prototyped both the above algorithm and the algorithm for terms in canonical forms, and have performed some experiments. Our prototypes are written in Java, and our experiments were carried out on a Workstation with a 3.2GHz Pentium 4 CPU and 512MB RAM. Some of our experimental results are presented in Table 1. As we can see in Table 1, UTS is efficiently solvable for terms in canonical form. Furthermore, our algorithm for canonical form scales well over the size of userset, as in Test 2 and Test 3, increasing userset size from 25 to 50 only results in 1 millisecond's increment on runtime. Finally, experimental data in Test 4 and Test 5 indicates that UTS can be solved quickly even for complicated terms that are not in canonical form.

Further improvements and optimizations can be made to our algorithms and prototypes; they are beyond the scope of this paper. Our goal here is to verify that UTS can be solved in reasonable amount of time for complicated terms, even though the problem is **NP**-complete in general.

5. DISCUSSIONS AND OPEN PROBLEMS

In this section we discuss a few small extensions to the algebra, the similarities and differences with regular expressions, and expressive power limitations of the algebra.

5.1 Extensions to 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 2.4, SoD policies are monotonic, as are policies in McLean's formulation of N-person policies [15]; 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 policies mentioned in Section 2.4 is policies stating that every user involved in a task must satisfy certain requirement and there need to be at least a certain number of users involved. Let ϕ be a unit term that expresses the requirement. A policy that requires 2 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. Using 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.

5.2 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 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 corresponds to a single element. In regular expression, the closure operator ($*$ or $+$) can be applied to arbitrary subexpressions. Our algebra requires that repetition (using operator $^+$) can only be applied to unit terms. As we discussed in Section 2.4, 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 relax such restriction so that the algebra can define a wider range of sets. The remaining binary operators $\odot$ and $\sqcap$ have flavor of set intersection, which does not have counterparts in regular expressions.

Observe that determining whether a string satisfies a regular expression is in **NL**-complete, where **NL** stands for Nondeterministic Logarithmic-Space, and is contained in **P**. On the other hand, determining whether a userset satisfies a term is **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.

5.3 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 $\{0, 1\}$ 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 equal number of users who are r_1 as those that are r_2. However, if we allow the application of $+$ to non-unit terms and define it as follows:

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

	Term	In CF?	Userset Size	Runtime (ms)
1	$((r_1 \sqcap r_2) \otimes (r_1 \sqcup r_3) \otimes \neg r_4 \otimes r_5 \otimes \mathsf{All})$	Yes	5	< 1
2	$((((r_1 \sqcap r_2) \otimes r_3 \otimes (r_1 \sqcup r_5)^+) \odot (r_4 \otimes (\neg r_1)^+)))$	Yes	25	15
3	$((((r_1 \sqcap r_2) \otimes r_3 \otimes (r_1 \sqcup r_5)^+) \odot (r_4 \otimes (\neg r_1)^+)))$	Yes	50	16
4	$((((r_1 \otimes r_2) \sqcap (r_3 \odot r_5)) \odot (r_1 \sqcup r_3)) \otimes \mathsf{All}^+)$	No	15	16
5	$(((r_1 \sqcap r_2)^+ \odot r_3 \odot \mathsf{All}^+) \otimes (r_6 \sqcap r_1) \otimes ((r_2^+ \odot (\neg r_1 \otimes r_2)^+ \odot r_4) \sqcap (r_2 \sqcup \neg r_5)^+))$	No	50	219

Table 1: A table that shows the runtime of testing whether a userset satisfies a term.

then we can express the policy that requires equal number of users who are r_1 as those that are r_2 using the term

$$(r_1 \sqcap r_2)^+ \otimes ((r_1 \sqcap \neg r_2) \otimes (r_2 \sqcap \neg r_1))^+ \otimes (\neg r_1 \sqcap \neg r_2)^+.$$

Even with the extension, there are sets of usersets that cannot be expressed. For example, it seems unlikely that one can express a policy that requires that the number of users who are member of r_1 is the same as the square of the number of users who are r_2. Further discussions of expressive power and more general algebras are interesting future research topics and are beyond the scope of this paper.

6. 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 [18] under the name "separation of privilege." Clark and Wilson's commercial security policy for integrity [7] identified SoD along with well-formed transactions as two major mechanisms of fraud and error control. Nash and Poland [16] 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 data item. In the latter, users are constrained a-priori from performing certain steps.

Sandhu [19, 20] 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.

There exists a wealth of literature [1, 2, 8, 10, 11, 12, 22, 23] on constraints in the context of RBAC. They either proposed and classified new kinds of constraints [10, 22] or proposed new languages for specifying sophisticated constraints [1, 2, 8, 12, 23]. 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.

There has also been recent interest in static and dynamic constraints to enforce separation of duty in workflow systems. Atluri and Huang [3] proposed an access control model for workflow environments, which supports temporal constraints. Bertino et al. [4] proposed a language for specifying static and dynamic constraints for separation of duty in role-based workflow systems. In these works, security requirements are associated with individual steps in the workflows.

McLean [15] introduced a framework that includes various mandatory access control models. These models differ in which users are allowed to change the security levels. They 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.

Several algebras have been proposed for combining security policies. These include the work by Bonatti et al. [5, 6], Wijesekera and Jajodia [24], Pincus and Wing [17]. 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. [14]. In [14], 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.

7. 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. A high-level policy associates a task with a term in the algebra and requires that all sets of users that perform the task satisfy the term. Specifying security policies at the task level has a number of advantages over the current approach of specifying such policies at the individual step level. Our algebra has two unary and four binary operators, and is expressive enough to specify a large number of diverse policies. We have also studied algebraic properties of these operators and several computational problems related to the algebra, including determining whether a term is satisfiable at all, determining whether a term is satisfiable under a given configuration, and determining whether a userset satisfies a term under a given configuration. 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.

Acknowledgement

This work is supported by NSF CNS-0448204 (CAREER: Access Control Policy Verification Through Security Analysis And Insider

Threat Assessment), and by sponsors of CERIAS. We thank Mahesh V. Tripunitara for helpful discussions. We also thank the anonymous reviewers for their helpful comments.

8. REFERENCES

[1] G.-J. Ahn and R. S. Sandhu. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the 4th Workshop on Role-Based Access Control*, pages 43–54, 1999.

[2] G.-J. Ahn and R. S. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security*, 3(4):207–226, Nov. 2000.

[3] V. Atluri and W. Huang. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS)*, pages 44–64, 1996.

[4] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, Feb. 1999.

[5] P. Bonatti, S. de Capitani di Vimercati, and P. Samarati. A modular approach to composing access control policies. In *Proceedings of the 7th ACM conference on Computer and Communications Security (CCS)*, pages 164–173, Nov. 2000.

[6] P. Bonatti, S. de Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, Feb. 2002.

[7] D. D. Clark and D. R. Wilson. A comparision of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, May 1987.

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

[9] M. R. Garey and D. J. Johnson. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[10] V. D. Gligor, S. I. Gavrila, and D. F. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 172–183, May 1998.

[11] T. Jaeger. On the increasing importance of constraints. In *Proceedings of ACM Workshop on Role-Based Access Control*, pages 33–42, 1999.

[12] T. Jaeger and J. E. Tidswell. Practical safety in flexible access control models. *ACM Transactions on Information and System Security*, 4(2):158–190, May 2001.

[13] N. Li, Z. Bizri, and M. V. Tripunitara. On mutually-exclusive roles and separation of duty. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS-11)*, pages 42–51. ACM Press, Oct. 2004.

[14] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[15] J. McLean. The algebra of security. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 2–7, Apr. 1988.

[16] M. J. Nash and K. R. Poland. Some conundrums concerning separation of duty. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 201–209, May 1990.

[17] J. Pincus and J. M. Wing. Towards an algebra for security policies (extended abstract). In *Proceedings of ICATPN 2005*, number 3536 in LNCS, pages 17–25. Springer, 2005.

[18] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[19] R. Sandhu. Separation of duties in computerized information systems. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*, Sept. 1990.

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

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[22] T. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *Proceedings of The 10th Computer Security Foundations Workshop*, pages 183–194. IEEE Computer Society Press, June 1997.

[23] J. Tidswell and T. Jaeger. An access control model for simplifying constraint expression. In *Proceedings of ACM Conference on Computer and Communications Security*, pages 154–163, 2000.

[24] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and Systems Security (TISSEC)*, 6(2):286–325, May 2003.

APPENDIX

A. PROOFS FOR THEOREMS IN SECTION 2

Proof for Theorem 1 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 satisfies 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 counter example 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. Counter example: 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.
   Counter example: 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 $((\phi_1 \sqcap r_3) \odot (\phi_1 \sqcap r_4))$.
   Counter example: 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. Counter example: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \emptyset$ and $U_{r_3} = \emptyset$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcup r_2) \odot (r_1 \sqcup r_3))$, but does not strictly satisfy $(r_1 \sqcup (r_2 \odot r_3))$.

   (d) The operator $\sqcup$ does not distribute over $\otimes$. Neither direction holds.
   Counter example: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \emptyset$ and $U_{r_3} = \emptyset$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcup r_2) \otimes (r_1 \sqcup r_3))$, but does not satisfy $(r_1 \sqcup (r_2 \otimes r_3))$.
   Counter example: Let $U_{r_1} = \{u_1\}$, $U_{r_2} = \emptyset$ and $U_{r_3} = \emptyset$, 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. Counter example: 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.
   Counter example: 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 $(r_1 \otimes (r_2 \sqcap r_3))$.
   Counter example: 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.
   Counter example: 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 $(r_1 \odot (r_2 \otimes r_3))$.
   Counter example: 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. Counter example: 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 straight forward by definition of $\neg, \sqcap$ and $\sqcup$.

B. PROOF FOR THEOREMS IN SECTION 3

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 $(r_1 \ \mathsf{op} \cdots \mathsf{op} \ r_n)$. Given $R = \{r_1, \cdots, r_m\}$, $(\mathsf{op} R)$ denotes $(r_1 \ \mathsf{op} \cdots \mathsf{op} \ r_m)$.

B.1 Proof for Lemma 2, Lemma 3, and Theorem 4

Proof for Lemma 2: TSAT with just role, $\neg$, $\sqcap$, and $\sqcup$ is NP-hard.

We reduce the **NP**-complete SAT problem to TSAT problem of terms consisting of just role, $\neg$, $\sqcap$, and $\sqcup$. Given a propositional logic formula e, let $\{v_1, \cdots, v_n\}$ be the set of propositional variables that appear in e. Construct a unit term ϕ by substituting every occurrence of v_i $(i \in [1, n])$ in e with 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 $\sqcup$. 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 then 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 for Lemma 3: TSAT with just userset, $\sqcap$, $\sqcup$, and $\odot$ is NP-hard.

We reduce the **NP**-complete SET COVERING problem to the TSAT problem of terms consisting of just sets of users, $\sqcap$, $\sqcup$, and $\odot$. In the set covering problem, we are given a finite set $U = \{u_1, \cdots, u_n\}$, a family $F = \{U_1, \cdots, U_m\}$ of subsets of U, and an integer k no larger than m, and we ask whether there are k sets in family F whose union is U.

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 $U_j = \{u_{j_1}, u_{j_2}, \cdots, u_{j_x}\}$. It is clear that ϕ_i can only be satisfied by U_i. Finally, we construct a term $\phi = ((\bigodot_k(\bigsqcup_{i=1}^n \phi_i)) \sqcap (\bigodot_{i=1}^m \{u_i\}))$. Since $(\bigodot_{i=1}^m \{u_i\})$ can be satisfied only by U, U is the only userset that could 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 $(\bigodot_k(\bigsqcup_{i=1}^n \phi_i))$, which means that there exist k sets $U'_1, \cdots, U'_k$ such that $\bigcup_{i=1}^k U'_i = U$ and each U'_i satisfies $(\bigsqcup_{i=1}^n \phi_i)$. 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 any $i \in [1, k]$, U_i satisfies ϕ_i and thus satisfies $(\bigsqcup_{i=1}^n \phi_i)$. Therefore, U satisfies $(\bigodot_k(\bigsqcup_{i=1}^n \phi_i))$. Since U also satisfies $(\bigodot_{i=1}^m \{u_i\})$, U satisfies $((\bigodot_k(\bigsqcup_{i=1}^n \phi_i)) \sqcap (\bigodot_{i=1}^m \{u_i\}))$.

Proof for Theorem 4: TSAT is NP-complete.

Since we have already proved that certain subcases of TSAT is **NP**-hard, to prove the theorem, we just need to show that the problem is in **NP**.

To prove that the problem is in **NP**, we need to show that there exists a nondeterministic Turing machine M that is able to generate a userset X and a configuration $\langle U, UR\rangle$ and then check whether X satisfies ϕ under $\langle U, UR\rangle$ in polynomial time. In Lemma 15, we show that checking whether a userset satisfies a term under a given configuration is in **NP**. In other words, one can design a nondeterministic Turing machine N that checks whether a userset satisfies a term in polynomial time. Here, M is the same as N except that M nondeterministically generates userset X and configuration $\langle U, UR\rangle$ at the very beginning. It is obvious that the additional steps M taken can be done in polynomial time.

B.2 Proof for Theorem 5

- $C(\mathsf{All}) = C(r) = \{1\}$ is straightforward.
- That $C(\phi_1 \sqcup \phi_2) = C(\phi_1) \cup C(\phi_2)$ follows from the definition of satisfaction (Definition 3).
- That $C(\phi_1 \sqcap \phi_2) = C(\phi_1) \cap C(\phi_2)$ follows from the definition of satisfaction (Definition 3).
- $C(\phi^+) = \{i \mid i \in [1, \infty)\}$: It follows from the computation of $C(\mathsf{All}), C(r), C(\phi_1 \sqcup \phi_2), C(\phi_1 \sqcap \phi_2)$ that the characteristic set of any unit term ϕ free of usersets and negations is $\{1\}$. Given a configuration $\langle U, UR\rangle$ and a singleton $\{u_1\}$ such that $\{u_1\}$ satisfies ϕ, we can make $n-1$ copies of u_1 for any $n \geq 2$ so that the n users together satisfies ϕ^+. In other words, ϕ^+ may be satisfied by n users for any $n \geq 1$.
- $C(\phi_1 \odot \phi_2) = \{\, i \mid \exists\, c_1 \in C(\phi_1)\ \ \exists\, c_2 \in C(\phi_2)\ \ [\, max(c_1, c_2) \leq i \leq c_1 + c_2\,]\}$

  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 definition of characteristic set, 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 $|X| = k$.
- $C(\phi_1 \otimes \phi_2) = \{\, c_1 + c_2 \mid c_1 \in C(\phi_1)\ \wedge\ c_2 \in C(\phi_2)\,\}$

  A 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 characteristic set, $|X_1| \in C(\phi_1)$ 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 characteristic number, 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 $|X| = |X_1| + |X_2| = c_1 + c_2$.

B.3 Proof that computing the characteristic set takes quadratic time

A straightforward algorithm to compute $C(\phi)$ is to follow Theorem 5. We now show that this can be done in time quadratic to the size of ϕ, denoted by $|\phi|$, which is defined to be the number of occurrences of atomic terms in ϕ. Using induction on the structure of ϕ, it is easy to show that $|\phi|$ is equal to the number of binary operators in ϕ plus 1. We need the following lemma to prove this.

Lemma 9. *$C(\phi)$ either is equal to a subset of $\{1, 2, \cdots, |\phi|\}$ or $W \cup \{i | i \in [|\phi|, \infty)\}$, where W is a subset of $\{1, 2, \cdots, |\phi|\}$.*

PROOF. Proof by induction on the structure of term ϕ.

Based case: When ϕ is a unit term, $C(\phi) = \{1\}$ is a subset of $\{1, 2, \cdots, |\phi|\}$. Otherwise, when ϕ is in the form of ϕ_1^+, where ϕ_1 is a unit term, from Theorem 5, $C(\phi) = \{i | i \in [1, \infty)\} = W \cup \{i | i \in [|\phi|, \infty)\}$, where $W = \{1, 2, \cdots, |\phi|\}$.

Induction 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 $\{1, 2, \cdots, |\phi_1|\}$ and W_2 denote a subset of $\{1, 2, \cdots, |\phi_2|\}$. We have the following three cases:

Case 1: $C(\phi_1) = W_1$ and $C(\phi_2) = W_2$. Since $|\phi| = |\phi_1| + |\phi_2|$, it follows from Theorem 5 that $C(\phi) = W$, where W is a subset of $\{1, 2, \cdots, |\phi|\}$.

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 $C(\phi_2) = W_2 \cup \{i \mid i \in [|\phi_2|, \infty)\}$. We compute $C(\phi)$ according to op:

- $\mathsf{op} = \sqcup$: $C(\phi) = C(\phi_1) \cup C(\phi_2) = W_1 \cup W_2 \cup \{i | i \in [|\phi_2|, \infty)\} = W_1 \cup W_2 \cup \{i \mid i \in [|\phi_2|, |\phi|)\} \cup \{i \mid i \in [|\phi|, \infty)\}$, in which $W_1 \cup W_2 \cup \{i \mid i \in [|\phi_2|, |\phi|)\}$ is a subset of $\{1, 2, \cdots, |\phi|\}$.
- $\mathsf{op} = \sqcap$: $C(\phi) = C(\phi_1) \cap C(\phi_2)$ which is a subset of W_1.
- $\mathsf{op} = \odot$: $C(\phi) = \{i \mid \exists c_1 \in W_1\ \exists c_2 \in W_2\ [max(c_1, c_2) \leq i \leq c_1 + c_2]\} \cup \{i \mid i \in [max(min(W_1), |\phi_2|), \infty)\} = \{i \mid \exists c_1 \in W_1\ \exists c_2 \in W_2\ [max(c_1, c_2) \leq i \leq c_1 + c_2]\} \cup \{i \mid i \in [max(min(W_1), |\phi_2|), |\phi|]\} \cup \{i \mid i \in [|\phi|, \infty)\}$. Note that $\{i \mid \exists c_1 \in W_1\ \exists c_2 \in W_2\ [max(c_1, c_2) \leq i \leq c_1 +$

$c_2]\} \cup \{i \mid i \in [max(min(W_1), |\phi_2|), |\phi|]\}$ is a subset of $\{1, 2, \cdots, |\phi|\}$.

- $\mathsf{op} = \otimes$: $C(\phi) = \{c_1 + c_2 | c_1 \in W_1 \wedge (c_2 \in W_2 \vee c_2 \in [|\phi_2|, \infty))\} = \{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \cup \{i | i \in [min(W_1) + |\phi_2|, \infty)\} = \{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \cup \{i | i \in [min(W_1) + |\phi_2|, |\phi|]\} \cup \{i | i \in [|\phi|, \infty)\}$. Note that $\{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \cup \{i | i \in [min(W_1) + |\phi_2|, |\phi|]\}$ is a subset of $\{1, 2, \cdots, |\phi|\}$.

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.

Given $C(\phi_1)$ and $C(\phi_2)$, Lemma 9 states that $C(\phi_i)$ contains at most $|\phi_i|$ ($i \in \{1, 2\}$) distinct numbers plus a consecutive numerical range, where the range may be treated as a unit during computation. Therefore, calculating $C(\phi_1 \ \mathsf{op} \ \phi_2)$ takes time at most linear in $|\phi_1| + |\phi_2|$. Thus, for each operator in ϕ, the algorithm takes time $O(|\phi|)$; therefore, it takes time at most quadratic in $|\phi|$ to calculate $C(\phi)$. Because ϕ is satisfiable if and only if $C(\phi) \neq \emptyset$, it follows that one can decide whether ϕ is satisfiable or not in time $O(|\phi|^2)$.

C. PROOFS FOR 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 $(r_1 \ \mathsf{op} \cdots \mathsf{op} \ r_n)$. Given $R = \{r_1, \cdots, r_m\}$, $(\mathsf{op} R)$ denotes $(r_1 \ \mathsf{op} \cdots \mathsf{op} \ r_m)$.

C.1 The five intractability subcases of UTS

Lemma 10. UTS $\langle \sqcup, \odot \rangle$ *is* **NP**-*hard.*

PROOF. We use a reduction from the **NP**-complete SET COVERING problem [9]. In the set covering problem, we are given a finite set $S = \{e_1, \cdots, e_n\}$, family of S's subsets $F = \{S_1, \cdots, S_m\}$, and an integer $k < m$, and we ask whether there are k sets in family F whose union is S. 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 $U = \{u_1, \cdots, u_n\}$ 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 family F whose union is S. On the one hand, if U satisfies ϕ, by definition, U has k subsets $U_1, \cdots, U_k$ 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, if k subsets in F cover S, without loss of generality, assume that $\bigcup_{i=1}^k S_i = S$. In this case, we divide U into k sets $U_1, \cdots, U_k$ 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 11. UTS $\langle \sqcap, \odot \rangle$ *is* **NP**-*hard.*

PROOF. We use a reduction from the **NP**-complete SET COVERING problem [9]. Given $S = \{e_1, \cdots, e_n\}$, a family of S's subsets $F = \{S_1, \cdots, S_m\}$, and an integer $k < m$, our reduction maps each element $e_j \in S$ to a role r_j and each subset $S_i \in F$ to a user u_i. We construct a configuration $\langle U, UR \rangle$ such that $U = \{u_1, \cdots, u_m\}$ 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 one hand, assume 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 a subset U' of 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 $\{r_1, \cdots, r_n\}$. Without loss of generality, suppose $U' = \{u_1, \cdots, u_t\}$, where $t \leq k$. As $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_1, \cdots, S_t\}$ is S. The answer to the set covering problem is "yes". On the other hand, if k subsets in F cover S, without loss of generality, assume that $\bigcup_{i=1}^k S_i = S$. From the construction of UR, users $u_1, \cdots, u_k$ together have membership of all roles in $\{r_1, \cdots, r_n\}$. In this case, $\{u_1, \cdots, u_k\}$ satisfies $(\bigodot_{i=1}^n r_i)$. Also, $\{u_1, \cdots, u_k\}$ satisfies $(\bigodot_k \mathsf{All})$. Hence, $\{u_1, \cdots, u_k\}$ 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 12. UTS $\langle \odot, \otimes \rangle$ *is* **NP**-*hard.*

PROOF. We use a reduction from the **NP**-complete DOMATIC NUMBER problem [9]. Given a graph $G(V, E)$, the Domatic Number problem asks whether V can be partitioned into k disjoint nonempty sets $V_1, V_2, \cdots, V_k$, 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 $U = \{u_1, u_2, \cdots, u_n\}$ and $R = \{r_1, r_2, \cdots, r_n\}$, 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 $r_1, r_2, \cdots, r_n$. Therefore, the answer to the Domatic Number problem is "yes" if and only if U satisfies ϕ under $\langle U, UR \rangle$.

Lemma 13. UTS $\langle \otimes, \sqcup \rangle$ *is* **NP**-*hard.*

PROOF. We use a reduction from the **NP**-complete SET PACKING problem [9], which asks, given a finite set $S = \{e_1, \cdots, e_n\}$, a family of S's subsets $F = \{S_1, \cdots, S_m\}$, and an integer k, whether there are k subsets in family F such that these k sets are pairwise disjoint. Without loss of generality, we assume that $S_i \not\subseteq S_j$ if $i \neq j$. (If $S_i \subseteq S_j$, one can remove S_j without affecting the answer.) Let $U = \{u_0, u_1, \cdots, u_n\}$, $R = \{r_1, \cdots, r_n\}$ and $UR = \{(u_i, r_i) \mid 1 \leq i \leq n\}$. In particular, 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 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 $U_j = \{u_i \mid e_i \in S_j\}$. 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 $S_1, \cdots, S_k$ are k pairwise disjoint sets. Then, $U_1, \cdots, U_k$ 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 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 $\hat{U}_1, \cdots, \hat{U}_k$, 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 $\hat{U}_1, \cdots, \hat{U}_k$ are pairwise disjoint indicates that $U_{a_1}, \cdots, U_{a_k}$ are also pairwise disjoint. Therefore, their corresponding sets $S_{a_1}, \cdots, S_{a_k}$ are pairwise disjoint. The answer to the Set Packing problem is "yes".

Lemma 14. 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 $F = \{S_1, \cdots, S_m\}$ of subsets of a finite set S and an integer k no larger than m, whether there are k sets in family F whose union is S.

Given $S = \{e_1, \cdots, e_n\}$ and a family of S's subsets $F = \{S_1, \cdots, S_m\}$, let $U = \{u_1, u_2, \cdots, u_m\}$, $R = \{r_1, r_2, \cdots, r_n\}$ 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 $k < m$.

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 from 1 to n. This means that users in U' together have membership of all roles in $\{r_1, \cdots, r_n\}$. Suppose $U' = \{u_{a_1}, \cdots, u_{a_k}\}$, where $a_i \in [1,m]$. As $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_{a_1}, \cdots, S_{a_k}\}$ 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 $u_1, \cdots, u_k$ together have membership of $r_1, \cdots, r_n$. In this case, $\{u_1, \cdots, u_k\}$ satisfies $r_i \otimes \bigotimes_{k-1} \mathsf{All}$ for every i from 1 to n. Since $k < m$, $\{u_1, \cdots, u_k\}$ is a strict subset of U. Therefore, U can be divided into two nonempty subset $\{u_1, \cdots, u_k\}$ and $U - \{u_1, \cdots, u_k\}$ such that $\{u_1, \cdots, u_k\}$ satisfies $(\bigsqcap_{i=1}^{n} r_i \otimes \bigotimes_{k-1} \mathsf{All})$ and $U - \{u_1, \cdots, u_k\}$ satisfies $(\bigotimes_{m-k} \mathsf{All})$. In other words, U satisfies ϕ.

C.2 Proof that UTS is in NP

Lemma 15. UTS $\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ *is in* **NP**.

PROOF. To determine whether a userset X satisfies a term ϕ under a configuration $\langle U, UR \rangle$, we first compute the syntax tree T of ϕ. When constructing T, an 1CF term (i.e., a term of the form ϕ or ϕ^+, where ϕ is a unit term, see Definition 7) is treated as a unit and is not further decomposed. In other words, the leaves in T correspond to sub-terms of ϕ that are 1CF terms and the inner nodes correspond to binary operators connecting these sub-terms. If X satisfies ϕ, then for each node in the tree, there exists a subset of X that satisfies the term rooted at that node, and the root of T corresponds to the set X. After these subsets are guessed and labeled with each node, verifying that they indeed satisfy the terms can be done efficiently. Verifying that a userset satisfies a 1CF term can be done efficiently. (See Proof of Theorem 8.) When the two children of a node are verified, checking that node is labeled correctly can also be done efficiently. Therefore, the problem is in **NP**.

C.3 The tractable cases

Lemma 16. UTS *for 4CF terms is in* **P**.

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

Given a userset $X = \{u_1, \cdots, u_n\}$ 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 $1 \le k \le n$, 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 matching has size 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, then 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 above 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 $X' = X$.