BEYOND SEPARATION OF DUTY: AN ALGEBRA FOR SPECIFYING HIGH-LEVEL SECURITY POLICIES

by Ninghui Li, Qihua Wang, Mahesh Tripunitara

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

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

Ninghui Li Qihua Wang Mahesh V. Tripunitara
Department of Computer Science and CERIAS
Purdue University
West Lafayette, IN, USA
{ninghui, wangq, mtripuni}@cs.purdue.edu

November 22, 2005

Abstract

A separation of duty policy 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 cannot 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 user qualification requirements with 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. Our algebra has four operators. We give the syntax and semantics of the algebra and study algebraic properties of these operators. We also study several computational problems related to the algebra. 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.

1 Introduction

Separation of Duty (SoD) is widely recognized as a fundamental principle in computer security [6, 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" [6]. 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 cannot express such 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 high-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. Finally, step-level policies can be more restrictive than necessary. For example, to enforce a high-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. 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 to enable the specification of high-level policies that consider user qualifications. 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 operators: $\sqcup, \sqcap, \odot, \otimes$. 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 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 operators result in 12 ordered pair of operators. For the eight pairs other than the four mentioned above, distributivity does not hold. We also study the following problems.

1. *The Term Satisfiability problem*: This asks whether a given term is satisfiable at all. We show that this problem can be efficiently solved by calculating what we call the characteristic set of a term.

2. *The Userset-Term Satisfaction (*UTS*) problem*: This asks whether a userset satisfies a term. We show that the UTS problem is $\mathbf{NP}$-complete in general. To better understand the properties of the four operators, we also study computational complexities for the UTS problem in all sub-algebras with only a subset of the four operators. For example, we show that when only $\otimes$ is allowed, the UTS problem can be reduced to the bipartite graph maximal matching problem, and is therefore efficiently decidable. However, combining $\otimes$ with any of $\sqcup, \sqcap, \odot$ makes UTS $\mathbf{NP}$-complete.

   We also identify syntactic restrictions so that even for terms with all four operators, as long as they satisfy these restrictions, UTS can be solved efficiently.

3. *The Static Safety Checking (*SSC*) problem:* This problem results from the static enforcement of a high-level policy specified using the algebra. It asks, given a term, a set of permissions that are necessary and sufficient to perform a task, and an access control state, whether every set of users who together have all permissions in the set satisfy the term. We show that the SSC problem is $\mathbf{coNP}$-hard in general, and is in $\mathbf{coNP}^{\mathbf{NP}}$, which is a complexity class in the polynomial hierarchy. We also give the computational complexities of all subcases where only some subset of the operators are allowed.

The remainder of the paper is organized as follows. We introduce the syntax and semantics of the algebra in Section 2. We study the three problems identified above in Sections 3, 4, and 5, respectively. We discuss related work in Section 6 and conclude with Section 7. In Appendices A to E, we present additional details about the algebra and proofs not included in the main body.

2 The Syntax and Semantics of the Algebra

In this section, we introduce the syntax and semantics for the algebra. 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) [21]. In our algebra, a role is simply a set of users with a name. The notion of roles can be replaced by groups or user attributes. We choose to call them roles to make the discussions more concrete.

Definition 1 (Terms in the Algebra) A *term* in the algebra is defined as follows:

- A role is a term, and the keyword All is a term. They are called *atomic terms*.
- Given two terms ϕ_1 and ϕ_2, the following are terms: $(\phi_1 \sqcup \phi_2)$, $(\phi_1 \sqcap \phi_2)$, $(\phi_1 \otimes \phi_2)$, and $(\phi_1 \odot \phi_2)$.

We now give several simple examples of terms in the algebra. These examples illustrate the intuition behind the four operators.

- $(\texttt{Manager} \sqcap \texttt{Accountant})$

  This term requires a user that is both a `Manager` and an `Accountant`.

- $(\texttt{Physician} \sqcup \texttt{Nurse})$

  This term requires a user that is either a `Physician` or a `Nurse`.

- $(\texttt{Manager} \odot \texttt{Clerk})$

  This term requires a `Manager` and a `Clerk`; when a user is both a `Manager` and a `Clerk`, that user by itself also satisfies the requirement.

- $((\mathsf{All} \otimes \mathsf{All}) \otimes \mathsf{All})$

  This term requires 3 different users. The keyword All allows us to refer to the set of all users.

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

Definition 2 (Configurations) A *configuration* is given by a binary relation $UR \subseteq \mathcal{U} \times \mathcal{R}$, where $\mathcal{U}$ represents the set of all users, and $\mathcal{R}$ represents the set of all roles. When $(u, r) \in UR$, we say that u is a member of the role r.

Note that the configuration 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.

Below we first define the notion of strict satisfaction, which captures the situation that a userset satisfies a term in a way that every user in the set is used to satisfy some component of the term. We then define the notion of term satisfaction.

Definition 3 (Strict Satisfaction of a Term) We say that a set X of users *strictly satisfies* a term ϕ under a configuration UR if one of the following holds:

- The term ϕ is the keyword All, and X is a singleton set.
- The term ϕ is a role r, and X is a singleton set $\{u\}$ such that $(u, r) \in UR$.
- The term ϕ is of the form $(\phi_1 \sqcup \phi_2)$, and either X strictly satisfies ϕ_1 under UR or X strictly satisfies ϕ_2 under UR.
- The term ϕ is of the form $(\phi_1 \sqcap \phi_2)$, and X strictly satisfies both ϕ_1 and ϕ_2 under UR.
- 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 strictly satisfies ϕ_1 under UR, and X_2 strictly satisfies ϕ_2 under UR.
- 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 strictly satisfies ϕ_1 under UR, and X_2 strictly satisfies ϕ_2 under UR. This differs from the definition for $\otimes$ in that this does not require $X_1 \cap X_2 = \emptyset$.

Definition 4 (Satisfaction of a Term) We say that a userset X *satisfies* a term ϕ under a configuration UR, if there exists $X_0 \subseteq X$ such that X_0 strictly satisfies ϕ under UR. We often say X satisfies (or strictly satisfies) ϕ (and omit "under UR"), when UR is obvious from the context or identifying UR is not important.

It follows from the above definition that term satisfaction is *monotonic* in the sense that if X satisfies ϕ under UR, so does any superset of X. We choose to define term satisfaction this way because our intent is that a term in the algebra represents the minimum qualifications a set of users together must possess. For example, if a policy requires 2 different users to together perform a task in order to prevent fraud, then having 3 different users to perform the task certainly satisfies the policy.

Term satisfaction is monotonic in another sense as well: If X satisfies ϕ under UR_1, then for any configuration UR_2 such that $UR_1 \subseteq UR_2$, X also satisfies ϕ under UR_2. This results from the fact that we do not have a negation operator in the algebra.

In Appendix A.1, we give an alternative definition of term satisfaction, which appears simpler and more elegant than Definitions 3 and 4. There we also discuss the deficiencies of the alternative definition that lead us to choose the current definition over it.

We now give several additional examples of terms, which help illustrate the expressive power of the algebra.

- $(($`Manager` $\odot$ `Accountant`$)$ $\otimes$ `Treasurer`$)$

  This term requires a `Manager`, an `Accountant`, and a `Treasurer`. A single user that is both a `Manager` and an `Accountant` can satisfy (`Manager` $\odot$ `Accountant`); however, the user that satisfies the subterm `Treasurer` must be different from the users (or user) that satisfy (`Manager` $\odot$ `Accountant`).

- $(($`Physician` $\sqcup$ `Nurse`$)$ $\otimes$ $($`Manager` $\sqcap$ `Accountant`$))$

  This term requires two different users, one of which is either a `Physician` or a `Nurse`, and the other is both a `Manager` and a `Treasurer`.

- $(((\texttt{Manager} \otimes \texttt{Manager}) \sqcup (\texttt{Manager} \otimes \texttt{Supervisor}) \sqcup (\texttt{Supervisor} \otimes \texttt{Supervisor} \otimes \texttt{Supervisor})) \odot (\texttt{Clerk} \otimes \texttt{Clerk}))$

  This example is taken from [20]. For a check to be prepared and issued, two different clerks need to be involved. In addition, a weight of 3 is required for approval, a `Manager` has weight 2, and a `Supervisor` has weight 1. In [20], the users who are involved all need to be different, which would require at least 4 different users to be involved. Here, we relax the policy to allow a user who is involved in the approval to prepare and issue checks; this provides more flexibility. The policy without relaxation can be expressed by replacing the $\odot$ operator in the term with $\otimes$.

Some properties about term satisfaction are given in the theorem below.

Theorem 1 The following properties about term satisfaction hold.

1. X satisfies $(\phi_1 \sqcup \phi_2)$ if and only if either X satisfies ϕ_1 or X satisfies ϕ_2.
2. X satisfies $(\phi_1 \sqcap \phi_2)$ implies that X satisfies both ϕ_1 and ϕ_2, but X satisfies both ϕ_1 and ϕ_2 does not imply that X satisfies $(\phi_1 \sqcap \phi_2)$.
3. X satisfies $(\phi_1 \odot \phi_2)$ if and only if X satisfies both ϕ_1 and ϕ_2.
4. X satisfies $(\phi_1 \otimes \phi_2)$ if and only if X can be partitioned into two disjoint subsets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. This also implies that X satisfies both ϕ_1 and ϕ_2.

Proof. Proof of (1): If X satisfies $(\phi_1 \sqcup \phi_2)$, then X has a subset X_0 that strictly satisfies $(\phi_1 \sqcup \phi_2)$. It follows that X_0 strictly satisfies either ϕ_1 or ϕ_2, implying that either X satisfies ϕ_1 or X satisfies ϕ_2. For the other direction: without loss of generality, assume that X satisfies ϕ_1, then X has a subset X_0 that strictly satisfies ϕ_1; therefore, X_0 strictly satisfies $(\phi_1 \sqcup \phi_2)$ and thus X satisfies $(\phi_1 \sqcup \phi_2)$.

Proof of (2): If X satisfies $(\phi_1 \sqcap \phi_2)$, then X has a subset X_0 that strictly satisfies $(\phi_1 \sqcap \phi_2)$, and thus X_0 strictly satisfies both ϕ_1 and ϕ_2. It follows that X satisfies both ϕ_1 and ϕ_2. For the other direction, consider the following example: u_1 is a member of r_1 but not r_2, and u_2 is a member of r_2 but not r_1, then $\{u_1, u_2\}$ satisfies both r_1 and r_2, but $\{u_1, u_2\}$ does not satisfy $(r_1 \sqcap r_2)$.

Proof of (3): If X satisfies $(\phi_1 \odot \phi_2)$, then X has a subset X_0, which in turn has two subsets X_1 and X_2 such that X_1 strictly satisfies ϕ_1 and X_2 strictly satisfies ϕ_2. It follows that X satisfies both ϕ_1 and ϕ_2. For other direction: if X satisfies both ϕ_1 and ϕ_2, then X has subsets X_1 and X_2 such that X_1 strictly satisfies ϕ_1 and X_2 strictly satisfies ϕ_2. Then the set $X_1 \cup X_2$ is a subset of X and strictly satisfies $(\phi_1 \odot \phi_2)$.

Proof of (4): Observe that X satisfies $(\phi_1 \otimes \phi_2)$ if and only if X can partitioned into three *disjoint* sets $X_0 \cup X_1 \cup X_2 = X$ such that X_1 strictly satisfies ϕ_1 and X_2 strictly satisfies ϕ_1. The properties in (4) follow directly. ∎

Given a configuration UR, we use $V_{UP}(\phi)$ to denote the set of all usersets that strictly satisfy ϕ under UP, and $S_{UP}(\phi)$ to denote the set of all usersets that satisfy ϕ under UP. We sometimes omit the subscript UP when doing so does not cause confusion. From Definitions 3, 4 and Theorem 1. The following equations hold for any given configuration UR.

$$\begin{array}{rclcrcl} V(\phi_1 \sqcup \phi_2) & = & V(\phi_1) \cup V(\phi_2) & \quad & V(\phi_1 \sqcap \phi_2) & = & V(\phi_1) \cap V(\phi_2) \\ S(\phi_1 \sqcup \phi_2) & = & S(\phi_1) \cup S(\phi_2) & & S(\phi_1 \odot \phi_2) & = & S(\phi_1) \cap S(\phi_2) \\ S(\phi_1 \sqcap \phi_2) & \subseteq & S(\phi_1) \cap S(\phi_2) & & S(\phi_1 \otimes \phi_2) & \subseteq & S(\phi_1) \cap S(\phi_2) \end{array}$$

We now introduce the notion equivalence among terms. This 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 UR, X strictly satisfies ϕ_1 under UR if and only if X strictly satisfies ϕ_2 under UR. That is, $\phi_1 \equiv \phi_2$ if and only if $\forall\, UR\, [V_{UR}(\phi_1) = V_{UR}(\phi_2)]$.

The reason we choose to use strict satisfaction rather than satisfaction in the above definition is explained in the next section. The following theorem states properties of term equivalence.

Theorem 2 If $\phi_1 \equiv \phi_2$, then the following hold:

1. For every userset X and every configuration UR, X satisfies ϕ_1 under UR if and only if X satisfies ϕ_2 under UR.
2. 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 $\phi \equiv \phi'$.

Property 1 follows directly from Definitions 4 and 5. Property 2 can be proved by a straightforward induction on the structure ϕ.

Theorem 3 The operators have the following algebraic properties:

1. The operators $\sqcup, \sqcap, \odot, \otimes$ are commutative and associative. That is, for each $\mathsf{op} \in \{\sqcup, \sqcap, \odot, \otimes\}$, and any terms ϕ_1, ϕ_2, and ϕ_3, we have $(\phi_1 \mathsf{op} \phi_2) \equiv (\phi_2 \mathsf{op} \phi_1)$ and $((\phi_1 \mathsf{op} \phi_2) \mathsf{op} \phi_3) \equiv (\phi_1 \mathsf{op} (\phi_2 \mathsf{op} \phi_3))$.
2. The operators $\sqcup$ and $\sqcap$ distribute over each other. That is, $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3)) \equiv ((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$ and $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$.
3. The operator $\odot$ distributes over $\sqcup$. That is, $(\phi_1 \odot (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$.
4. The operator $\otimes$ distributes over $\sqcup$. That is, $(\phi_1 \otimes (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$.
5. No other ordered pair of operators have the distributive property.

The proof for the above theorem is in Appendix A.2. For each case that the distributive property does not hold, we give a counter example.

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. Sometimes we also omit the outermost parentheses of a term.

3 The Term Satisfiability Problem

Given the definitions of terms and term satisfaction, a natural question that arises is whether every term in the algebra is satisfiable. That is, whether for every term ϕ, there exists a set X of users and a configuration UR such that X satisfies ϕ under UR. The answer is "no"; there exist terms that are not satisfiable. 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 strictly satisfiable only by a singleton userset, and $(r_2 \otimes r_3)$ is strictly satisfiable only by a userset of cardinality 2. Therefore, there does not exist any userset that strictly satisfies ϕ, and therefore no userset satisfies ϕ. Consequently, in this section we study the term satisfiability problem, which asks for a given term ϕ, whether there exists a userset X and a configuration UR such that X satisfies ϕ under UR.

We now explain why we do not define two terms to be equivalent when they are satisfied by exactly the same usersets under every configuration. Consider the two terms r_1 and $(r_1 \odot r_1)$. They are both satisfied by any userset that contains at least one user who is a member r_1; however, they differ when we consider strict satisfaction. The term r_1 can never be strictly satisfied by $\{u_1, u_2\}$. On the other hand, when both u_1 and u_2 are members of the role r_1, the userset $\{u_1, u_2\}$ strictly satisfies $(r_1 \odot r_1)$ because $\{u_1, u_2\} = \{u_1\} \cup \{u_2\}$, $\{u_1\}$ strictly satisfies r_1, and $\{u_2\}$ strictly satisfies r_1. (In this case, the three usersets $\{u_1\}$, $\{u_2\}$, and $\{u_1, u_2\}$ all strictly satisfies $(r_1 \odot r_1)$.) This difference between r_1 and $(r_1 \odot r_1)$ is clearly shown in the following example. In the term $\phi = (r_1 \sqcap (r_2 \otimes r_3))$, if we replace r_1 with $(r_1 \odot r_1)$, the resulting term $\phi' = ((r_1 \odot r_1) \sqcap (r_2 \otimes r_3))$ is quite different from ϕ. The term ϕ is not satisfiable, as we discuss in the beginning of this section; yet the term ϕ' is satisfiable.

With respect to the term satisfiability problem, we observe that, by definition, a term is satisfiable if and only if it is strictly satisfiable. Furthermore, in order to strictly satisfy a term, a userset must be of certain size. For example, $(r_1 \odot (r_2 \otimes r_3))$ can be strictly 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 term. This notion turns out to be useful in determining whether a term is satisfiable or not.

Definition 6 (Characteristic Numbers) Given a term ϕ and a positive integer k, we say that k is a *characteristic number* of a term ϕ when there exists a userset of size k that strictly 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.

We point out that for any term ϕ, it is satisfiable if and only if $C(\phi) \neq \emptyset$.

Theorem 4 The characteristic set of a term can be computed as follows:

- $C(\mathsf{All}) = C(r) = \{1\}$, where r is a role
- $C(\phi_1 \sqcup \phi_2) = C(\phi_1) \cup C(\phi_2)$
- $C(\phi_1 \sqcap \phi_2) = C(\phi_1) \cap C(\phi_2)$
- $C(\phi_1 \odot \phi_2) = \{\, k \mid \max[\min(C(\phi_1)),\ \min(C(\phi_2))] \leq k \ \wedge\ k \leq [\max(C(\phi_1)) + \max(C(\phi_2))] \,\}$
- $C(\phi_1 \otimes \phi_2) = \{\, c_1 + c_2 \mid c_1 \in C(\phi_1) \ \wedge\ c_2 \in C(\phi_2) \,\}$

The proof for the theorem is in Appendix B. We now illustrate each case in the theorem using the examples from Section 2.

- $C(\mathsf{All} \otimes \mathsf{All} \otimes \mathsf{All}) = \{3\}$
- $C(\texttt{Manager} \sqcap \texttt{Accountant}) = C(\texttt{Physician} \sqcup \texttt{Nurse}) = \{1\}$
- $C(\texttt{Manager} \odot \texttt{Accountant}) = \{1, 2\}$

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

  Either 1 or 2 users suffice for (`Manager` $\odot$ `Accountant`), and an additional user is needed to satisfy `Treasurer`.

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

  Only one user is needed for both $(\texttt{Manager} \sqcup \texttt{Accountant})$ and $(\texttt{Manager} \sqcap \texttt{Treasurer})$, and the $\otimes$ mandates that these users must be different from one another.

- $C(((\texttt{Manager} \otimes \texttt{Manager}) \sqcup (\texttt{Manager} \otimes \texttt{Supervisor}) \sqcup (\texttt{Supervisor} \otimes \texttt{Supervisor} \otimes \texttt{Supervisor})) \odot (\texttt{Clerk} \otimes \texttt{Clerk})) = \{2, 3, 4, 5\}$

  Let ϕ_1 denote the subterm before $\odot$, then $C(\phi_1) = \{2, 3\}$. $C(\texttt{Clerk} \otimes \texttt{Clerk}) = \{2\}$. Using the rule in Theorem 4 to combine them, we get $\{2, 3, 4, 5\}$. This term can be strictly satisfied by 2 users that are members of both the `Manager` role and the `Clerk` role. This term can also be strictly satisfied by 5 users that include 3 supervisors and 2 clerks.

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

We point out that the characteristic set of a term can be used for another important purpose, that is, to determine whether the term 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 now show that $C(\phi)$ can be computed in time quadratic in the size of ϕ.

Definition 7 We define the size of a term ϕ, denoted by $|\phi|$, to be the number 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 operators in ϕ plus 1.

Lemma 5 Every characteristic number k of ϕ satisfies $k \leq |\phi|$.

The proof for the above lemma is straightforward by induction on the structure of terms. If follows from Lemma 5 that the cardinality of $C(\phi)$ is no more than $|\phi|$. A straightforward algorithm to compute $C(\phi)$ is to follow Theorem 4. Given $C(\phi_1)$ and $C(\phi_2)$, calculating $C(\phi_1 \;\mathsf{op}\; \phi_2)$ takes time at most linear in $|C(\phi_1)| + |C(\phi_2)|$. 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 $O(|\phi|^2)$.

4 The Userset-Term Satisfaction (UTS) Problem

In Section 3, we have shown that the problem of determining whether a term ϕ is satisfiable at all takes time $O(|\phi|^2)$. In this section, we consider the following problem.

Definition 8 (The Userset-Term Satisfaction (UTS) problem) Given a configuration UR, a user set X, and a term ϕ, the problem of determining whether X satisfies ϕ under UR is called the Userset-Term Satisfaction (UTS) problem.

4.1 Computational Complexity of UTS

We show that UTS in the most general case (i.e., arbitrary terms in which all four 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 four operators $\{\sqcap, \sqcup, \odot, \otimes\}$ is allowed. For example, UTS$\langle\sqcup, \sqcap, \odot\rangle$ denotes the sub-case of UTS where ϕ does not contain operator $\otimes$, while UTS$\langle\otimes\rangle$ denotes the sub-case of UTS where $\otimes$ is the only kind of operator in ϕ. UTS$\langle\sqcup, \sqcap, \odot, \otimes\rangle$ denotes the general case.

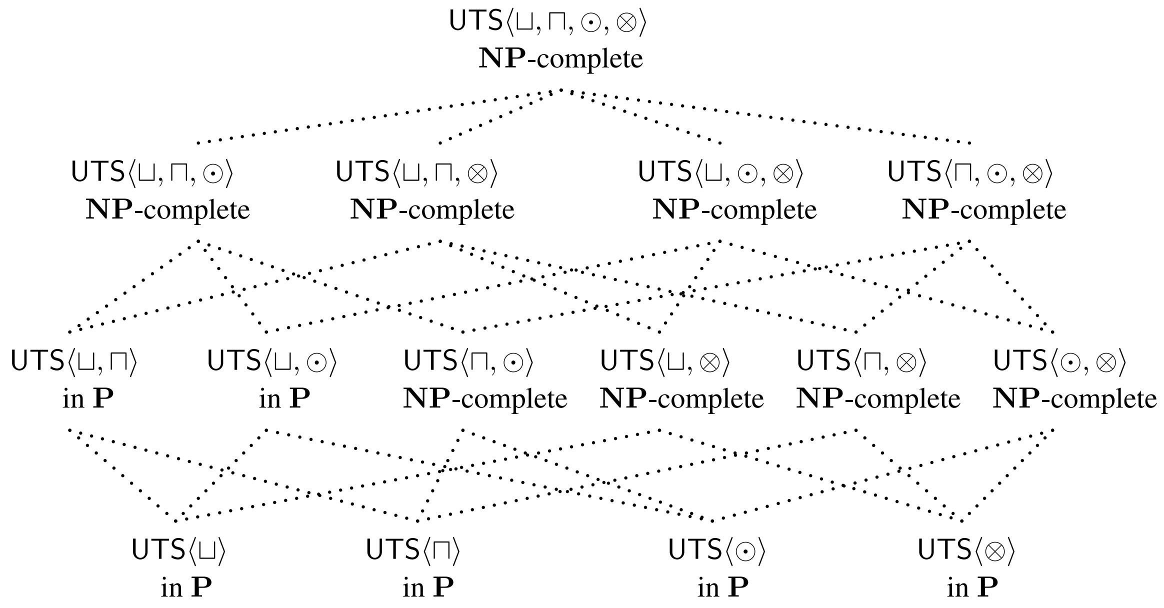

Figure 1: Various sub-cases of the Userset Term Satisfaction (UTS) problem and the corresponding time-complexity.

Theorem 6 The computational complexities for UTS and its subcases are given in Figure 1.

From Figure 1, we observe that when only the $\otimes$ operator is allowed, UTS can be efficiently solved; however, when $\otimes$ is combined with any of $\sqcap$, $\sqcup$, and $\odot$, UTS becomes $\mathbf{NP}$-complete. Also, the combination $\langle \sqcap, \odot \rangle$ is $\mathbf{NP}$-complete; whereas the combinations $\langle \sqcup, \sqcap \rangle$ and $\langle \sqcup, \odot \rangle$ are efficiently decidable. Therefore, one can order the difficulty of these operators as $\otimes > \sqcap, \odot > \sqcup$. The most difficult operator, $\otimes$, when combined with any other operator, results in intractable cases. The two operators in the middle, when combined with $\sqcup$, result in a tractable case; however, their combination is intractable. We point out that both $\sqcap$ and $\odot$ have the flavor of set intersection. Recall that we have $V(\phi_1 \sqcap \phi_2) = V(\phi_1) \cap V(\phi_2)$ and $S(\phi_1 \odot \phi_2) = S(\phi_1) \cap S(\phi_2)$, where $V(\phi)$ denotes the set of usersets that strictly satisfy ϕ and $S(\phi)$ denotes the set of usersets that satisfy ϕ.

We now prove the $\mathbf{NP}$-completeness results in Figure 1. It suffices to prove that the general case UTS$\langle \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP}$ and that the four cases UTS$\langle \sqcap, \odot \rangle$, UTS$\langle \sqcup, \otimes \rangle$, UTS$\langle \sqcap, \otimes \rangle$, and UTS$\langle \odot, \otimes \rangle$ are $\mathbf{NP}$-hard. All $\mathbf{NP}$-completeness results follow because of the obvious hardness relationships among these cases. Below we state lemmas that establish these results. The proofs for these lemmas are in Appendix C. For each $\mathbf{NP}$-hardness result, we discuss the terms used in the reductions. These terms illustrate what the hard cases are for the UTS problem. The observations we gain from these terms help us in Section 4.2, where we identify a wide class of syntactically restricted terms for which the UTS problem is tractable.

Lemma 7 UTS $\langle \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP}$.

The proof (in Appendix C) uses the observation that if one can guess which subset X_0 of the given userset X strictly satisfies the term and how X_0 is divided to strictly satisfy each component of the term, then the verification takes only polynomial time.

Lemma 8 UTS $\langle \sqcap, \odot \rangle$ is $\mathbf{NP}$-hard.

The proof (in Appendix C) uses a reduction from the $\mathbf{NP}$-complete SET COVERING problem [8]. The terms used in the reduction have the form $((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i))$, where $(\bigodot_k \phi)$ denotes k copies of ϕ connected together by $\odot$ and $(\bigodot_{i=1}^{n} r_i)$ denotes $(r_1 \odot \cdots \odot r_n)$. This illustrates that if we allow $\odot$ to appear within the scope of $\sqcap$, then the UTS problem may be intractable.

Lemma 9 UTS $\langle \odot, \otimes \rangle$ is $\mathbf{NP}$-hard.

The proof (in Appendix C) uses a reduction from the $\mathbf{NP}$-complete DOMATIC NUMBER problem [8]. The terms used in the reduction have the form $(\bigotimes_k (\bigodot_{i=1}^{n} r_i))$. This illustrates that if we allow $\odot$ to appear within the scope of $\otimes$, then the UTS problem may be intractable.

Lemma 10 UTS $\langle \sqcup, \otimes \rangle$ is $\mathbf{NP}$-hard.

The proof (in Appendix C) uses a reduction from the $\mathbf{NP}$-complete SET PACKING problem [8]. The terms used in the reduction have the form $(\bigotimes_k (\bigsqcup_{i=1}^{m} (\bigotimes R_i)))$, where R_i is a set of roles, and $(\bigotimes R_i)$ denotes the roles in R_i connected by the $\otimes$ operator. This illustrates that if we allow $\otimes$ to occur within the scope of $\sqcup$ and allow these occurrences of $\sqcup$ to occur within the scope of $\otimes$, then the UTS problem may be intractable.

Lemma 11 UTS $\langle \sqcap, \otimes \rangle$ is $\mathbf{NP}$-hard.

The proof (in Appendix C) uses a reduction from the $\mathbf{NP}$-complete SET COVERING problem [8]. The terms used in the reduction have the form $\bigsqcap_{i=1}^{n} \left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right)$. This illustrates that if we allow $\otimes$ to occur within the scope of $\sqcap$, then the UTS problem may be intractable.

Three Tractable Cases To prove all the $\mathbf{P}$ results in Figure 1, it suffices to prove that the three cases UTS$\langle \sqcap, \sqcup \rangle$, UTS$\langle \sqcup, \odot \rangle$, and UTS$\langle \otimes \rangle$ are in $\mathbf{P}$.

Lemma 12 UTS $\langle \sqcup, \sqcap \rangle$ can be solved in time $O(|\phi| \cdot |X|)$.

Proof. A term ϕ using only $\sqcup$ and $\sqcap$ has 1 as its only characteristic number. Therefore, a userset X satisfies ϕ if and only if there exists a user in X that strictly satisfies ϕ. A straightforward algorithm is for each user $u \in X$, check whether $\{u\}$ strictly satisfies ϕ. By definition, $\{u\}$ strictly satisfies $(\phi_1 \sqcup \phi_2)$ if and only if either $\{u\}$ strictly satisfies ϕ_1 or $\{u\}$ strictly satisfies ϕ_2, and $\{u\}$ strictly satisfies $(\phi_1 \sqcap \phi_2)$ if and only if $\{u\}$ strictly satisfies both ϕ_1 and ϕ_2. Assuming that the configuration UR is stored in a data structure such that checking whether $(u, r) \in UR$ takes constant time, one can check whether $\{u\}$ strictly satisfies ϕ in time linear in $|\phi|$ by following the structure of ϕ. ∎

Lemma 13 UTS $\langle \sqcup, \odot \rangle$ can be solved in time $O(|\phi| \cdot |X|)$.

Proof. By Theorem 1, a userset X satisfies $(\phi_1 \sqcup \phi_2)$ if and only if either X satisfies ϕ_1 or X satisfies ϕ_2, and a userset X satisfies $(\phi_1 \odot \phi_2)$ if and only if X satisfies both ϕ_1 and ϕ_2. Therefore, one can determine whether X satisfies a term ϕ that uses only the operators $\sqcup$ and $\odot$ by following the structure of the term. To determine whether X satisfies a role r takes time linear in $|X|$, assuming that one can determine whether $(u, r) \in UR$ in constant time. ∎

Lemma 14 UTS $\langle \otimes \rangle$ can be solved in time $O((|\phi| + |X|) \cdot |\phi| \cdot |X|)$.

Proof. Given a term ϕ that uses only the operator $\otimes$, we show that determining whether a userset X satisfies ϕ under a configuration UR can be reduced to the maximum matching problem on bipartite graphs, which can be solved in $O(MN)$ time, where M is the number of edges and N is the number of nodes in G [16].

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

We now show that X satisfies ϕ if and only if the maximal matching in the graph G has size s. If the maximal matching has size s, then each node in V_1 matches to a certain node in V_2, which means that the s atomic terms in ϕ are strictly satisfied by s distinct users in X; thus X satisfies ϕ. If X satisfies ϕ, by definition, X contains s users such that each user strictly satisfies an atomic term in ϕ. From our construction of G, a maximal matching of size s exists. ∎

4.2 UTS is Tractable for Canonical Terms

From the computational complexity results in Figure 1, one can see that when only $\otimes$ is allowed, the UTS problem can be solved in polynomial time. However, when $\otimes$ is combined with any of the three operators $\sqcup$, $\sqcap$, and $\odot$, the UTS problem becomes $\mathbf{NP}$-hard. We now show that if a term satisfies certain syntactic restrictions, then even if all four operators appear in the term, one can still efficiently determine whether a userset satisfies the term. To characterize the syntactic restrictions, we recall the observations about the hard instances used in the $\mathbf{NP}$-hardness proofs:

1. It $\odot$ appears within the scope of $\sqcap$, then UTS may be intractable.

2. If $\odot$ appears within the scope of $\otimes$, then UTS may be intractable.

3. If $\sqcup$ appears in between the scopes of two $\otimes$, e.g., in terms such as $(\phi_1 \otimes (\phi_2(\sqcup(\phi_3 \otimes \phi_4))))$, then UTS may be intractable.

4. If $\otimes$ appears within the scope of $\sqcap$, then the UTS problem may be intractable.

From (2) and (4), we observe that in order to avoid the intractable cases, $\odot$ should be used outside the scope of $\otimes$, which in turn should be used outside the scope of $\sqcap$. In other words, the order from outmost to innermost is $\odot, \otimes, \sqcap$. This order also avoids intractable cases in (1). From (3), we observe that $\sqcup$ should not be arbitrarily mixed with $\otimes$. However, $\sqcup$ can be mixed with $\odot$ and with $\sqcap$. These observations lead us to the following definition.

Definition 9 (Canonical Forms for Terms) The canonical forms for terms are defined as follows:

- A term is *in level-1 canonical form* if it uses only operators from the set $\{\sqcup, \sqcap\}$. An atomic term is in level-1 canonical form, because no operator is used.

- A term is *in level-2 canonical form* if it is of the form $(\phi_1 \otimes \cdots \otimes \phi_n)$, where $n \geq 1$ and for each i such that $1 \leq i \leq n$, the subterm ϕ_i is in level-1 canonical form.

- A term is *in level-3 canonical form* if it consists of subterms that are in level-2 canonical forms, and these subterms are connected only by operators in the set $\{\sqcup, \odot\}$.

We say that a term is *in canonical form* if it is in level-3 canonical form. Observe that any term that is in level-1 canonical form is also in level-2 canonical form, and any term that is in level-2 canonical form is also in level-3 canonical form.

We point out that a term in canonical form is always satisfiable. A term in level-1 canonical form has characteristic set $\{1\}$. And a term in canonical form combines subterms in level-1 canonical form using the three operators $\otimes, \odot, \sqcup$. Because of the rules by which one calculates the characteristic sets of $\phi_1 \otimes \phi_2$, $\phi_1 \odot \phi_2$, and $\phi_1 \sqcup \phi_2$ (See Theorem 4), one can never derive an empty characteristic set for a term in canonical form. We believe that terms in the canonical forms are general enough to specify many high-level security policies in practice. All except one example of terms in this paper are in canonical form.

While canonical forms are defined to avoid the intractable cases we have identified in the $\mathbf{NP}$-hardness proofs, the following theorem shows that UTS is tractable for all terms that are in canonical form. This shows that the intractable cases that we have identified are exhaustive in a sense, and the canonical forms capture the most general tractable cases.

Theorem 15 Given a term ϕ in canonical form, a set X of users, and a configuration UR, checking whether X satisfies ϕ under UR can be done in cubic time.

Proof. The algorithm for deciding whether X satisfies a term ϕ in canonical form combines the ideas in the algorithms for solving $\mathsf{UTS}\langle \sqcup, \sqcap \rangle$ (Lemma 12), $\mathsf{UTS}\langle \sqcup, \odot \rangle$ (Lemma 13), and $\mathsf{UTS}\langle \otimes \rangle$ (Lemma 14).

First observe that by Theorem 1, a userset X satisfies $(\phi_1 \sqcup \phi_2)$ if and only if either X satisfies ϕ_1 or X satisfies ϕ_2, and a userset X satisfies $(\phi_1 \odot \phi_2)$ if and only if X satisfies both ϕ_1 and ϕ_2. Therefore, as long as one can decide whether X satisfies each subterm of ϕ in level-2 canonical form, one can easily combine the results to determine whether X satisfies ϕ.

Second, observe that if a term is in the level-2 canonical form, it has the form $(\phi_1 \otimes \cdots \otimes \phi_s)$, where each ϕ_i uses only operators in $\{\sqcup, \sqcap\}$. Such a ϕ_i can be strictly satisfied only by a singleton userset. Furthermore, it takes time linear in $|\phi_i|$ to determine whether any singleton userset $\{u\}$ satisfies ϕ_i or not. Therefore, to determine whether X satisfies $(\phi_1 \otimes \cdots \otimes \phi_s)$, one can first determine whether each user in X satisfies each ϕ_i for $1 \leq i \leq s$, and then use the maximal matching problem for bipartite graphs to determine whether X satisfies $(\phi_1 \otimes \cdots \otimes \phi_s)$ or not, using the same idea as in the proof of Lemma 14.

This can be performed in time cubic in the size of the instance. ∎

5 The Static Safety Checking (SSC) Problem

A high-level policy associates a task with a term ϕ and requires that every set of users who complete an instance of the task satisfy a term ϕ. For example, one may associate a task of buying and paying for goods with the term $(\mathtt{Employee} \otimes \mathtt{Supervisor}) \odot (\mathtt{Accountant} \otimes \mathtt{Accountant})$. Such a policy can be enforced either statically or dynamically. In this paper, we look at static enforcement. Dynamic enforcement is interesting future work and is beyond the scope of this paper.

In static enforcement, one first identifies all permissions that are needed to perform the task. Assume that the following four permissions are involved in the task of buying and paying for goods: p_{order}, $p_{approve}$, p_{goods}, and $p_{payment}$. One then specifies a static safety policy that requires that every set of users who together have all four permissions satisfy the term (`Employee` $\otimes$ `Supervisor`) $\odot$ (`Accountant` $\otimes$ `Accountant`). In this section we look at the problem of determining whether a configuration satisfies such a static safety policy.

We first extend a configuration from UR to $\langle UR, UP \rangle$, where $UR \subseteq \mathcal{U} \times \mathcal{R}$ and $UP \subseteq \mathcal{U} \times \mathcal{P}$, where $\mathcal{U}$ is the set of all users, $\mathcal{R}$ is the set of all roles, and $\mathcal{P}$ is the set of all permissions. If $(u, p) \in UP$, then we say that the user u has the permission p. Note that by assuming that a configuration contains the binary relation $UP \subseteq \mathcal{U} \times \mathcal{P}$, we are not assuming permissions are directly assigned to users; rather, we assume only that one can calculate the relation UP from the access control state. Static safety policies are formally defined below.

Definition 10 (Static Safety Policy) A *static safety policy* is specified as

$$\mathsf{sp}\langle P, \phi \rangle$$

where $P = \{p_1, \cdots, p_n\}$ is a set of permissions and ϕ is a term in the Algebra.

A configuration $\langle UR, UP \rangle$ is safe with respect to a safety policy $\mathsf{sp}\langle P, \phi \rangle$ if any set of users who together possess all permissions in P satisfies ϕ.

The permissions in a safety policy are the permissions needed to carry out a sensitive task, and the policy guarantees that a set of users can successfully execute the task only if these users together meet a certain safety requirement. The following problem naturally arises.

Definition 11 (The Static Safety Checking (SSC) problem) Determining whether a configuration $\langle UR, UP \rangle$ is safe with respect to a static safety policy $\mathsf{sp}\langle P, \phi \rangle$ is called the *Static Safety Checking (*SSC*) problem.*

Given a policy $\mathsf{sp}\langle P, \phi \rangle$ and a configuration $\langle UR, UP \rangle$, there may be multiple sets of users who together have all permissions in P. For each such set, we need to check whether the set satisfies ϕ under UR. Therefore, the SSC problem is at least as hard as the UTS problem. We now give the computational complexities of SSC in the general case as well as SSC in sub-algebras where only a subset of the operators is allowed.

Theorem 16 The computational complexities of SSC and its subcases are given in Figure 2.

Comparing the complexities of SSC in Figure 2 with the complexities of UTS in Figure 1, we observe that all cases that are $\mathbf{NP}$-complete in Figure 1 become $\mathbf{coNP}$-hard in Figure 2. We also observe that the two cases ($\langle \sqcup, \odot \rangle$ and $\langle \otimes \rangle$) that are in $\mathbf{P}$ in Figure 1 become $\mathbf{coNP}$-complete in Figure 2. However, the other four cases ($\langle \sqcup, \sqcap \rangle$, $\langle \sqcup \rangle$, $\langle \sqcap \rangle$, and $\langle \odot \rangle$) that are in $\mathbf{P}$ in Figure 1 remain in $\mathbf{P}$ in Figure 2.

Lemma 17 $\mathsf{SSC}\langle \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{coNP^{NP}}$.

Proof. $\mathsf{SSC}\langle \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{coNP^{NP}}$ means that the complement of SSC $\langle \sqcup, \sqcap, \odot, \otimes \rangle$ can be solved by a nondeterministic Oracle Turing Machine that has oracle access to a machine that can answer any $\mathbf{NP}$ queries. (See Appendix D for a brief overview of Oracle Turing Machines.) If a configuration $\langle UR, UP \rangle$ does not satisfy a static safety policy $\mathsf{sp}\langle P, \phi \rangle$, then there exists an evidence which is a set X of users such that the users in X together have all the permissions in P and X does not satisfy ϕ. Verifying that X does not satisfy ϕ can be performed by an oracle query. ∎

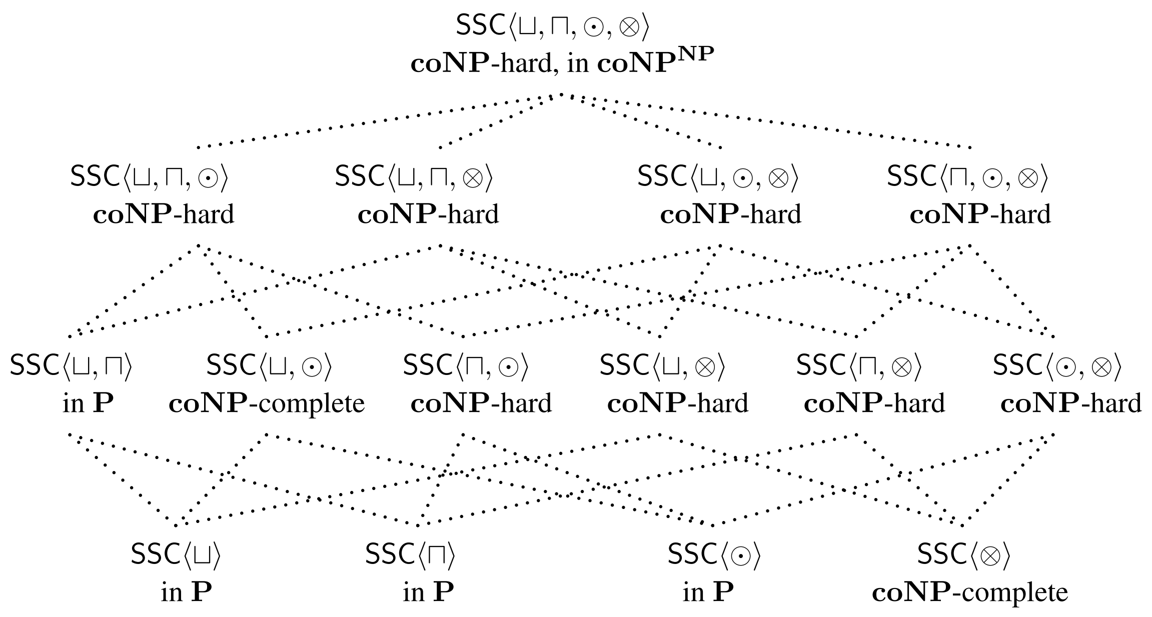
$\mathsf{SSC}\langle\sqcup,\sqcap,\odot,\otimes\rangle$
coNP-hard, in $\mathbf{coNP^{NP}}$
$\mathsf{SSC}\langle\sqcup,\sqcap,\odot\rangle$
coNP-hard
$\mathsf{SSC}\langle\sqcup,\sqcap,\otimes\rangle$
coNP-hard
$\mathsf{SSC}\langle\sqcup,\odot,\otimes\rangle$
coNP-hard
$\mathsf{SSC}\langle\sqcap,\odot,\otimes\rangle$
coNP-hard
$\mathsf{SSC}\langle\sqcup,\sqcap\rangle$
in **P**
$\mathsf{SSC}\langle\sqcup,\odot\rangle$
coNP-complete
$\mathsf{SSC}\langle\sqcap,\odot\rangle$
coNP-hard
$\mathsf{SSC}\langle\sqcup,\otimes\rangle$
coNP-hard
$\mathsf{SSC}\langle\sqcap,\otimes\rangle$
coNP-hard
$\mathsf{SSC}\langle\odot,\otimes\rangle$
coNP-hard
$\mathsf{SSC}\langle\sqcup\rangle$
in **P**
$\mathsf{SSC}\langle\sqcap\rangle$
in **P**
$\mathsf{SSC}\langle\odot\rangle$
in **P**
$\mathsf{SSC}\langle\otimes\rangle$
coNP-complete

Figure 2: Various sub-cases of the SSC problem and the corresponding time-complexity. As the most general case of the problem, $\mathsf{SSC}\langle\sqcup,\sqcap,\odot,\otimes\rangle$, is in $\mathbf{coNP^{NP}}$, all subcases are also in $\mathbf{coNP^{NP}}$.

We have shown that $\mathsf{SSC}\langle\sqcup,\sqcap,\odot,\otimes\rangle$ and a number of its subcases are **coNP**-hard and are in $\mathbf{coNP^{NP}}$. It remains open whether these problems are $\mathbf{coNP^{NP}}$-complete or not. Readers who are familiar with computational complexity theory will recognize that $\mathbf{coNP^{NP}}$ is a complexity class in the Polynomial Hierarchy. (See Appendix D for a brief introduction to the Polynomial Hierarchy.)

Lemma 18 If UTS in a sub-algebra is in **P**, then SSC in the same sub-algebra is in **coNP**.

Proof. A configuration is *not safe* with respect to a policy $\mathsf{sp}\langle P,\phi\rangle$ if and only if there exists a counter evidence userset X such that X covers all permissions in P and X does not satisfy ϕ. Given that UTS is in **P**, verifying whether X is a counter evidence of safety can be done in polynomial time. Therefore, the complement of SSC is in **NP**. ∎

Lemma 19 $\mathsf{SSC}\langle\otimes\rangle$ is **coNP**-complete.

The fact that the problem is in **coNP** follows from Lemma 18. The proof (in Appendix E) that this is **coNP**-hard is by a reduction from the SET COVERING problem to the complement of SSC $\langle\otimes\rangle$.

Lemma 20 $\mathsf{SSC}\langle\sqcup,\odot\rangle$ is **coNP**-complete.

The fact that this is in **coNP** follows from Lemma 18. The proof (in Appendix E) that this is **coNP**-hard is by a reduction from the PROPOSITIONAL VALIDITY problem [8].

Lemma 21 $\mathsf{SSC}\langle\odot\rangle$ is in **P**.

An algorithm is given in the proof (in Appendix E). It takes time at most cubic in the size of the instance.

Lemma 22 $\mathsf{SSC}\langle\sqcap,\sqcup\rangle$ is in **P**.

An algorithm is given in the proof (in Appendix E). It takes time at most cubic in the size of the instance.

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 [6] identified SoD along with well-formed transactions as two major mechanisms of fraud and error control. Nash and Poland [15] 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, 7, 9, 10, 11, 22, 23] on constraints in the context of RBAC. They either proposed and classified new kinds of constraints [9, 22] or proposed new languages for specifying sophisticated constraints [1, 2, 7, 11, 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.

Li et al. [12] studied the problem of using static mutually exclusive role (SMER) constraints to enforce Static Separation of Duty (SSoD) policies, which require at least k different users to possess all permissions in a given set. The SSoD policies are special cases of the static safety policies in this paper. In these policies, the terms have the form $\mathsf{All} \otimes \cdots \otimes \mathsf{All}$. Li et al. [12] did not consider high-level policies other than SSoD policies.

McLean [14] 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 observed 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], Wijesekera and Jajodia [24], Pincus and Wing [17]. These algebras are quite different from ours, as each element in their algebra is a policy, which specifies what subjects are allowed to access which resources.

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. [13]. In [13], 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. One can write $A.R \longleftarrow B_1.R_1 \otimes \cdots \otimes B_k.R_k$, which means that

$$\mathit{members}(A.R) \supseteq \{\, s_1 \cup \cdots \cup s_k \mid s_i \in \mathit{members}(B_i.R_i) \ \wedge\ s_i \cap s_j = \emptyset \text{ for } 1 \leq i \neq j \leq k \,\}.$$

One can also write $A.R \longleftarrow B_1.R_1 \odot \cdots \odot B_k.R_k$, which means that

$$\textit{members}(A.R) \supseteq \{\ s_1 \cup \cdots \cup s_k \mid s_i \in \textit{members}(B_i.R_i) \text{ for } 1 \leq i \leq k\ \}.$$

This corresponds exactly to our notion of strict satisfaction for the two operators. The focus on [13] is on introducing the RT family of trust-management languages. Li et al. [13] did not propose to use these operators to specify high-level security policies, or to study the interactions of these two operators with $\sqcup$ and $\sqcap$. They also did not study the algebraic properties of these operators, the term satisfiability problem, the UTS problem, or the SSC problem.

7 Summary

While separation of duty policies are extremely important and widely used, they cannot capture qualification requirements. We have introduced a novel algebra that enables the specification of high-level policies that combine user qualification requirements with 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 four operators, and is expressive enough to specify many diverse policies. We have also studied several computational problems related to the algebra, including determining whether a term is satisfiable at all, determining whether a userset satisfies a term (UTS), and determining whether a configuration is safe with respect to a static safety policy (SSC). As our algebra is about the general concept of sets of sets, we conjecture that it will prove to be useful in other contexts as well.

References

[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] 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.

[7] 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.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] C. H. Papadimitrou and K. Steiglitz. *Combinatorial Optimization*. Prentice Hall, 1982.

[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.

A More Details on the algebra

A.1 An Alternative Definition of Term Satisfaction

We now discuss an alternative definition for term satisfaction in the algebra. This is the first definition we came up with. While this definition appears simpler and more elegant than Definitions 3 and 4, it turns out to have some undesirable properties.

Definition 12 (Alternative Definition of Term Satisfaction) We say that a set X of users satisfy a term ϕ if one of the following holds:

- The term ϕ is the keyword All, and X is nonempty.
- The term ϕ is a role r, and there exists a user $u \in X$ such that u is a member of the role r.
- The term ϕ is of the form $(\phi_1 \sqcup \phi_2)$, and either X satisfies ϕ_1 or X satisfies ϕ_2.
- The term ϕ is of the form $(\phi_1 \sqcap \phi_2)$, and X satisfies ϕ_1 as well as ϕ_2.
- The term ϕ is of the form $(\phi_1 \otimes \phi_2)$, and there exist 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 X_1 and X_2 such that $X_1 \cup X_2 = X$, X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2.

Using this definition, one can show that the operator $\sqcap$ is equivalent to the operator $\odot$ in the sense that for any X, ϕ_1 and ϕ_2, X satisfies $(\phi_1 \sqcap \phi_2)$ if and only if X satisfies $(\phi_1 \odot \phi_2)$. Thus one of the operators is redundant. More importantly, under this definition $\{u_1, u_2\}$ can satisfy $(r_1 \sqcap r_2)$ when neither u_1 nor u_2 is a member of both r_1 and r_2. This would happen when u_1 is a member of r_1 and u_2 is a member of r_2. Therefore, using this definition, one cannot express a policy that requires a user that is a member of both r_1 and r_2.

A.2 Proof for Theorem 3 on Algebraic Properties

1. The operators $\sqcup, \sqcap, \otimes, \odot$ are commutative and associative.

   This is straightforward from Definition 4 and Theorem 1.

2. The operators $\sqcup$ is distributive over $\sqcap$.

   If a userset X strictly satisfies $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3))$, then either X strictly satisfies ϕ_1, or X strictly satisfy both ϕ_2 and ϕ_3. It follows that X strictly satisfies $((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$.

If X strictly satisfies $((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$, then X strictly satisfies $(\phi_1 \sqcup \phi_2)$ and $(\phi_1 \sqcup \phi_3)$. There are only two cases: (1) X strictly satisfies ϕ_1; and (2) X strictly satisfies both ϕ_2 and ϕ_3. In either case, X strictly satisfies $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3))$.

The operators $\sqcap$ is distributive over $\sqcup$.

If X strictly satisfies $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3))$, then X strictly satisfies both ϕ_1 and $(\phi_2 \sqcup \phi_3)$. Then X strictly satisfies either ϕ_2 or ϕ_3. It follows that X strictly satisfies $((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$.

If X strictly satisfies $((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$, then either (1) X strictly satisfies $(\phi_1 \sqcap \phi_2)$ or (2) X strictly satisfies $(\phi_1 \sqcap \phi_3)$. Therefore X strictly satisfies ϕ_1; furthermore, X strictly satisfies either ϕ_2 or ϕ_3. It follows that X strictly satisfies $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3))$.

3. The operator $\odot$ is distributed over $\sqcup$.

   If X strictly satisfies $(\phi_1 \odot (\phi_2 \sqcup \phi_3))$, then there exists X_1, X_2 such that $X_1 \cup X_2 = X$, X_1 strictly satisfies ϕ_1, and X_2 strictly satisfies $(\phi_2 \sqcup \phi_3)$. By Definition 3, X_2 strictly satisfies ϕ_2 or strictly satisfies ϕ_3. In the former case, X strictly satisfies $(\phi_1 \odot \phi_2)$, which implies that X strictly satisfies $((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$, as desired. The argument is analogous if X_2 strictly satisfies ϕ_3 but not ϕ_2.

   If X strictly satisfies $((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$, then either X strictly satisfies $(\phi_1 \odot \phi_2)$ or X strictly satisfies $(\phi_1 \odot \phi_3)$. In the former case, by Definition 3, there exist X_1, X_2 such that $X_1 \cup X_2 = X$, X strictly satisfies ϕ_1 and X_2 strictly satisfies ϕ_2. Therefore, X_2 strictly satisfies $(\phi_2 \sqcup \phi_3)$, and consequently, X strictly satisfies $(\phi_1 \odot (\phi_2 \sqcup \phi_3))$ as desired. The argument is analogous when X strictly satisfies $(\phi_1 \odot \phi_3)$.

4. The operator $\otimes$ is distributive over $\sqcup$.

   If X strictly 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 strictly satisfies ϕ_1 and X_2 strictly satisfies ϕ_2 or ϕ_3. In this case, by definition, X strictly satisfies $(\phi_1 \otimes \phi_2)$ or $(\phi_1 \otimes \phi_3)$, which means X strictly satisfies $((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$.

   For the other direction, if X strictly satisfies $((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$, it strictly satisfies either $(\phi_1 \otimes \phi_2)$ or $(\phi_1 \otimes \phi_3)$. Without loss of generality, assume that X strictly satisfies $(\phi_1 \otimes \phi_2)$. Then, X can be partitioned into two disjoint sets X_1 and X_2 such that X_1 strictly satisfies ϕ_1 and X_2 strictly satisfies ϕ_2. By definition, X_2 strictly satisfies $(\phi_2 \sqcup \phi_3)$. Therefore, X strictly 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 $\odot$ *is not* distributive over $\sqcap$.
   If X strictly satisfies $(\phi_1 \odot (\phi_2 \sqcap \phi_3))$, then X also strictly 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 $\{u_1, u_2\}$ strictly 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 $\sqcap$ *is not* distributive 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 $\{u_1, u_2\}$ strictly 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 $\{u_1, u_2\}$ strictly 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 $\sqcup$ *is not* distributive over $\odot$.
If X strictly satisfies $(\phi_1 \sqcup (\phi_2 \odot \phi_3))$, then X strictly 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 $\{u_1, u_2\}$ strictly 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 $\sqcup$ *is not* distributive 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 $\{u_1, u_2\}$ strictly satisfies $((r_1 \sqcup r_2) \otimes (r_1 \sqcup r_3))$, but does not strictly 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 $\{u_1\}$ strictly 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 $\otimes$ *is not* distributive over $\sqcap$.
If X strictly satisfies $(\phi_1 \otimes (\phi_2 \sqcap \phi_3))$, then X strictly 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 $\{u_1, u_2\}$ strictly 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 $\sqcap$ *is not* distributive 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 $\{u_1, u_2\}$ strictly 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 $\{u_1, u_2\}$ strictly 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 $\odot$ *is not* distributive 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 $\{u_1, u_2, u_3, u_4\}$ strictly satisfies $((r_1 \odot r_2) \otimes (r_1 \odot r_3))$, but does not strictly 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 $\{u_1, u_2\}$ strictly 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 $\otimes$ *is not* distributive over $\odot$.
If X strictly satisfies $(\phi_1 \otimes (\phi_2 \odot \phi_3))$, then X strictly 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 $\{u_1, u_2\}$ strictly satisfies $((r_1 \otimes r_2) \odot (r_1 \otimes r_3))$, but does not satisfy $(r_1 \otimes (r_2 \odot r_3))$.

B Proof for Theorem 4

Proof.

- $C(\mathsf{All}) = C(r) = \{1\}$ is straightforward.
- That $C(\phi_1 \sqcup \phi_2) = C(\phi_1) \cup C(\phi_2)$ follows from the definition of strict satisfaction (Definition 3), which implies that $V(\phi_1 \sqcup \phi_2) = V(\phi_1) \cup V(\phi_2)$, where $V(\phi)$ denotes the set of all usersets that strictly satisfy ϕ.

- That $C(\phi_1 \sqcap \phi_2) = C(\phi_1) \cap C(\phi_2)$ follows from the definition of strict satisfaction (Definition 3), which implies that $V(\phi_1 \sqcap \phi_2) = V(\phi_1) \cap V(\phi_2)$.

- As to $C(\phi_1 \odot \phi_2)$, let X be a userset that strictly satisfies $(\phi_1 \odot \phi_2)$. There exist X_1, X_2 such that X_1 strictly satisfies ϕ_1, X_2 strictly satisfies ϕ_2, and $X_1 \cup X_2 = X$. By definition of characteristic set, $|X_1| \in [\min(C(\phi_1)), \max(C(\phi_1))]$ and $|X_2| \in [\min(C(\phi_2)), \max(C(\phi_2))]$. Hence, $|X| \in [\max(\min(C(\phi_1)), \min(C(\phi_2))), \max(C(\phi_1)) + \max(C(\phi_2))]$. It follows that $C(\phi_1 \odot \phi_2) \subseteq [\max(\min(C(\phi_1)), \min(C(\phi_2))), \max(C(\phi_1)) + \max(C(\phi_2))]$.

  On the other hand, given an integer $k \in [\max(\min(C(\phi_1)), \min(C(\phi_2))), \max(C(\phi_1)) + \max(C(\phi_2))]$, we can construct a set X of k users and a configuration UR such that X strictly satisfies $(\phi_1 \odot \phi_2)$ under UR. By the range of k, there exit $c_1 \in C(\phi_1)$ and $c_2 \in C(\phi_2)$ such that $c_1 \leq k$, $c_2 \leq k$ and $k \leq c_1 + c_2$. By definition of characteristic set, there exist X_1 and X_2 strictly satisfying ϕ_1, ϕ_2 under UR_1, UR_2 respectively, such that $|X_1| = c_1$ and $|X_2| = c_2$. Name the users in such a way that $|X_1 \cap X_2| = c_1 + c_2 - k$. Let $UR = UR_1 \cup UR_2$. By monotonicity of term satisfaction, X_1, X_2 strictly satisfy ϕ_1, ϕ_2 respectively under UR. Therefore, $X = X_1 \cup X_2$ strictly satisfies $(\phi_1 \odot \phi_2)$ under UR, where $|X| = |X_1| + |X_2| - |X_1 \cap X_2| = k$.

- As to $C(\phi_1 \otimes \phi_2)$, on the one hand, a set X of users strictly 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 strictly 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, X_2 strictly satisfying ϕ_1, ϕ_2 under UR_1, UR_2 respectively, such that $|X_1| = c_1$ and $|X_2| = c_2$. Name the users in such a way that $X_1 \cap X_2 = \emptyset$. Let $UR = UR_1 \cup UR_2$. By monotonicity of term satisfaction, X_1, X_2 strictly satisfy ϕ_1, ϕ_2 respectively under UR. Therefore, $X = X_1 \cup X_2$ strictly satisfies $(\phi_1 \otimes \phi_2)$ under UR, where $|X| = |X_1| + |X_2| = c_1 + c_2$.

∎

C Proofs for Lemmas in Section 4.1

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 $(r_1 \; \mathsf{op} \cdots \mathsf{op} \; r_m)$.

Proof of Lemma 7: UTS $\langle \sqcup, \sqcap, \odot, \otimes \rangle$ **is in NP**

Given a set X of users, a term ϕ and a configuration UR, we want to check whether X satisfies ϕ under UR. We design a nondeterministic Turing machine that decides the problem in polynomial time.

First, the Turing machine computes the syntax tree T of ϕ, where the n leaves in T correspond to the n atomic terms in ϕ and the $n-1$ inner nodes corresponds to the $n-1$ operators in ϕ. Let $t_1, \cdots, t_{2n-1}$ be the sequence of nodes visited by post-order traversal on T. We say that a set of users strictly satisfies t_i if it strictly satisfies the term corresponding to the subtree rooted at t_i. Let $L(t)$ and $R(t)$ denote the left child and the right child of inner node t respectively.

Next, the Turing machine nondeterministically generates a sequence of X's subsets $X_1, \cdots X_{2n-1}$ and does the following to check whether X_i strictly satisfies t_i. Note that $X_i \cup \infty = \infty$.

```
For $i = 1$ To $2n-1$ Do
  Switch $t_i$ Of
    Case $\mathsf{All}$: If $|X_i| \neq 1$ Then $X_i \leftarrow \infty$;
    Case $r$: If $|X_i| \neq \{u\}$ such that $(u,r) \in \mathit{UR}$ Then $X_i \leftarrow \infty$;
    Case $\sqcup$: If $X_i \neq X_{L(t_i)}$ and $X_i \neq X_{R(t_i)}$ Then $X_i \leftarrow \infty$;
    Case $\sqcap$: If $X_i \neq X_{L(t_i)}$ or $X_i \neq X_{R(t_i)}$ Then $X_i \leftarrow \infty$;
    Case $\odot$: If $X_{L(t_i)} \cup X_{R(t_i)} \neq X_i$ Then $X_i \leftarrow \infty$;
    Case $\otimes$: If $X_{L(t_i)} \cup X_{R(t_i)} \neq X_i$ or $X_{L(t_i)} \cap X_{R(t_i)} \neq \emptyset$ Then $X_i \leftarrow \infty$;
  EndSwitch
EndFor
If $X_{2n-1} = \infty$ Then return false Else return true
```

In the above algorithm, X_i is changed to ∞ if it does not strictly satisfy the term corresponding to the subtree rooted at t_i. The correctness of the algorithm follows immediately from the definition 3. As a nondeterministic Turing runs the algorithm in Polynomial time, UTS $\langle \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP}$.

Proof of Lemma 8: UTS $\langle \sqcap, \odot \rangle$ **is NP-hard.**

We use a reduction from the $\mathbf{NP}$-complete SET COVERING problem [8]. In the set covering problem, we are given a family $F = \{S_1, \cdots, S_m\}$ of subsets of a finite set S and an integer k no larger than m, and we ask 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 $F = \{S_1, \cdots, S_m\}$, we construct a configuration UR such that $(u_i, r_j) \in \mathit{UR}$ if and only if $e_j \in S_i$. Let $U = \{u_1, \cdots, u_m\}$ and $\phi = ((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^n r_i))$.

We now demonstrate that U satisfies ϕ under UR if and only if there are no more than k sets in family F whose union is S.

If U satisfies ϕ, by definition, a subset U' of U strictly satisfies $(\bigodot_k \mathsf{All})$ and $(\bigodot_{i=1}^n r_i)$. U' strictly satisfying $(\bigodot_k \mathsf{All})$ indicates that $|U'| \leq k$, while U' strictly satisfying $(\bigodot_{i=1}^n r_i)$ indicates that users in U' together have membership of r_i for every i from 1 to n. Without loss of generality, suppose $U' = \{u_1, \cdots, u_t\}$, where $t \leq k$. As $(u_i, r_j) \in \mathit{UR}$ 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, 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_i for every i. In this case, $\{u_1, \cdots, u_k\}$ satisfies $(\bigodot_{i=1}^n r_i)$. Also, $\{u_1, \cdots, u_k\}$ strictly satisfies $(\bigodot_k \mathsf{All})$. Hence, U satisfies ϕ.

Proof of Lemma 9: UTS $\langle \odot, \otimes \rangle$ **is NP-hard.**

We use a reduction form the $\mathbf{NP}$-complete DOMATIC NUMBER problem [8]. Given a graph $G(V, E)$, the Domatic Number problem asks whether V can be partitioned into k disjoint 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. $\mathit{UR} = \{(u_i, r_j) \mid i = j \text{ or } (v(u_i), v(u_j)) \in E\}$. 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 if and only if U has a subset U' that can be divided into k pairwise disjoint sets, each of which have role membership of $r_1, r_2, \cdots, r_n$. Therefore, the answer to the Domatic Number problem is "yes" if and only if U satisfies ϕ.

Proof of Lemma 10: $\mathsf{UTS}\ \langle \otimes, \sqcup \rangle$ **is NP-hard.**

We use a reduction from the **NP**-complete SET PACKING problem [8], which asks, given a family $F = \{S_1, \cdots, S_m\}$ of subsets of a finite set S and an integer k, whether there are k pairwise disjoint sets in family F. Without loss of generality, we assume that $S_i \not\subseteq S_j$ if $i \neq j$.

Given $S = \{e_1, \cdots, e_n\}$ and a family of S's subsets $F = \{S_1, \cdots, S_m\}$, let $U = \{u_1, \cdots, u_n\}$, $R = \{r_1, \cdots, r_n\}$ and $UR = \{(u_i, r_i) \mid 1 \leq i \leq n\}$. We then construct a term $\phi = (\bigotimes_k (\bigsqcup_{i=1}^m (\bigotimes R_j)))$, where $R_j = \{r_i \mid e_i \in S_j\}$. We show that U satisfies ϕ under UR 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 strictly satisfies $\phi_i = (\bigotimes R_j)$ is $U_j = \{u_i \mid e_i \in S_j\}$. A userset X strictly 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 strictly satisfies ϕ_1, and thus strictly satisfies ϕ'. Similarly, we have U_i strictly satisfies ϕ' for every i from 1 to k. Since $U_i \subseteq U$, U satisfies ϕ.

On the other hand, suppose U satisfies ϕ. Then, U has a subset U' that can be divided into k pairwise disjoint sets $\hat{U_1}, \cdots, \hat{U_k}$, such that $\hat{U_i}$ strictly satisfies ϕ_i. In order to strictly satisfy ϕ', $\hat{U_i}$ must strictly satisfy a certain ϕ_{a_i} and hence be equivalent to U_{a_i}. 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".

Proof of Lemma 11: $\mathsf{UTS}\ \langle \sqcap, \otimes \rangle$ **is NP-hard.**

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 $\phi = (\bigsqcap_{i=1}^n (r_i \otimes (\bigotimes_{k-1} \mathsf{All})))$. We now demonstrate that U satisfies ϕ under UR if and only if there are k sets in family F whose union is S.

If U satisfies ϕ, by definition, a subset U' of U strictly satisfies $(r_i \otimes (\bigotimes_{k-1} \mathsf{All}))$ for every i, which means users in U' together have membership of r_i for every i. As $C(r_i \otimes (\bigotimes_{k-1} \mathsf{All})) = \{k\}$, $|U'| = k$. Suppose $U' = \{u_{a_1}, \cdots, u_{a_k}\}$. 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_i for every i from 1 to n. In this case, as at least k users are required to satisfy ϕ_i, $\{u_1, \cdots, u_k\}$ strictly satisfies ϕ_i for every i from 1 to n, which indicates that it strictly satisfies ϕ.

D Background on the Polynomial Hierarchy

Oracle Turing Machine An oracle Turing machine, with oracle L, is denoted as M^L. L is a language. M^L can use the oracle to determine whether a string is in L or not in one step. More precisely, M^L is a two-tape deterministic Turing machine. The extra tape is called the oracle tape. M^L has three additional states: $q_?$ (the query state), and q_{yes} and q_{no} (the answer states). The computation of M^L proceeds like in any ordinary Turing machine, except for transitions from $q_?$. When M^L enters $q_?$, it checks whether the contents of the oracle tape are in L. If so, M^L moves to q_{yes}. Otherwise, M^L moves to q_{no}. In other words, M^L is given the ability to "instantaneously" determine whether a particular string is in L or not.

Polynomial Hierarchy The polynomial hierarchy provides a more detailed way of classifying NP-hard decision problems. The complexity classes in this hierarchy are denoted by $\Sigma_k\mathbf{P}, \Pi_k\mathbf{P}, \Delta_k\mathbf{P}$, where k is a nonnegative integer. They are defined as follows:

$$\Sigma_0\mathbf{P} = \Pi_0\mathbf{P} = \Delta_0\mathbf{P} = \mathbf{P},$$

and for all $k \geq 0$,

$$\Delta_{k+1}\mathbf{P} = \mathbf{P}^{\Sigma_k\mathbf{P}},$$
$$\Sigma_{k+1}\mathbf{P} = \mathbf{NP}^{\Sigma_k\mathbf{P}},$$
$$\Pi_{k+1}\mathbf{P} = \mathbf{co}\text{-}\Sigma_{k+1}\mathbf{P} = \mathbf{coNP}^{\Sigma_k\mathbf{P}}.$$

Some classes in the hierarchy are

$$\Delta_1\mathbf{P} = \mathbf{P}\ , \Sigma_1\mathbf{P} = \mathbf{NP}\ , \Pi_1\mathbf{P} = \mathbf{coNP},$$
$$\Delta_2\mathbf{P} = \mathbf{P}^{\mathbf{NP}}, \Sigma_2\mathbf{P} = \mathbf{NP}^{\mathbf{NP}}, \Pi_2\mathbf{P} = \mathbf{coNP}^{\mathbf{NP}}.$$

E Proofs of Lemmas in Section 5

Proof of Lemma 19: $\mathsf{SSC}\langle\otimes\rangle$ **is coNP-complete.**

We can reduce the Set Covering problem to the complement of $\mathsf{SSC}\langle\otimes\rangle$. In Set Covering problem, we are given a family $F = \{S_1, \cdots, S_m\}$ of subsets of a finite set $S = \{e_1, \cdots, e_n\}$ and a goal K. We are asking for a set of K sets in F whose union is S.

Given an instance of the Set Covering problem, construct a configuration $\langle UR, UP\rangle$ such that $(u_i, r_i) \in UR$ and $(u_i, p_j) \in UP$ if and only if $e_j \in S_i$. Construct a safety policy $E = \mathsf{sp}\langle P, \phi\rangle$, where $P = \langle p_1, \cdots, p_n\rangle$ and $\phi = (\bigotimes_{K+1} \mathsf{All})$. ϕ is satisfied by any set of no less than $K+1$ users.

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

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

Since the Set Covering problem is $\mathbf{NP}$-complete, we conclude that the complement of $\mathsf{SSC}\langle\otimes\rangle$ is $\mathbf{NP}$-hard. Hence, $\mathsf{SSC}\langle\otimes\rangle$ is $\mathbf{coNP}$-hard.

Proof of Lemma 20: $\mathsf{SSC}\langle\sqcup, \odot\rangle$ **is coNP-complete.** We reduce the $\mathbf{coNP}$-complete validity problems for propositional logic to $\mathsf{SSC}\langle\sqcup, \odot\rangle$. Given a propositional logic formula φ using $\wedge$, $\vee$, and $\neg$, let $v_1, \cdots, v_n$ be the propositional variables in φ.

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

We now show that the formula φ is valid if and only if $\langle UR, UP\rangle$ is safe with respect to the policy $\mathsf{sp}\langle\{p_1, p_2, \cdots, p_n\}, \phi\rangle$. If the formula φ is not valid, then there is an assignment I that makes it false. Using the assignment, we construct a userset $X = \{u_i \mid I(v_i) = 1\} \cup \{u_i' \mid I(v_i) = 0\}$. X covers all permissions in P, and X does not satisfy ϕ. If $\langle UR, UP\rangle$ is not safe with respect to $\mathsf{sp}\langle\{p_1, p_2, \cdots, p_n\}, \phi\rangle$, then there exists a set X of users that together have all permissions in P and does not satisfy ϕ. In order to cover all permissions in P, for each i at least one of u_i, u_i' is in X. Without loss of generality, assume that for each i, exactly one of u_i, u_i' is in X. If both u_i, u_i' are in X, we can remove either one, the resulting set still has all permissions in P but clearly does not satisfy ϕ because of the monotonicity property. Then we can derive an truth assignment from X so that the formula φ evaluates to false.

Proof of Lemma 21: $\mathsf{SSC}\langle\odot\rangle$ **is in P.**

Given a static safety policy $\mathsf{sp}\langle P, \phi\rangle$ where ϕ uses only the operator $\odot$, and a configuration $\langle UR, UP\rangle$. We need to determines whether $\langle UR, UP\rangle$ is safe with respect to $\mathsf{sp}\langle P, \phi\rangle$.

We observe that, as $\odot$ is associative, ϕ can be equivalently written as$\phi_1 \odot \phi_2 \cdots \odot \phi_n$, where each ϕ_i is an atomic term, and ϕ is satisfied by a nonempty userset X if and only if for each role in ϕ, there exists a user u in X such that u is a member of the role. Based on this observation, we have the following algorithm.

```
isSafe($P$, $\phi$, $UR$ , $UP$)
begin
   $R_\phi$ = all roles that appear in $\phi$
   $R$ = $R_\phi$
   For each $p$ in $P$ do
      $R_p$ = $\emptyset$
      For each $u$ such that $(u,p) \in UP$ do
         $R_p$ = $R_p$ $\cup$ ({ $r$ $\in$ $R_\phi$ | $(u,r) \notin$ $UR$ })
      EndFor;
      $R$ = $R$ $\cap$ $R_p$
   EndFor;
   if ($R$ == $\emptyset$) return true
   else return false
end
```

If the algorithm returns false, it means that there is a role $r \in R_\phi$ such that $r \in R_p$ for each $p \in P$. That $r \in R_p$ means that there exists a user u such that $(u, p) \in UP$ and $r \in \{\ r \in R_\phi | (u, r) \notin UR\ \}$. In other words, there exists a user u such that u has permission p but is not a member of r. By choosing one such user for each p, we have a userset that covers all permissions, but does not satisfy ϕ.

If the algorithm returns true, it means that for every role in R_ϕ, there exists a permission p such that $r \notin R_p$, which means that $(u, r) \in UR$ for each u such that u has the permission p. In order to have the permission p, one thus have to pick a user that satisfies r. As every role in R_ϕ can be satisfied by any userset that together have all permissions in P, ϕ is also satisfied by any such userset. The configuration is safe.

We assume that whether $(u, r) \in UR$ and $(u, p) \in UP$ can be determined in constant time. Let $|U|$ be the set of users appear in UR and R_ϕ be the set of roles appear in ϕ. Note that $|R_\phi| \leq |\phi|$. The running time of the first two steps of the algorithm is $O(|\phi|)$. The outer loop runs $|P|$ times and the inner loop runs $|U|$ times. Computing the set union within the inner loop takes time $O(|R| + |P|)$. In general, the running time of the above algorithm is $O(|\phi| + |P| \cdot |U| \cdot (|R| + |P|))$.

Lemma 23 $\mathsf{SSC}\langle\sqcup, \sqcap\rangle$ *is in* **P**.

Proof. Given a static safety policy $\mathsf{sp}\langle P, \phi\rangle$ where ϕ uses only operators in $\{\sqcup, \sqcap\}$, and a configuration $\langle UR, UP\rangle$. As $C(\phi) = \{1\}$, ϕ is strictly satisfied only by singleton userset. The following algorithm determines whether $\langle UR, UP\rangle$ is safety with respect to $\mathsf{sp}\langle P, \phi\rangle$.

```
isSafe($P$, $\phi$, $UR$ , $UP$)
begin
   For each $p$ in $P$ do
```

```
        res = true;
        For each $u$ such that $(u,p) \in UP$ do
           If ($\{u\}$ does not strictly satisfies $\phi$ under $UR$) then res = false;
        EndFor;
    if (res==true) return true
    EndFor;
    return false;
end
```

Determining whether a singleton userset strictly satisfies a term or not can be done in term linear in the size of the term. Therefore, the above algorithm is clearly polynomial in the size of the instance.

The algorithm returns true when there exists one permission in P such that all users who have the permission satisfy the term ϕ. As any userset that covers all permissions in P must include a user among these, all such usersets satisfy ϕ, the answer true is thus correct. The algorithm returns false, when for every permission in P, there exists a user that has the permission yet does not satisfy ϕ. The userset that consists of all these users has all permissions in P, yet does not satisfy ϕ.

We assume that whether $(u, r) \in UR$ and $(u, p) \in UP$ can be determined in constant time. Let $|U|$ be the set of users appear in UR. The outer loop of the algorithm runs $|P|$ times and the inner loop runs $|U|$ times. Checking whether a user strictly satisfies a term can be done in time $O(|\phi|)$. In general, the running time of the above algorithm is $O(|P| \cdot |U| \cdot |\phi|)$. ∎