CERIAS Tech Report 2006-40

DIRECT STATIC ENFORCEMENT OF HIGH-LEVEL SECURITY POLICIES

by Qihua Wang, Ninghui Li

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

Direct Static Enforcement of High-Level Security Policies

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

Abstract

A high-level security policy states an overall safety 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. Recently, Li and Wang [4] proposed an algebra for specifying a wide range of high-level security policies with both qualification and quantity requirements on users who perform a task. In this paper, we study the problem of direct static enforcement of high-level security policies expressed in this algebra. We formally define the notion of a static safety policy, which requires that every set of users together having all permissions needed to complete a sensitive task must contain a subset that satisfies the corresponding security requirement expressed as a term in the algebra. The static safety checking problem asks whether an access control state satisfies a given high-level policy. We study several computational problems related to the static safety checking problem, and design and evaluate an algorithm for solving the problem.

1 Introduction

A high-level security policy states an overall safety requirement for a sensitive task. One well-known high-level security policy is Separation of Duty (SoD). In its simplest form, an SoD policy states that a sensitive task should be performed by two different users acting in cooperation. More generally, an SoD policy requires the cooperation of at least k $(k \geq 2)$ different users to complete the task. SoD is a high-level policy because it does not place restrictions on which users are allowed to perform which individual steps in a sensitive task, but instead states an overall requirement that must be satisfied by any set of users that together complete the task. An SoD policy states only a quantity requirement and does not express qualification requirements on users who complete a task. Recently, Li and Wang [4] proposed an algebra that enables the specification of high-level policies that combine qualification requirements with quantity requirements. To use the algebra to specify high-level security policies, the administrators first identify sensitive tasks and then, for each sensitive task t, specifies a security policy of the form $\langle t, \phi \rangle$, where ϕ is a term in the algebra. This policy means that any set of users (we call *userset*) that together complete the task must satisfy the term ϕ. The algebra has three kinds of atomic terms: a role (which implicitly identifies a set of users), the keyword All (which refers to the set of all users), and an explicitly listed set of users. Two unary operators, $\neg$ and $+$, and four binary operators, $\sqcup$, $\sqcap$, $\odot$, and $\otimes$, can be used with these atomic terms to form more sophisticated terms. Li and Wang [4] gave many examples to illustrate the expressive power of the algebra. For instance, a simple SoD policy that requires at least two different users can be expressed using the term $(\mathsf{All} \otimes \mathsf{All}^+)$. A more sophisticated policy that requires two `Clerks` plus a third user who is either a `Treasurer` or a `Manager` can be expressed using the term $(\texttt{Clerk} \otimes \texttt{Clerk} \otimes (\texttt{Treasurer} \sqcup \texttt{Manager}))$.

A high-level policy can be enforced either statically or dynamically. In dynamic enforcement, one identifies all steps in performing the task, and maintains, for each instance of the task, the history of which user has performed which steps. When a user requests to perform the next step, the request is authorized only when the overall security requirement can be met by allowing this user to perform the next step. In static enforcement, one identifies the set of permissions that are necessary to perform the task, and ensures that each access control state that can be reached is safe with respect to the policy for the task. An access control state is safe if each userset such that users in the set together have all the permissions for the task (in which case we say the userset *covers* the permissions for the task) satisfies the security requirement. Static enforcement can be achieved either directly or indirectly. In direct static enforcement, before making changes to the access control state, one checks that the resulting state is safe and makes the change only when it is safe. In indirect static enforcement, one specifies constraints so that any access control state satisfying the constraints is safe and thus only needs to check whether a resulting state satisfies the constraints during state changes.

In this paper we study direct static enforcement of policies specified in the algebra proposed by Li and Wang [4]. Direct static enforcement of SoD policies, which are a subclass of the policies that can be specified in the algebra, has been studied by Li et al [3]. It has been shown that checking whether an access control state satisfies an Static SoD (SSoD) policy, i.e., whether every userset that covers the permissions for the task contains at least k users, is $\mathbf{coNP}$-complete [3]. As a policy specified in the algebra can be more expressive and sophisticated than an SSoD policy, it is expected that the problem considered in this paper is also in an intractable computational complexity class. Computationally expensive notwithstanding, we argue that the study of direct enforcement of static high-level policies should be given higher priority than indirect static enforcement and dynamic enforcement for the following reasons. First, direct static enforcement is the most simple and straightforward enforcement mechanism for high-level security policies. Its performance will be used as a benchmark for comparison when evaluating other enforcement mechanisms. Second, even though direct static enforcement is computationally intractable in theory, it is interesting and necessary to study its performance for instances that are likely to occur practice. Third, direct enforcement cannot be entirely replaced by indirect enforcement. It is oftentimes difficult or even impossible to generate efficiently-verifiable constraints to precisely capture a high-level policy. For example, Li et al. [3] studied indirect enforcement of using Static Mutually Exclusive Roles (SMER) to enforce SSoD policies in the context of role-based access control (RBAC), and showed that there exist SSoD policies such that no set of SMER constraints can *precisely* capture them [3]. Most of the time, the set of constraints generated for a security policy is more restrictive than the policy itself. That is to say, some access control states that are safe with respect the security policy will be ruled out by the constraints. In situations where precise enforcement is desired, direct static enforcement is more desirable than indirect static enforcement. We consider dynamic enforcement and indirect static enforcement interesting future research problems.

In direct static enforcement, we need to solve the following problem: Given an access control state, determine whether each userset that covers all permissions for a task is safe with respect to the term associated with the task, we call this the Static Safety Checking problem. To solve this, we must first solve that problem of checking whether a given userset is safe with respect to a term; we call this the Userset-Term Safety Checking problem.

Our contributions in this paper are as follows:

1. We formally define the notion of static safety polices and the Static Safety Checking problem. We

also give a necessary and sufficient condition for a static safety policy to be satisfiable.

2. We study the computational complexity of the Userset-Term Safety Checking problem.

3. We study computational complexity of the Static Safety Checking problem. We show that the Static Safety Checking problem is both $\mathbf{NP}$-hard and $\mathbf{coNP}$-hard and is in $\mathbf{NP}^{\mathbf{NP}}$, a complexity class in the Polynomial Hierarchy. Furthermore, we show that several subcases of the problem remain intractable. Finally, we identify syntactic restrictions so that if the term in a safety policy satisfies the restrictions, then determining whether a state satisfies the policy can be solved in polynomial time.

4. We present an algorithm for the Static Safety Checking problem. Our algorithm uses pruning techniques that reduce the number of users and usersets needed to be considered. Furthermore, we design an abstract representation of usersets that can reduce the memory storage requirement and accelerate set operations, which leads to a fast bottom-up approach for solving the Userset-Term Safety Checking problem.

The remainder of this paper is organized as follows. In Section 2, we review the algebra. In Section 3, we define static safety policy, the Static Safety Checking problem and the notion of policy satisfiability. We present computational complexities of the Static Safety Checking problem in Section 4, and an algorithm for the problem as well as its evaluation in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2 Preliminary

In this section, we give a brief overview of the algebra introduced in [4] and then discuss potential enforcement mechanisms for policies specified in the algebra. The algebra is motivated by the following limitation of SoD policies: 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. Previous 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. This approach, however, results in the loss of the several important advantages offered by a higher-level policy. The algebra enables one to specify, at a high-level, a wide range of security policies with both qualification and quantity requirements on users who perform a task. For more information on the algebra beyond that in this section, readers are referred to [4].

We use $\mathcal{U}$ to denote the set of all users and $\mathcal{R}$ to denote the set of all roles. In the algebra, a role is simply a named set of users. The notion of roles can be replaced by groups or user attributes.

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.

Before formally assigning meanings to terms, it is necessary to assign meanings to the roles used in the term. The following definition introduces the notion of configurations.

Definition 2 (Configurations). A *configuration* is given by a pair $\langle U, UR \rangle$, where $U \subseteq \mathcal{U}$ denotes the set of all users in the configuration, and $UR \subseteq U \times \mathcal{R}$ determines role memberships. We say that u is a member of the role r under a configuration $\langle U, UR \rangle$ if and only if $(u, r) \in UR$.

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.
- 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 $X_1 \cap X_2 = \emptyset$.

It has been shown that the four binary operators are commutative and associative. We are thus able to omit some parenthesis when writing the terms without introducing ambiguity. Note that term satisfaction does not have the monotonicity property. In other words, a userset X satisfying a term ϕ does not imply that any superset of X also satisfies ϕ. This design was chosen in [4] because it has more expressive power. For example, a policy that requires (1) everyone involved in a task must be a `Accountant`, can be expressed as $\texttt{Accountant}^+$, and (2) there must be at least two users involved, can be expressed as $(\texttt{Accountant} \otimes \texttt{Accountant}^+)$. The policy cannot be expressed in an algebra that has the monotonicity property, because this property mandates that a set containing two accountants and one non-accountant user (which is a superset of the set containing just the two accountants) satisfies the term.

The following examples demonstrate 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.

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

- $(\texttt{Accountant} \sqcup \texttt{Treasurer})^+$

  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.

2.1 The Enforcement of High-Level Security Policies

A problem that naturally arises is how to enforce high-level security policies specified in the algebra. There are two dimensions in policy enforcement. A high-level security policy specified in the algebra may be enforced either *statically* or *dynamically*, and either *directly* or *indirectly*.

To dynamically enforce a policy $\langle t, \phi \rangle$, where t is a task and ϕ is a term in the algebra, one identifies the steps in performing the task t, and maintains a history of each instance of the task, which includes who has performed which steps. Given a task instance, let U_{past} be the set of users who have performed at least one step of the instance. A user u is allowed to perform a next step on the instance only if there exists a superset of $U_{past} \cup \{u\}$ that can satisfy ϕ upon finishing all steps of the task. In direct dynamic enforcement, the system solves this problem directly each time a user requests to perform a step. In indirect dynamic enforcement, the system uses authorization constraints on the steps in the task (e.g., two steps cannot be performed by the same user) to enforce that the policy is satisfied. For example, there are three users, say $Alice$, Bob and $Carl$, in the system. $Alice$ is a member of role r_1; Bob is a member of both r_1 and r_3; $Carl$ is a member of r_2 and r_4. There is a task consisting of two steps and any user is authorized to perform any step. Let $\phi = (r_1 \otimes r_2) \sqcap (r_3 \otimes r_4)$ be a term associated with the task. Either Bob or $Carl$ may perform the first step of the task. The reason is that if Bob (or $Carl$) performs the first step, then $Carl$ (or Bob) may perform the second step to finish that task and the userset $\{Bob, Carl\}$ satisfies ϕ. However, $Alice$ is not allowed to perform the first step (nor the second step) of the task, as any superset of $\{Alice\}$ in the system does not satisfy ϕ.

To statically enforce the policy $\langle t, \phi \rangle$, one identifies the set P of all permissions that are needed to perform the task t and requires that any userset that covers P satisfies the term ϕ. We denote such a security policy $\mathsf{sp}\langle P, \phi \rangle$ and call it a *static safety policy*. A static safety policy can be satisfied by careful design (such as careful permission assignments) of the access control state, without maintaining a history for each task instance. In direct static enforcement, before making changes to the access control state, one checks that the resulting state is safe with respect to the static safety policy and makes the change only when it is safe. In indirect enforcement, one specifies constraints so that any access control state satisfying the constraints is safe with respect to the policy (but possibly not the other way around) and thus only needs to check whether a resulting state satisfies the constraints during state changes.

In this paper, we focus on direct static enforcement. Investigating other enforcement approaches for policies specified in the algebra is beyond the scope of this paper.

3 The Static Safety Checking (SSC) Problem

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

Definition 4 (State)**.** An access control system *state* is given by a triple $\langle U, UR, UP\rangle$, where $UR \subseteq U \times \mathcal{R}$ determines user-role memberships and $UP \subseteq U \times \mathcal{P}$ determines user-permission assignment, where $\mathcal{P}$ is the set of all permissions. We say that a userset X covers a set P of permissions if and only if the following holds: $\bigcup_{u \in X}\{\, p \in P \mid (u,p) \in UP \,\} \supseteq P$.

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

Definition 5 (Term Safety)**.** A userset X is *safe* with respect to a term ϕ under configuration $\langle U, UR\rangle$ if and only if there exists $X' \subseteq X$ such that X' satisfies ϕ under $\langle U, UR\rangle$.

Definition 6 (Static Safety Policy)**.** A *static safety policy* is given as a pair $\mathsf{sp}\langle P, \phi\rangle$, where $P \subseteq \mathcal{P}$ is a set of permissions and ϕ is a term in the algebra. An access control state $\langle U, UR, UP\rangle$ *satisfies* the policy $\mathsf{sp}\langle P, \phi\rangle$, if and only if, for every userset X that covers P, X is safe with respect to ϕ. If a state satisfies a policy, we say that it is *safe* with respect to the policy.

Note that in the above definition, we require that each userset X that covers P is safe with respect to ϕ (Definition 5) rather than that X satisfies ϕ (Definition 3). The reason is that permission coverage is monotonic with respect to userset. In other words, if X covers P then any superset of X also covers P. However, as we pointed out right after Definition 3, term satisfaction does not have the monotonicity property. This means that static enforcement can be applied only for policies that have the monotonicity property. We thus define safety with respect to a static safety policy in a monotonic fashion.

Definition 7 (Static Safety Checking (SSC) Problem)**.** Given a static safety policy $\mathsf{sp}\langle P, \phi\rangle$, the problem of determining whether a given state $\langle U, UR, UP\rangle$ is safe with respect to $\mathsf{sp}\langle P, \phi\rangle$ is called the *Static Safety Checking* (SSC) problem.

We will study the computational complexity of SSC in Section 4. In the rest of this section, we study two other problems related to static safety policies.

3.1 Satisfiability of Static Safety Policies

Given a static safety policy, it is natural to ask whether it is possible to satisfy the policy at all. In particular, if a static safety policy cannot be satisfied by any access control state, it is probably not what the designers of the policy desire.

Definition 8 (Policy Satisfiability)**.** A static safety policy $\mathsf{sp}\langle P, \phi\rangle$ is *satisfiable* if and only if there exists a state $\langle U, UR, UP\rangle$ such that $\langle U, UR, UP\rangle$ satisfies $\mathsf{sp}\langle P, \phi\rangle$ and there is at least one userset in $\langle U, UR, UP\rangle$ that covers P.

Note that the above definition requires that there exists at least one userset in $\langle U, UR, UP\rangle$ that covers P. Without this requirement, a state γ trivially satisfies $\mathsf{sp}\langle P, \phi\rangle$, if γ does not contain any userset that covers P. In particular, an empty access control state satisfies any static safety policy; and thus any static safety policy is trivially satisfiable.

A term ϕ is *satisfiable* if there exists a userset X and a configuration $\langle U, UR\rangle$, such that X satisfies ϕ under $\langle U, UR\rangle$. From Definition 8, it is clear that when ϕ is unsatisfiable, a static safety policy $\mathsf{sp}\langle P, \phi\rangle$ is unsatisfiable as well. However, even if ϕ is satisfiable, it is still possible that $\mathsf{sp}\langle P, \phi\rangle$ is unsatisfiable. For example, $\mathsf{sp}\langle \{p_1, p_2\}, \mathtt{Clerk} \otimes \mathtt{Accountant} \otimes \mathtt{Manager}\rangle$ is unsatisfiable, as a minimal set of users having all permissions in $\{p_1, p_2\}$ contains at most two users, while a set of at least three users are required to satisfy the term ($\mathtt{Clerk} \otimes \mathtt{Accountant} \otimes \mathtt{Manager}$).

The following theorem states a necessary and sufficient condition for a static safety policy to be satisfiable. Intuitively, a policy $\mathsf{sp}\langle P, \phi\rangle$ is satisfiable when the number of permissions in P is no smaller than the size of the smallest userset that satisfies ϕ.

Theorem 1. Let k be the smallest number such that there exists a size-k userset X and a configuration $\langle U, UR\rangle$, such that X satisfies ϕ under $\langle U, UR\rangle$. $\mathsf{sp}\langle P, \phi\rangle$ is satisfiable if and only if $|P| \geq k$.

Proof. Let X be a sized-k userset that satisfies ϕ under $\langle U, UR\rangle$. On the one hand, if $|P| \geq k$, we can construct an access control state $\langle U, UR, UP\rangle$ such that $X \subseteq U$ and X is the only userset that covers P. In this case, $\langle U, UR, UP\rangle$ satisfies $\mathsf{sp}\langle P, \phi\rangle$. On the other hand, if $|P| < k$, assume by contradiction that there exists a state $\langle U, UR, UP\rangle$ that satisfies $\mathsf{sp}\langle P, \phi\rangle$. Then, there exists a userset $X' \subseteq U$ such that X' covers P and X' satisfies ϕ. We have $|X'| \leq |P| < k$. This contradicts the assumption that there does not exist a userset with less than k users that satisfies ϕ. In general, $\mathsf{sp}\langle P, \phi\rangle$ is satisfiable if and only if $|P| \geq k$. □

3.2 The Userset-Term Safety Problem

To solve the SSC problem, which asks whether every userset that covers a set of permissions is safe with respect to a term ϕ, we need to solve the problem of determining whether a given userset is safe with respect to a term.

Definition 9 (Userset-Term Safety (SAFE) Problem)**.** Given a userset X and a term ϕ, the problem of determining whether X is safe with respect to ϕ is called the *Userset-Term Safety (*SAFE*) Problem.*

SAFE is related to yet different from the Userset-Term Satisfaction (UTS) problem studied in [4]. SAFE asks whether X contains a subset that satisfies a term ϕ under a configuration; this is monotonic in that if X is safe, then any superset of X is also safe. UTS asks whether a userset X satisfies a term ϕ under a configuration $\langle U, UR\rangle$; this is not monotonic, as discussed in Section 2. This difference has subtle but important effects. For example, under SAFE, the operator $\odot$ is equivalent to logical conjunction, that is, X is safe with respect to $\phi_1 \odot \phi_2$ if and only if X is safe with respect to both ϕ_1 and ϕ_2. This is because X is safe with respect to $\phi_1 \odot \phi_2$ if and only if X contains a subset X_0 that is the union of two subsets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. This is equivalent to X contains two subsets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. On the other hand, the operator $\odot$ is different from logical conjunction under UTS. That X satisfies $\phi_1 \odot \phi_2$ does not imply X satisfies both ϕ_1 and ϕ_2. For example $\{u_1, u_2\}$ satisfies $\mathsf{All} \odot \mathsf{All}$, but does not satisfy All, because term satisfaction is not monotonic. Another difference regards the operation $\sqcap$. The operator $\sqcap$ is equivalent to logical conjunction under UTS, by definition of term satisfaction. On the other hand, $\sqcap$ is stronger than logical conjunction under SAFE. That X is safe with respect to $\phi_1 \sqcap \phi_2$ implies that X is safe with respect to both ϕ_1 and ϕ_2, but the other

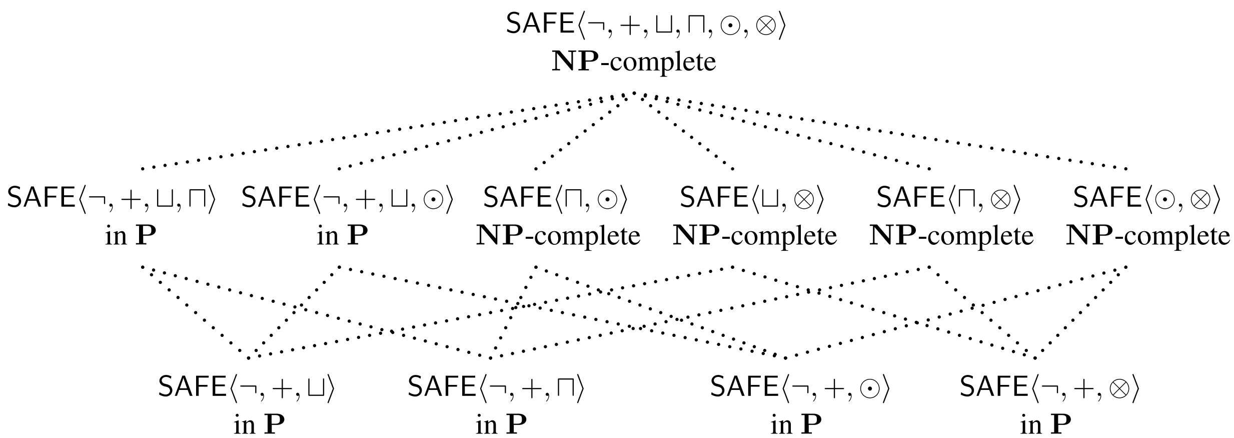

Figure 1: Various sub-cases of the Userset-Term Safety (SAFE) problem and the corresponding time-complexity. Time-complexity of other subcases can be implied from the subcases shown in the figure.

direction is not true. For example, given $UR = \{(u_1, r_1), (u_2, r_2)\}$, $X = \{u_1, u_2\}$ is safe with respect to both r_1 and r_2, but is not safe with respect to $r_1 \sqcap r_2$.

Because of these and other differences, the computational complexity results about UTS do not naturally imply computational complexity results for SAFE. In the rest of this section, we give the computational complexities for SAFE and compare them with those of UTS. We show that SAFE in the most general case (i.e., arbitrary terms in which all operators are allowed) is $\mathbf{NP}$-complete. In order to understand how the operators affect the computational complexity, we consider all sub-algebras in which only some subset of the six operators in $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed. For example, SAFE$\langle \neg, +, \sqcup, \sqcap \rangle$ denotes the sub-case of SAFE where ϕ does not contain operators $\odot$ or $\otimes$, while SAFE$\langle \otimes \rangle$ denotes the sub-case of SAFE where $\otimes$ is the only kind of operator in ϕ. SAFE$\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ denotes the general case.

Theorem 2. The computational complexities for SAFE and its subcases are given in Figure 1.

According to Figure 1, the computational complexities of all subcases of SAFE are the same as those of UTS except for the subcase in which only operators in $\{\neg, +, \sqcup, \odot\}$ are allowed. SAFE$\langle \neg, +, \sqcup, \odot \rangle$ is in $\mathbf{P}$, while UTS$\langle \sqcup, \odot \rangle$ is $\mathbf{NP}$-hard. Intuitively, UTS$\langle \sqcup, \odot \rangle$ is computationally more expensive than SAFE $\{\sqcup, \odot\}$ for the following reason: given a term $\phi = (\phi_1 \odot \cdots \odot \phi_m)$ and a userset U, U is safe with respect to ϕ if and only if U is safe with respect to ϕ_i for every $i \in [1, m]$. In other words, for SAFE, one may check whether U is safe with respect to ϕ_i independently from ϕ_j $(i \neq j)$. However, when it comes to UTS, such independency no longer exists and one has to take into account whether every user in U is used to satisfied some ϕ_i in the term ϕ.

Proofs for the $\mathbf{P}$ results in Theorem 2 To prove all the $\mathbf{P}$ results in Figure 1, it suffices to prove that the three cases SAFE$\langle \neg, +, \sqcap, \sqcup \rangle$, SAFE$\langle \neg, +, \sqcup, \odot \rangle$, and SAFE$\langle \neg, +, \otimes \rangle$ are in $\mathbf{P}$. We first prove the following lemma, which will be prove useful. We need the following definition taken from [4].

Definition 10. A term is *in level-1 canonical form* (called a 1CF term) if it is t or t^+, where t is a unit term. Recall that a unit term can use the operators $\neg$, $\sqcap$, and $\sqcup$.

Lemma 3. The following Properties hold.

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

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

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

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

Lemma 4. SAFE $\langle \neg, +, \sqcup, \odot \rangle$ is in $\mathbf{P}$.

Proof. A userset X is safe with respect to $(\phi_1 \sqcup \phi_2)$ if and only if either X is safe with respect to ϕ_1 or X is safe with respect to ϕ_2. Furthermore, X is safe with respect to $(\phi_1 \odot \phi_2)$ if and only if X is safe with respect to both ϕ_1 and ϕ_2. Therefore, one can determine whether U is safe with respect to ϕ that uses only the operators in $\{\neg, +, \sqcup, \odot\}$ by following the structure of the term until reaching subterms in 1CF. From Property 4 of Lemma 3, checking whether U is safe with respect to such a term amounts to checking whether there exists a user in U that satisfies t, which can be done in polynomial time. □

Lemma 5. SAFE $\langle \neg, +, \sqcup, \sqcap \rangle$ is in $\mathbf{P}$.

Proof. Given a term ϕ using only operators in $\{\neg, +, \sqcup, \sqcap\}$, we prove that a userset X is safe with respect to ϕ if and only if there exists a user $u \in X$ such that u satisfies ϕ. The "if" direction follows by definition. For the "only if" direction: Suppose that X contains a nonempty subset X_0 that satisfies ϕ, then by Property 3 of Lemma 3, every user in X_0 satisfies ϕ; thus X must contain a user that satisfies ϕ. Therefore, to determine whether X is safe with respect to ϕ, one can, for each user in X, check whether the user satisfies ϕ. From [4], checking whether one user satisfies a term using only operators in $\{\neg, +, \sqcup, \sqcap\}$ can be done in $\mathbf{P}$. □

Lemma 6. SAFE $\langle \neg, +, \otimes \rangle$ is in $\mathbf{P}$.

Proof. Given a term ϕ that uses only the operator $\otimes$, we show that determining whether a userset X is safe with respect to ϕ under a configuration $\langle U, UR \rangle$ can be reduced to the maximum matching problem on bipartite graphs, which can be solved in $O(MN)$ time, where M is the number of edges and N is the number of nodes in G [6].

Let s be the number of 1CF terms in ϕ and $t = |X|$. Since $\otimes$ is associative [4], ϕ can be equivalently expressed as $(\phi_1 \otimes \phi_2 \otimes \cdots \otimes \phi_s)$, where each ϕ_i is a 1CF term . Let $X = \{u_1, \cdots, u_t\}$. We construct a bipartite graph $G(V_1 \cup V_2, E)$, where each node in V_1 corresponds to a 1CF term in ϕ and each node in V_2 corresponds to a user in X. More precisely, $V_1 = \{a_1, \cdots, a_s\}$, $V_2 = \{b_1, \cdots, b_t\}$, and $(a_i, b_j) \in E$ if and only if $\{u_j\}$ satisfies ϕ_i. The resulting graph G has $s + t$ nodes and $O(st)$ edges, and can be constructed in time polynomial in the size of G. Solving the maximal matching problem for G takes time $O((s + t)st)$.

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

Proving the $\mathbf{NP}$-completeness results in Figure 1. It suffices to prove that the general case $\mathsf{SAFE}\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP}$ and that the four cases $\mathsf{SAFE}\langle \sqcap, \odot \rangle$, $\mathsf{SAFE}\langle \sqcup, \otimes \rangle$, $\mathsf{SAFE}\langle \sqcap, \otimes \rangle$, and $\mathsf{SAFE}\langle \odot, \otimes \rangle$ are $\mathbf{NP}$-hard. Below we state lemmas that establish these results. The proofs to these lemmas that are not included in this section are given in Appendix B. For each $\mathbf{NP}$-hardness result, we discuss the $\mathbf{NP}$-complete problem used in the reduction.

Lemma 7. $\mathsf{SAFE}\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP}$.

Proof. To determine whether a userset U is safe with respect to a term ϕ under a configuration $\langle U, UR \rangle$, we first compute the syntax tree T of ϕ. When constructing T, a 1CF term 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 U is safe with respect to ϕ, then for each node in the tree, there exists a subset of U that satisfies the term rooted at that node, and the root of T corresponds to a subset of U. After these subsets are guessed and labeled with each node, verifying that they indeed satisfy the terms can be done efficiently. From Lemma 3, verifying that a userset satisfies a 1CF term is in $\mathbf{P}$. 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 $\mathbf{NP}$. $\square$

In the following, $(\mathsf{op}_k \phi)$ 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 $(r_1 \ \mathsf{op} \cdots \mathsf{op} \ r_m)$.

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

We use a reduction from the $\mathbf{NP}$-complete SET COVERING problem [2]. The term we constructed for reduction has the form $((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i))$, where r_i is a role.

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

We use a reduction from the $\mathbf{NP}$-complete DOMATIC NUMBER problem [2]. The term we constructed for reduction has the form $(\bigotimes_k (\bigodot_{i=1}^{n} r_i))$, where r_i is a role.

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

We use a reduction from the $\mathbf{NP}$-complete SET PACKING problem [2]. The term we constructed for reduction has the form $(\bigotimes_k (\bigsqcup_{i=1}^{m} (\bigotimes R_j)))$, where R_j is a set of roles.

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

We use a reduction from the NP-complete SET COVERING problem. The term we constructed for reduction has the form $(\bigsqcap_{i=1}^{n} (r_i \otimes (\bigotimes_{k-1} \mathsf{All})))$, where r_i is a role.

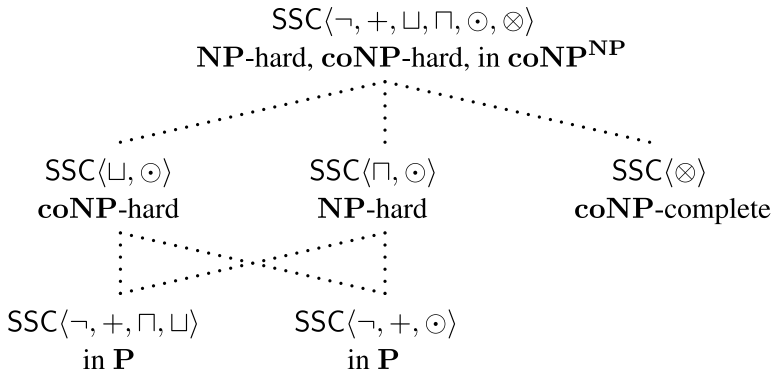

Figure 2: Various sub-cases of the Static Safety Checking (SSC) problem and the corresponding time-complexity. Time-complexity of other subcases can be implied from the subcases shown in the fi gure.

4 Computational Complexity of SSC

In this section, we study the computational complexity of SSC, which determines whether a state is safe with respect to a static safety policy. We will show that SSC in the most general case (i.e., the policy uses an arbitrary term in which all operators are allowed) is both $\mathbf{NP}$-hard and $\mathbf{coNP}$-hard, but it is in polynomial hierarchy $\mathbf{coNP^{NP}}$. A brief introduction on polynomial hierarchy can be found in Appendix A. Similar to the discussion of SAFE in Section 3.2, we consider all subcases where only some subset of the operators in $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed.

Theorem 12. The computational complexities for SSC and its subcases are given in Figure 2.

In the following, we prove that SSC is in $\mathbf{coNP^{NP}}$. The proofs to those intractable cases in Figure 2 are given in Appendix C. In Section 4.1, we identify a class of syntactically restricted terms such that SSC for policies using these syntactically restricted terms is tractable. The class of syntactically restricted terms subsumes both cases listed as in $\mathbf{P}$ in Figure 2.

Lemma 13. $\mathsf{SSC}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{coNP^{NP}}$.

Proof. We show that the complement of $\mathsf{SSC}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{NP^{NP}}$. Because SAFE is in $\mathbf{NP}$ (see Figure 1), an $\mathbf{NP}$ oracle can decide whether a userset is safe with respect to a term. We construct a nondeterministic Oracle Turing Machine M that accepts an input consisting of a state $\langle U, UR, UP\rangle$ and a policy $\mathsf{sp}\langle P, \phi\rangle$ if and only if $\langle U, UR, UP\rangle$ is not safe with respect to $\mathsf{sp}\langle P, \phi\rangle$. M nondeterministically selects a set U of users in $\langle U, UR, UP\rangle$. If U does not cover P, then M rejects. Otherwise, M involves the $\mathbf{NP}$ oracle to check whether U is safe with respect to ϕ. If the oracle answers 'yes", then M rejects; otherwise, M accepts, as it has found a userset that covers P but is not safe with respect to ϕ, which violates the static safety policy. The construction of M shows that the complement of $\mathsf{SSC}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{NP^{NP}}$. Hence, $\mathsf{SSC}\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in $\mathbf{coNP^{NP}}$. □

Lemma 14. $\mathsf{SSC}\langle\sqcup, \odot\rangle$ is $\mathbf{coNP}$-hard.

We reduce the $\mathbf{coNP}$-complete VALIDITY problem for propositional logic to $\mathsf{SSC}\langle\sqcup, \odot\rangle$.

Lemma 15. $\mathsf{SSC}\langle\sqcap, \odot\rangle$ is $\mathbf{NP}$-hard.

Proof. There is a straightforward reduction from $\mathsf{SAFE}\langle\sqcap,\odot\rangle$ to $\mathsf{SSC}\langle\sqcap,\odot\rangle$. Given a term ϕ using only operators $\sqcap$ or $\odot$, in order to check whether a userset X is safe with respect to ϕ, we can construct a policy $\mathsf{sp}\langle P,\phi\rangle$ and a state $\langle U, UR, UP\rangle$ such that X is the only set of users in the state that covers P. In this case, X is safe with respect to ϕ if and only if the state we constructed satisfies $\mathsf{sp}\langle P,\phi\rangle$. Since $\mathsf{SAFE}\langle\sqcap,\odot\rangle$ is $\mathbf{NP}$-hard (see Figure 1), $\mathsf{SSC}\langle\sqcap,\odot\rangle$ is $\mathbf{NP}$-hard. □

Remind that a reduction from the $\mathbf{NP}$-complete SET COVERING problem is used to prove that $\mathsf{SSC}\langle\sqcap,\odot\rangle$ is $\mathbf{NP}$-hard. The term we constructed for the reduction has the form $((\bigodot_{i=1}^{m}\phi_i)\sqcap(\bigodot_{j=1}^{n}\phi'_j))$. Such information on term construction will be useful in Section 4.1.

Lemma 16. $\mathsf{SSC}\langle\otimes\rangle$ is $\mathbf{coNP}$-hard.

We reduce the $\mathbf{NP}$-complete SET COVERING problem to the complement of $\mathsf{SSC}\langle\otimes\rangle$.

4.1 The Most General Tractable Form

From Figure 2, when the operator $\otimes$ is used or when the operator $\odot$ is used in conjunction with any other binary operator, SSC is intractable in general. In this section, we show that if the term in a static safety policy satisfies certain syntactic restriction, then even if all operators except $\otimes$ appear in the term, one can still efficiently determine whether a state satisfies the policy. Furthermore, we show that the syntactic restriction presented in this section allows the most general form of terms such that SSC is tractable with these terms.

Definition 11 (Syntactically Restricted Forms of Terms)**.** The syntactically restricted forms of terms are defined as follows:

- A term is in *level-1 syntactically restricted form* (called a *1RF term*) if it is t or t^+, where t is a unit term. Recall that a unit term can use operators $\neg$, $\sqcup$ and $\sqcap$.
- A term is in *level-2 syntactically restricted form* (called a *2RF term*) if it consists of one or more sub-terms that are 1RF terms, and (when there are more than one such sub-terms) these sub-terms are connected only by operators in the set $\{\sqcup,\sqcap\}$.
- A term is in *level-3 syntactically restricted form* (called a *3RF term*) if it consists of one or more sub-terms that are 2RF terms, and these sub-terms are connected only by operator $\odot$.

We say that a term is *in syntactically restricted form* if it is in level-3 syntactically restricted form. Observe that any term that is in level-i syntactically restricted form is also in level-$(i+1)$ syntactically restricted form for any $i = 1$ or 2.

Theorem 17. Given an access control state $\langle U, UR, UP\rangle$ and a static safety policy $\mathsf{sp}\langle P,\phi\rangle$ where ϕ is in syntactically restricted form, checking whether $\langle U, UR, UP\rangle$ satisfies ϕ can be done in polynomial time.

Proof. Let $\phi = (\phi_1\odot\cdots\odot\phi_m)$ be a 3RF term, where ϕ_i $(1\le i\le m)$ is a 2RF term. The following algorithm checks whether a state $\langle U, UR, UP\rangle$ satisfies a policy $\mathsf{sp}\langle P,\phi\rangle$, where $P=\{p_1,\cdots,p_n\}$.

```
isSafe(P, φ, UR , UP)
begin
   Γ = {φ1, ··· φm};
```

```
    For every $p_i$ in $\{p_1, \cdots, p_n\}$ do
        $G_{p_i}$ = $\emptyset$;
        For every $u$ ∈ $\mathcal{U}$ such that $(u, p_i) \in UP$ do
            $G_{p_i}$ = $G_{p_i}$ ∪ { $\phi_i$ ∈ $\phi$ | $\{u\}$ does not satisfy $\phi_i$};
        EndFor;
        Γ = Γ ∩ $G_{p_i}$;
    EndFor;
    if (Γ == ∅) return true;
    else return false;
end
```

In the above algorithm, G_{p_i} stores the set of 2RF sub-terms in ϕ such that there exists a user u having p_i but $\{u\}$ does not satisfy the sub-term. At the end of the algorithm, on the one hand, if Γ contains a sub-term ϕ_i, it means that for every permissions p_j in $\{p_1, \cdots, p_n\}$, there exists a user u_{p_j} such that u_{p_j} has permission p_j but $\{u_{p_j}\}$ does not satisfy ϕ_i. Furthermore, from Property 3 of Lemma 3, the fact that $\{u_{p_j}\}$ does not satisfy ϕ_i implies that any superset of $\{u_{p_j}\}$ does not satisfy ϕ_i. (Note that 2RF terms use only the operators $\neg, +, \sqcup, \sqcap$.) Therefore, users in $\{u_{p_1}, \cdots, u_{p_n}\}$ together have all permissions in $\{p_1, \cdots, p_n\}$ but does not contain a subset that satisfies ϕ_i, and hence does not contain a subset that satisfies ϕ. The state is not safe. On the other hand, $\Gamma = \emptyset$ indicates that if U covers permissions in $\{p_1, \cdots, p_n\}$, then for every sub-term ϕ_i, there exists $u \in U$ such that $\{u\}$ satisfies ϕ_i. In other words, there exists $U' \subseteq U$ such that U' satisfies ϕ'. The state is safe.

The worst-case time complexity of the above algorithm is $O(m \times |U| \times T)$, where T is the time taken to check whether a singleton satisfies a 1RF term, which is known to be in $\mathbf{P}$ [4]. □

Finally, we would like to show that level-3 syntactically restricted form is the most general syntactic form of terms that keeps SSC tractable. Fist of all, from Lemma 16, if $\otimes$ is allowed, SSC becomes intractable. Furthermore, from the proof of Lemma 15, if $\sqcap$ is allowed to connect sub-terms containing $\odot$, SSC becomes intractable. Finally, in the proof of Lemma 14, the **coNP**-complete validity problem is reduced to SSC$\langle\sqcup, \odot\rangle$. Since checking validity for propositional logic formula in disjunct normal form (DNF) remains **coNP**-complete, SSC is intractable when $\sqcup$ is allowed to connect sub-terms containing $\odot$. In summary, to make SSC tractable, operator $\otimes$ cannot be used, and if $\odot$ is used, it must appear "outside of" $\sqcup$ and $\sqcap$. Such a restriction is precisely captured by the level-3 syntactically restricted form.

5 An Algorithm for SSC

Despite the fact that SSC is intractable in general, it is still possible that many instances encountered in practice are efficiently solvable. In order to study the efficiency of solving SSC, we have designed and implemented an algorithm, which is described in detail in this section.

5.1 Description of the Algorithm

To determine whether $\langle U, UR, UP\rangle$ is safe with respect to $\mathsf{sp}\langle P, \phi\rangle$, a straightforward algorithm is to enumerate all usersets that cover P and for every such userset, check whether it has a subset that satisfies ϕ. If the answer is "no" for any such userset, then we know that $\langle U, UR, UP\rangle$ is not safe with respect to $\mathsf{sp}\langle P, \phi\rangle$. Otherwise, $\langle U, UR, UP\rangle$ is safe. Our algorithm is based on this idea but has a number of improvements that greatly reduces the running time. Here is a summary of the improvement techniques in our algorithm on determining whether $\langle U, UR, UP\rangle$ is safe with respect to $\mathsf{sp}\langle P, \phi\rangle$.

- We preprocess the input and eliminates information in $\langle U, UR, UP\rangle$ that is irrelevant to the result of static safety checking with respect to $\mathsf{sp}\langle P, \phi\rangle$.
- Only minimal usersets that cover P will be checked for userset-term safety.
- We define a partial-order over sets of roles and perform static pruning to reduce the number of users that need to be considered based on the partial-order over their role membership.
- We propose an abstract representation of sets which enables us to design an efficient bottom-up approach for determining userset-term safety.

In the rest of this section, for simplicity of discussion, the keyword All and user names in a term of the algebra are also treated as roles. For instance, we may treat the atomic term $Alice$ as a role such that user $Alice$ is the only member of the role, while All is treated as a role such that everybody in the system is its member.

Preprocessing Given a state $\langle U, UR, UP\rangle$ and a policy $\mathsf{sp}\langle P, \phi\rangle$, we first remove all pairs (u, p) from UP if $p \notin P$, and all pairs (u, r) from UR if r does not appear in ϕ. We also remove all users u from U if u does not have any permission in P.

Furthermore, we rewrite the term ϕ into an equivalent term where $\neg$ (if any) only applies to atomic term. Such a rewriting is always possible, as the operators $\neg, \sqcup$ and $\sqcap$ satisfy the DeMorgan's Law. (See [4] for algebraic properties of the operators.) This will be useful in static pruning, which will be discussed later.

Minimal Usersets Only Given a policy $\mathsf{sp}\langle P, \phi\rangle$, let X be a userset that covers P. It is clear that a superset of X covers P as well. If X is safe with respect to ϕ, then any superset of X is safe with respect to ϕ, but not the other way around. Therefore, when considering whether the state satisfies $\mathsf{sp}\langle P, \phi\rangle$, we may consider X without considering the supersets of X. In other words, we check whether X satisfies ϕ if and only if X covers P and there does not exist $X' \subset X$ such that X' covers P, and such a userset X is called a *minimal userset* that covers P.

Static Pruning The number of all usersets in $\mathcal{U}$ is 2^n, where $|\mathcal{U}| = n$. But it is clear that not all these subsets need to be considered. In particular, we are only interested in those minimal usersets that cover all permissions in the policy. In the following, we describe a static pruning technique that aims at reducing the number of users that need to be taken into account. Intuitively, given a policy $\mathsf{sp}\langle P, \phi\rangle$, we try to ignore those users who have a relatively small number of permissions in P but satisfy many sub-terms in ϕ.

Definition 12 (Positive and Negative Dependance)**.** We say that a term ϕ *positively* (or *negatively*) depends on role r, if ϕ contains r (or $\neg r$). R_{pos} and R_{neg} denote the set of roles that ϕ positively and negatively depends on, respectively.

For instance, if $\phi = (\texttt{Accountant} \odot \texttt{Clerk}) \cup (\neg\texttt{Manager} \sqcap \neg\texttt{Clerk})$, then $R_{pos} = \{\texttt{Accountant}, \texttt{Clerk}\}$ and $R_{neg} = \{\texttt{Manager}, \texttt{Clerk}\}$. Note that Clerk appears in both R_{pos} and R_{neg}. Definition 13 defines a partial relation between role sets with respect to a term, and Lemma 18 states a condition on which a user may be ignored without affecting the soundness of static safety checking.

Definition 13 (Partial-Order $\preceq_\phi$)**.** Given a term ϕ and two sets of roles R_a and R_b, we have $R_a \preceq_\phi R_b$ (or equivalently $R_b \succeq_\phi R_a$) if and only if $R_a \cap R_{pos} \subseteq R_b \cap R_{pos}$ and $R_a \cap R_{neg} \supseteq R_b \cap R_{neg}$.

Note that the relation $\preceq_\phi$ is transitive, i.e. if $R_1 \preceq_\phi R_2$ and $R_2 \preceq_\phi R_3$, then $R_1 \preceq_\phi R_3$.

Lemma 18. Given a policy $\mathsf{sp}\langle P, \phi\rangle$, a state $\langle U, UR, UP\rangle$ and two users u_1, u_2 ($u_1 \neq u_2$), let P_i and R_i be the set of permissions and roles of u_i ($i = 1$ or 2). If $(P_1 \cap P) \supseteq (P_2 \cap P)$ and $R_1 \preceq_\phi R_2$, then $\langle U, UR, UP\rangle$ is safe with respect to $\mathsf{sp}\langle P, \phi\rangle$ if and only if $\langle U/\{u_2\}, UR, UP\rangle$ is safe with respect to $\mathsf{sp}\langle P, \phi\rangle$. In other words, u_2 may be ignored without affecting the soundness of static safety checking.

Proof. Let X be a userset covering P. In the following, we prove that if $u_2 \in X$, we can always find another userset X' ($u_2 \notin X'$) that covers P, and X is safe with respect to ϕ only if X' is safe with respect to ϕ. Hence, we may consider X' and ignore X, which indicates that u_2 may be ignored without affecting the soundness of static safety checking.

On the one hand, assume that both u_1 and u_2 are in X. Since $(P_1 \cap P) \supseteq (P_2 \cap P)$, $X' = X/\{u_2\}$ still covers P and $X' \subset X$. Hence, if X' is safe with respect to ϕ, so is X.

On the other hand, assume that $u_2 \in X$ but $u_1 \notin X$. Let $X' = (X/\{u_2\}) \cup u_1$. X covering P and $(P_1 \cap P) \supseteq (P_2 \cap P)$ imply that X' covers P. We would like to show that if X' is safe with respect to ϕ, then so is X. Assume that X' contains a subset X'_1 that satisfies ϕ. We are only interested in the case where $u_1 \in X'_1$. By definition of term satisfaction, X'_1 satisfying ϕ indicates that $\{u_1\}$ is used to satisfy a set of atomic terms and/or negation of atomic terms in ϕ. (Note that $\neg$ is only applied to atomic terms in ϕ after preprocessing.) Let $\{\gamma_1, \cdots, \gamma_m\}$ ($m \geq 1$) be a set of atomic terms or negation of atomic terms in ϕ such that $\{u_1\}$ satisfies γ_i ($1 \leq i \leq m$). If $\gamma_i = r$, then u_1 must be a member of role r, which means that $r \in R_1$. $R_1 \preceq_\phi R_2$ indicates that $r \in R_2$. Otherwise, if $\gamma_i = \neg r$, then $r \notin R_1$. $R_1 \preceq_\phi R_2$ indicates that $r \notin R_2$. In either case, $\{u_2\}$ satisfies γ_i. In general, $\{u_2\}$ satisfies all elements in $\{\gamma_1, \cdots, \gamma_m\}$. Therefore, $X = (X'/\{u_1\}) \cup \{u_2\}$ satisfies ϕ. In generale, we may only consider X' without considering X. $\square$

The above lemma may greatly reduce the number of users we need to considered. In particular, if multiple users have the same set of permissions in P and roles in ϕ, then at most one of these users need to be taken into account.

The following example illustrates how static punning works.

Example 1. Given a policy $\mathsf{sp}\langle\{p_1, p_2, p_3\}, (r_1 \odot \neg r_2)\rangle$ and a state $\langle U, UR, UP\rangle$, we have

$$U = \{Alice, Bob, Carl, Doris, Elaine\}$$
$$UP = \{(Alice, p_1), (Alice, p_2), (Bob, p_1), (Carl, p_1), (Carl, p_2), (Doris, p_3), (Elaine, p_3), (Elaine, p_4)\}$$
$$UR = \{(Alice, r_1), (Bob, r_1), (Bob, r_3), (Carl, r_1), (Carl, r_2)\}$$

There are five users in the system altogether. However, according to Lemma 18, we only need to consider two users $Carl$ and $Doris$. First of all, Bob may be ignored as he has the same set of roles in $\{r_1, r_2\}$ as $Alice$, but his set of permissions is subsumed by $Alice$'s. Secondly, $Alice$ does not need to be considered as she has the same set of permissions as $Carl$, but $R_{Carl} \preceq_{(r_1 \odot \neg r_2)} R_{Alice}$. Finally, since $Doris$ and $Elaine$ have the same permissions and roles with respect to the given policy, only one of them should be taken into account.

Determining Term Safety In [4], Li and Wang described an algorithm for the Userset-Term Satisfaction (UTS) problem. Their algorithm employs both a top-down approach and a bottom-up approach based on the syntax tree of the term. In the top-down approach, one starts with the root of the syntax tree and the given userset and tries to split the userset into subsets so as to satisfy different sub-terms. The processing is then performed recursively on those subsets and sub-terms. In the bottom-up processing, one starts with unit terms. For each unit term, one calculates all subsets of the given userset that satisfy the term. One then goes bottom-up to calculate that for each node in the syntax tree. We call the set of usersets that satisfy a term the *satisfaction set* of the term. An example of bottom-up processing of a term in a given configuration is given in Figure 3.

As to SSC, instead of determining whether a userset satisfies a term, we are only interested in whether there exists a subset of userset X that satisfies the term. In this case, using a pure bottom-up design should be more efficient than a combination of top-down and bottom-up processing.

A major challenge for bottom-up processing is that the number of subsets that satisfy a sub-term may be very large, especially when $+$ is used. The algorithm for UTS in [4] stops performing bottom-up processing when $+$ is encountered, as the sub-term t^+ can be satisfied by $2^{|Y|} - 1$ usersets, where t is a unit term and $Y = \{u \in U \mid \{u\} \text{ satisfies } t\}$.

In our algorithm for SSC, we introduce a novel abstract representation of sets, which greatly reduces the number of elements generated during the computation. Intuitively, an abstract set is a set of sets and is represented as a pair of two disjoint sets, the *explicit-element set (*EES*)* and the *possible-element set (*PES*)*, where *EES* contains elements that must appear and *PES* contains elements that may or may not appear. For example, an abstract userset $\langle \texttt{ees}\{Alice\} :: \texttt{pes}\{Bob, Carl\}\rangle$ indicates that $Alice$ appears in the set for sure, while Bob and $Carl$ may be included in the set as well. In other words, $\langle \texttt{ees}\{Alice\} :: \texttt{pes}\{Bob, Carl\}\rangle$ is a set of four different usersets, $\{Alice\}$, $\{Alice, Bob\}$, $\{Alice, Carl\}$ and $\{Alice, Bob, Carl\}$.

Definition 14 (Abstract Set)**.** An *abstract set* is given as a pair $\Psi = \langle \texttt{ees}\{a_1, \cdots, a_m\} :: \texttt{pes}\{b_1, \cdots, b_n\}\rangle$ $(m \geq 1, n \geq 0)$, which stands for a set of sets. $\Psi.ees = \{a_1, \cdots, a_m\}$ is the explicit-element set of Ψ and $\Psi.pes = \{b_1, \cdots, b_n\}$ is the possible-element set of Ψ. A set S is in Ψ if and only if $\{a_1, \cdots, a_m\} \subseteq S \subseteq \{a_1, \cdots, a_m\} \cup \{b_1, \cdots, b_n\}$.

Abstract sets are especially useful in representing satisfaction sets of terms containing sub-terms in the form of t^+. For example, assume that $Alice$, Bob and $Carl$ are members of role r. The set of usersets that satisfy r^+ may be represented as $\{\langle \texttt{ees}\{Alice\} :: \texttt{pes}\{Bob, Carl\}\rangle, \langle \texttt{ees}\{Bob\} :: \texttt{pes}\{Carl\}\rangle, \langle \texttt{ees}\{Carl\} :: \texttt{pes}\{\}\rangle\}$. In general, $|Y|$ rather than $2^{|Y|} - 1$ usersets are stored for t^+, where t is a unit term and $Y = \{u \in U \mid \{u\} \text{ satisfies } t\}$.

Our bottom-up approach employs abstract sets and involves performing set operations over abstract sets. The description of our bottom-up approach is given in Lemma 19.

Lemma 19. Given a userset X and a term ϕ, the satisfaction set Ψ_ϕ of ϕ can be computed as follows. Initially, $\Psi_\phi = \emptyset$.

- $\phi = r$: For every $u \in (X \wedge X_r)$, where X_r is the set members of r, $\Psi_\phi \leftarrow \Psi_\phi \cup \{\langle \texttt{ees}\{u\} :: \texttt{pes}\{\}\rangle\}$.
- $\phi = \neg\phi_1$: For every $u \in X$, if $\nexists_{\alpha \in \Psi_{\phi_1}}(\{u\} = \alpha.ees)$, $\Psi_\phi \leftarrow \Psi_\phi \cup \{\langle \texttt{ees}\{u\} :: \texttt{pes}\{\}\rangle\}$.
- $\phi = \phi_1^+$: Let $X_s = \{u \mid \exists_{\alpha \in \Psi_{\phi_1}}(\alpha.ees = \{u\})\} = \{u_{a_1} \cdots u_{a_m}\}$, where a_i $(i \in [1, m])$ is an integer and $a_i < a_j$ when $i < j$. For every $i \in [1, m]$, $\Psi_\phi \leftarrow \Psi_\phi \cup \{\langle \texttt{ees}\{u_{a_i}\} :: \texttt{pes}\{u_{a_{i+1}}, \cdots, u_{a_m}\}\rangle\}$.
- $\phi = \phi_1 \sqcap \phi_2$: For every $\alpha \in \Psi_{\phi_1}$ and every $\beta \in \Psi_{\phi_2}$, if $\alpha.ees \subseteq \beta.ees \cup \beta.pes$ and $\beta.ees \subseteq \alpha.ees \cup \alpha.pes$, then $\Psi_\phi \leftarrow \Psi_\phi \cup \{\langle \texttt{ees}\{\alpha.ees \cup \beta.ees\} :: \texttt{pes}\{\alpha.pes \cap \beta.pes\}\rangle\}$.
- $\phi = \phi_1 \sqcup \phi_2$: $\Psi_\phi \leftarrow \Psi_{\phi_1} \cup \Psi_{\phi_2}$.
- $\phi = \phi_1 \odot \phi_2$: For every $\alpha \in \Psi_{\phi_1}$ and every $\beta \in \Psi_{\phi_2}$, $\Psi_\phi \leftarrow \Psi_\phi \cup \{\langle \texttt{ees}\{E\} :: \texttt{pes}\{P - E\}\rangle\}$, where $E = \alpha.ees \cup \beta.ees$ and $P = \alpha.pes \cup \beta.pes$.

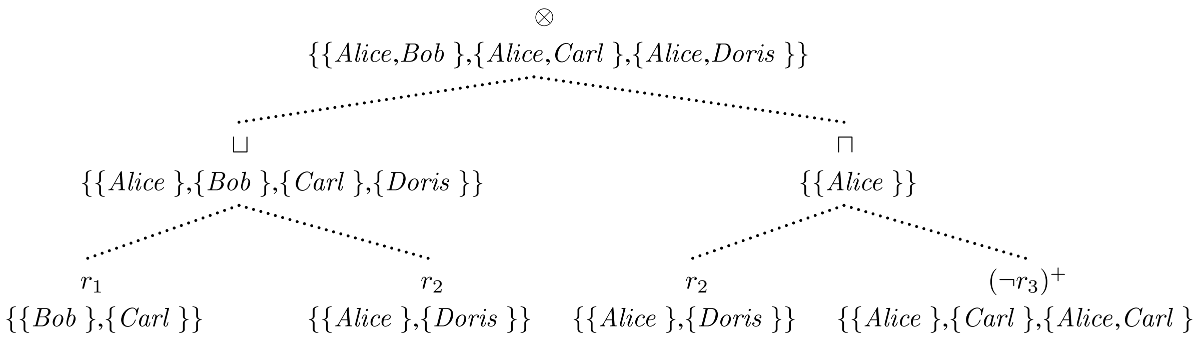

Figure 3: An example of the bottom-up process proposed in [4]. Let $\phi = ((r_1 \sqcup r_2) \otimes (r_2 \sqcap (\neg r_3)^+))$. In confi guration $\langle U, UR \rangle$, $UR = \{(Alice, r_2), (Bob, r_1), (Bob, r_3), (Carl, r_1), (Doris, r_2), (Doris, r_3)\}$. For each sub-term of ϕ, the subsets of $\{Alice, Bob, Carl, Doris\}$ that satisfi es that sub-term is displayed.

- $\phi = \phi_1 \otimes \phi_2$: For every $\alpha \in \Psi_{\phi_1}$ and every $\beta \in \Psi_{\phi_2}$, if $\alpha.ees \cap \beta.ees = \emptyset$, then $\Psi_\phi \leftarrow \Psi_\phi \cup \{\langle \texttt{ees}\{E\} :: \texttt{pes}\{P - E\}\rangle\}$, where $E = \alpha.ees \cup \beta.ees$ and $P = \alpha.pes \cup \beta.pes$.

The proof of correctness of our bottom-up approach can be found in Appendix D.

Besides making use of abstract sets to represent satisfaction sets of terms, an additional technique is used to further accelerate the bottom-up processing. Given a term ϕ, we are only interested in whether the satisfaction set of ϕ is empty or not. To acquire such information, it is sometimes unnecessary to explicitly compute the satisfaction set for every sub-term of ϕ. In particular, if the satisfaction sets of both ϕ_1 and ϕ_2 are not empty, then the satisfaction sets of ϕ_1^+, $\phi_1 \cup \phi_2$ and $\phi_1 \odot \phi_2$ are not empty; if either of the satisfaction sets of ϕ_1 and ϕ_2 is not empty, then the satisfaction set of $\phi_1 \sqcup \phi_2$ is not empty. Hence, we need to compute the exact satisfaction set for a sub-term only if it is an atomic term or the path from the node corresponding to the sub-term to the root of the syntax tree contains operators $\neg$, $\sqcap$ or $\otimes$. For all other sub-terms, we just need to mark whether the satisfaction set is empty or not. For example, given term $(r_1 \otimes r_2) \odot (r_3 \sqcup \neg r_4)$, we just need to explicitly compute the satisfaction sets for sub-terms $(r_1 \otimes r_2)$, r_3 and $\neg r_4$.

5.2 Implementation and Evaluation

We prototyped the algorithm described in Section 5.1 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. The parameters we used in our experiments are chosen to be close to practical cases. In particular, the number of permissions involved in a task will not be very large and the term used in the policy will not be very complicated. However, the number of users in the system may be large.

Some of our experimental results are presented in Table 1. As we can see in Table 1, our algorithm solves SSC effi ciently when the number of users is small. The algorithm does not scale very well when the number of users grows. However, it is still capable to solve SSC instances with nontrivial size in a relatively short time. As SSC needs to be performed only when the access control state of the system changes, which is not expected to happen frequently, relative slow running time may be acceptable in some situations. Further research is needed on improving the performance of the algorithm and on assessing whether solving SSC is practical in real-world scenarios.

Policy	Size of P	Users	UR Size	UP Size	Safe?	Runtime
$\mathsf{sp}\langle P, ((r1^+ \odot r2) \otimes \neg r3) \odot (r1 \sqcap r4^+)\rangle$	5	10	18	15	Yes	47 ms
$\mathsf{sp}\langle P, ((r1^+ \odot r2) \otimes \neg r3) \odot (r1 \sqcap r4^+)\rangle$	10	10	18	30	Yes	1.0 s
$\mathsf{sp}\langle P, ((r1^+ \odot r2) \otimes \neg r3) \odot (r1 \sqcap r4^+)\rangle$	10	20	34	46	Yes	5.8 s
$\mathsf{sp}\langle P, ((r1^+ \odot r2) \otimes \neg r3) \odot (r1 \sqcap r4^+)\rangle$	10	40	65	82	Yes	97.7 s
$\mathsf{sp}\langle P, ((r1^+ \odot r2) \otimes \neg r3) \odot (r1 \sqcap r4^+)\rangle$	10	40	65	84	No	5.7 s

Table 1: A table that shows the runtime of testing whether a state is safe with respect to a static safety policy.

6 Related Work

The concept of SoD has long existed in the physical world, sometimes under the name "the two-man rule" 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 [7] under the name "separation of privilege." Clark and Wilson's commercial security policy for integrity [1] identified SoD along with well-formed transactions as two major mechanisms of fraud and error control. There exists a wealth of literatures [5, 8, 9, 3] on the enforcement of SoD policies. Nash and Poland [5] 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 [8, 9] presented Transaction Control Expressions, a history-based mechanism for dynamically enforcing SoD policies. A transaction control expression associates each step in the transaction with a role. By default, the requirement is such that each step must be performed by a different user. One can also specify that two steps must be performed by the same user. In Transaction Control Expressions, user qualification requirements are associated with individual steps in a transaction, rather than a transaction as a whole.

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

Our paper studies enforcement of policies specified in the algebra introduced by Li and Wang [4]. They mentioned static enforcement and dynamic enforcement as two possible enforcement mechanisms for high-level security policies specified in the algebra, but they did not investigate enforcement in detail.

7 Conclusion

In this paper, we formally define and study direct static enforcement of high-level security policies specified in the algebra proposed by Li and Wang [4]. We give comprehensive computational complexity results for solving the Static Safety Checking problem and the related Userset-Term Safety problem. We also propose a syntactically restricted form of terms such that if the term in a policy satisfies the syntactic restriction, the direct enforcement of the policy is tractable. Finally, we design and evaluate an algorithm to solve the static safety checking problem for high-level security policies.

In the future, we plan to study other enforcement approaches for policies specified in the algebra, including indirect static enforcement, which uses constraints to rule out unsafe states, and dynamic enforcement, which enforces the policy using history for each instance of a sensitive task.

References

[1] 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.

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

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

[4] N. Li and Q. Wang. Beyond separation of duty: An algebra for specifying high-level security policies. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, Nov. 2006.

[5] 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.

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

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

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

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

A Background on Oracle Turing Machines and Polynomial Hierarchy

Oracle Turing Machines 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

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

B Proof of Theorem 2

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 8: $\mathsf{SAFE}\langle\sqcap, \odot\rangle$ is $\mathbf{NP}$-hard.

Proof. We use a reduction from the $\mathbf{NP}$-complete SET COVERING problem [2]. 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 $\langle U, \mathit{UR}\rangle$ 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 is safe with respect to ϕ under $\langle U, \mathit{UR}\rangle$ if and only if there are no more than k sets in family F whose union is S.

If U is safe with respect to ϕ, by definition, a subset U' of U satisfies $(\bigodot_k \mathsf{All})$ and $(\bigodot_{i=1}^{n} r_i)$. U' satisfying $(\bigodot_k \mathsf{All})$ indicates that $|U'| \leq k$, while U' satisfying $(\bigodot_{i=1}^{n} r_i)$ indicates that users in U' together have membership of r_i for every $i \in [1, n]$. Without loss of generality, suppose $U' = \{u_1, \cdots, u_t\}$, where $t \leq k$. Since $(u_i, r_j) \in \mathit{UR}$ 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 $i \in [1, n]$, which indicates that $\{u_1, \cdots, u_k\}$ is safe with respect to $(\bigodot_{i=1}^{n} r_i)$. Also, any non-empty subset of $\{u_1, \cdots, u_k\}$ satisfies $(\bigodot_k \mathsf{All})$. Hence, U is safe with respect to ϕ. □

Proof of Lemma 9: $\mathsf{SAFE}\langle\odot, \otimes\rangle$ is $\mathbf{NP}$-hard.

Proof. We use a reduction from the $\mathbf{NP}$-complete DOMATIC NUMBER problem [2]. 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 is safe with respect to ϕ if and only if U has a subset U' that can be divided into k pairwise disjoint sets, each of which have role membership of $r_1, r_2, \cdots, r_n$. Therefore, the answer to the Domatic Number problem is "yes" if and only if U is safe with respect to ϕ. $\square$

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

Proof. We use a reduction from the $\mathbf{NP}$-complete SET PACKING problem [2], 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 is safe with respect to ϕ under $\langle U, UR\rangle$ if and only if there are k pairwise disjoint sets in family F.

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

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. Since $U_i \subseteq U$, U is safe with respect to ϕ.

On the other hand, suppose U is safe with respect to ϕ. Then, U has a subset U' that can be divided into k pairwise disjoint sets $\hat{U}_1, \cdots, \hat{U}_k$, such that $\hat{U}_i$ satisfies ϕ_i. In order to satisfy ϕ', $\hat{U}_i$ must satisfy a certain ϕ_{a_i} and hence be equivalent to U_{a_i}. The assumption that $\hat{U}_1, \cdots, \hat{U}_k$ 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". $\square$

Proof of Lemma 11: $\mathsf{SAFE}\,\langle\sqcap, \otimes\rangle$ is $\mathbf{NP}$-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 $\phi = (\bigsqcap_{i=1}^n \left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right))$. We now demonstrate that U satisfies ϕ under $\langle U, UR\rangle$ if and only if there are k sets in family F whose union is S.

If U is safe with respect to ϕ, by definition, a subset U' of U satisfies $\left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right)$ for every i, which means users in U' together have membership of r_i for every $i \in [1, n]$. For any $i \in [1, n]$, U' satisfying $(r_i \otimes (\bigotimes_{k-1} \mathsf{All}))$ indicates that $|U'| = k$. Suppose $U' = \{u_{a_1}, \cdots, u_{a_k}\}$. 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 $i \in [1, n]$, which indicates that $\{u_1, \cdots, u_k\}$ satisfies ϕ_i for every $i \in [1, n]$. Hence, $\{u_1, \cdots, u_k\}$ satisfies ϕ and U is safe with respect to ϕ. $\square$

C Proof of Theorem 13

Proof of Lemma 14: $\mathsf{SSC}\langle\sqcup, \odot\rangle$ is $\mathbf{coNP}$-hard.

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

We create a state $\langle U, UR, UP\rangle$ with n permissions $p_1, p_2, \cdots, p_n$, $2n$ 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 $UP = \{(u_i, p_i), (u'_i, p_i) \mid 1 \leq i \leq n\}$ and $UR = \{(u_i, r_i), (u'_i, r'_i) \mid 1 \leq i \leq n\}$. We also construct a term ϕ from the formula φ by replacing each literal v_i with r_i, each literal $\neg v_i$ with r'_i, each occurrence of $\wedge$ with $\odot$ and each occurrence of $\vee$ with $\sqcup$.

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

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

Proof of Lemma 16: $\mathsf{SSC}\langle\otimes\rangle$ is $\mathbf{coNP}$-hard.

Proof. We can reduce the $\mathbf{NP}$-complete 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 fi nite set $S = \{e_1, \cdots, e_n\}$ and a budget K, where K is an integer smaller than m and n. 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 state $\langle U, UR, UP\rangle$ such that $UR = \{(u_i, r_i) \mid i \in [1, m]\}$ and $UP = \{(u_i, p_j) \mid e_j \in S_i\}$. Construct a safety policy $\mathsf{sp}\langle P, \phi\rangle$, where $P = \{p_1, \cdots, p_n\}$ and $\phi = (\bigotimes_{K+1} \mathsf{All})$. ϕ is satisfi ed by any set of no less than $K + 1$ users.

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

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

Since the Set Covering problem is $\mathbf{NP}$-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. □

D Proof of Lemma 19

Let S_ϕ be the satisfaction set (in normal representation) of ϕ. Recall that S_ϕ is a set of usersets and an abstract set α is a set of sets. To proof the correctness of our bottom-up approach, we need to show that $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i = S_\phi$. That is to say, for any $s \in S_\phi$, there exists $\alpha \in \Psi_\phi$ such that $s \in \alpha$; and for any $\alpha \in \Psi_\phi$ and any $s \in \alpha$, we have $s \in S_\phi$.

The proofs to the case of $\phi = r$ and $\phi = \neg\phi_1$ are straightforward. In the following, we prove other cases by induction. We assume that the bottom-up approach correctly computes the satisfaction sets of ϕ_1 and ϕ_2.

- $\phi = \phi_1^+$: By Defi nition 3, $S_\phi = P(S_{\phi_1})/\{\}$, where $P(S)$ is the power set of S. Without loss of generality, assume that $S_{\phi_1} = \{u_1, \cdots, u_m\}$. On the one hand, for any $s \in S_\phi$, let $s = \{u_{a_1}, \cdots, u_{a_n}\}$

where $a_i \in [1, m]$, $a_i < a_j$ when $i < j$, and $n \leq m$. According to our bottom-up approach, $\alpha = \langle \mathtt{ees}\{u_{a_1}\} :: \mathtt{pes}\{u_{a_1+1}, \cdots, u_m\}\rangle \in \Psi_\phi$. We have $\alpha.ees \subseteq s$ and $s \subseteq \alpha.ees \cup \alpha.pes$. By Definition 14, we have $s \in \langle \mathtt{ees}\{u_{a_1}\} :: \mathtt{pes}\{u_{a_2}, \cdots, u_{a_n}\}\rangle$. Therefore, $S_\phi \subseteq \bigcup_{\alpha_i \in \Psi_\phi} \alpha_i$. On the other hand, for any $\alpha \in \Psi_\phi$ and any $s \in \alpha$, $s \in \{u_1, \cdots, u_m\}$, which indicates that $s \in P(\{u_1, \cdots, u_m\})/\{\}$. Hence, $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i \subseteq S_\phi$. In general, $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i = S_\phi$.

- $\phi = \phi_1 \sqcap \phi_2$: By Definition 3, $S_\phi = S_{\phi_1} \cap S_{\phi_2}$. We just need to prove that $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i = \bigcup_{\beta_j \in \Psi_{\phi_1}} \beta_j \cap \bigcup_{\gamma_k \in \Psi_{\phi_2}} \gamma_k$.

  On the one hand, according to our bottom-up approach, for any $\alpha \in \Psi_\phi$, there exist $\beta \in \Psi_{\phi_1}$ and $\gamma \in \Psi_{\phi_2}$ such that $\beta.ees \subseteq \gamma.ees \cup \gamma.pes$, $\gamma.ees \subseteq \beta.ees \cup \beta.pes$, $\alpha.ees = \beta.ees \cup \gamma.ees$ and $\alpha.pes = \beta.pes \cap \gamma.pes$. We have $\beta.ees \subseteq \alpha.ees$ and $\alpha.ees \cup \alpha.pes \subseteq \beta.ees \cup \beta.pes$. By Definition 14, for any set $s \in \alpha$, $\alpha.ees \subseteq s$ and $s \subseteq \alpha.ees \cup \alpha.pes$. Hence, we have $\beta.ees \subseteq s$ and $s \subseteq \beta.ees \cup \beta.pes$, which indicates that $s \in \beta$. Since s is picked arbitrarily from α, we have $\alpha \subseteq \beta$. Thus, $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i \subseteq \bigcup_{\beta_j \in \Psi_{\phi_1}} \beta_j$. Similarly, we can prove that $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i \subseteq \bigcup_{\gamma_k \in \Psi_{\phi_2}} \gamma_k$. In general, $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i \subseteq \bigcup_{\beta_j \in \Psi_{\phi_1}} \beta_j \cap \bigcup_{\gamma_k \in \Psi_{\phi_2}} \gamma_k$.

  On the other hand, for any s such that there exist $\beta \in \Psi_{\phi_1}$ and $\gamma \in \Psi_{\phi_2}$ such that $s \in \beta$ and $s \in \gamma$, by Definition 14, we have $\beta.ees \subseteq s$, $\gamma.ees \subseteq s$, $s \subseteq \beta.ees \cup \beta.pes$ and $s \subseteq \gamma.ees \cup \gamma.pes$. Hence, $\beta.ees \cup \gamma.ees \subseteq s$ and $s \subseteq (\beta.ees \cup \beta.pes) \cap (\gamma.ees \cup \gamma.pes) = (\beta.ees \cap \gamma.ees) \cup (\beta.ees \cap \gamma.pes) \cup (\beta.pes \cap \gamma.ees) \cup (\beta.pes \cap \gamma.pes) \subseteq (\beta.ees \cup \gamma.ees) \cup (\beta.pes \cap \gamma.pes)$. In this case, $s \in \langle \mathtt{ees}\{\beta.ees \cup \gamma.ees\} :: \mathtt{pes}\{\beta.pes \cap \gamma.pes\}\rangle$. According to our bottom-up approach, we have $\langle \mathtt{ees}\{\beta.ees \cup \gamma.ees\} :: \mathtt{pes}\{\beta.pes \cap \gamma.pes\}\rangle \in \Psi_\phi$. Hence, $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i \supseteq \bigcup_{\beta_j \in \Psi_{\phi_1}} \beta_j \cap \bigcup_{\gamma_k \in \Psi_{\phi_2}} \gamma_k$.

  In general, $\bigcup_{\alpha_i \in \Psi_\phi} \alpha_i = \bigcup_{\beta_j \in \Psi_{\phi_1}} \beta_j \cap \bigcup_{\gamma_k \in \Psi_{\phi_2}} \gamma_k$.

- $\phi = \phi_1 \sqcup \phi_2$: By Definition 3, $S_\phi = S_{\phi_1} \cup S_{\phi_2}$. The proof is straightforward.

- $\phi = \phi_1 \otimes \phi_2$: We have $S_\phi = \{s_1 \cup s_2 \mid s_1 \in S_{\phi_1} \wedge s_2 \in S_{\phi_2} \wedge s_1 \cap s_2 = \emptyset\}$.

  On the one hand, for any $s \in S_\phi$, there exist $s_1 \in S_{\phi_1}$ and $s_2 \in S_{\phi_2}$ such that $s = s_1 \cup s_2$ and $s_1 \cap s_2 = \emptyset$. By induction assumption, there exist $\alpha_1 \in \Psi_{\phi_1}$ and $\alpha_2 \in \Psi_{\phi_2}$ such that $s_1 \in \alpha_1$ and $s_2 \in \alpha_2$. By Definition 14, we have $\alpha_1.ees \subseteq s_1$, $\alpha_2.ees \subseteq s_2$, $s_1 \subseteq \alpha_1.ees \cup \alpha_1.pes$ and $s_2 \subseteq \alpha_2.ees \cup \alpha_2.pes$. Hence, $\alpha_1.ees \cup \alpha_2.ees \subseteq s_1 \cup s_2$ and $s_1 \cup s_2 \subseteq \alpha_1.ees \cup \alpha_1.pes \cup \alpha_2.ees \cup \alpha_2.pes = (\alpha_1.ees \cup \alpha_2.ees) \cup ((\alpha_1.pes \cup \alpha_2.pes) - (\alpha_1.ees \cup \alpha_2.ees))$, which indicates that $s_1 \cup s_2 \in \langle \mathtt{ees}\{\alpha_1.ees \cup \alpha_2.ees\} :: \mathtt{pes}\{(\alpha_1.pes \cup \alpha_2.pes) - (\alpha_1.ees \cup \alpha_2.ees)\}\rangle$. According to our bottom-up approach, we have $\langle \mathtt{ees}\{\alpha_1.ees \cup \alpha_2.ees\} :: \mathtt{pes}\{(\alpha_1.pes \cup \alpha_2.pes) - (\alpha_1.ees \cup \alpha_2.ees)\}\rangle \in \Psi_\phi$. Therefore, $S_\phi \subseteq \bigcup_{\alpha_i \in \Psi_\phi} \alpha_i$.

  On the other hand, for any $\alpha \in \Psi_\phi$ and any $s \in \alpha$, by Definition 14, we have $\alpha.ees \subseteq s$ and $s \subseteq \alpha.ees \cup \alpha.pes$. According to our bottom-up approach, there exist $\alpha_1 \in \Psi_{\phi_1}$ and $\alpha_2 \in \Psi_{\phi_2}$ such that $\alpha_1.ees \cap \alpha_2.ees = \emptyset$, $\alpha.ees = \alpha_1.ees \cup \alpha_2.ees$ and $\alpha.pes = (\alpha_1.pes \cup \alpha_2.pes) - (\alpha_1.ees \cup \alpha_2.ees)$.

  Let $s_1 = s \cap (\alpha_1.ees \cup (\alpha_1.pes - \alpha_2.ees))$ and $s_2 = s \cap (\alpha_2.ees \cup (\alpha_2.pes - (\alpha_1.ees \cup \alpha_1.pes)))$. We would like to show that $s_1 \cup s_2 \in S_\phi$ and $s_1 \cup s_2 = s$. Since $\alpha_1.ees \subseteq s$, we have $\alpha_1.ees \subseteq s_1$. Furthermore, $\alpha_1.ees \cup (\alpha_1.pes - \alpha_2.ees) \subseteq \alpha_1.ees \cup \alpha_1.pes$ implies that $s_1 \subseteq \alpha_1.ees \cup \alpha_1.pes$. Hence, $s_1 \in \alpha_1$. Similarly, we can prove that $s_2 \in \alpha_2$. But induction assumption, we have $s_1 \in S_{\phi_1}$ and $s_2 \in S_{\phi_2}$. Also, since $\alpha_1.ees \cap \alpha_2.ees = \emptyset$, it can be easily show that $s_1 \cap s_2 = \emptyset$. Therefore,

$s_1 \cup s_2 \in S_\phi$. Finally, we have

$$\begin{aligned} s_1 \cup s_2 &= (s \cap (\alpha_1.ees \cup (\alpha_1.pes - \alpha_2.ees))) \cup (s \cap (\alpha_2.ees \cup (\alpha_2.pes - (\alpha_1.ees \cup \alpha_1.pes)))) \\ &= s \cap (\alpha_1.ees \cup (\alpha_1.pes - \alpha_2.ees) \cup \alpha_2.ees \cup (\alpha_2.pes - (\alpha_1.ees \cup \alpha_1.pes))) \\ &= s \cap (\alpha_1.ees \cup \alpha_2.ees \cup \alpha_1.pes \cup \alpha_2.pes) \end{aligned}$$

Since $s \subseteq \alpha.ees \cup \alpha.pes = (\alpha_1.ees \cup \alpha_2.ees \cup \alpha_1.pes \cup \alpha_2.pes)$, we have $s_1 \cup s_2 = s$. In general, $s \in S_\phi$, which implies that $S_\phi \supseteq \bigcup_{\alpha_i \in \Psi_\phi} \alpha_i$.

In general, $S_\phi = \bigcup_{\alpha_i \in \Psi_\phi} \alpha_i$.

- $\phi = \phi_1 \odot \phi_2$: The proof is similar to that of $\phi = \phi_1 \otimes \phi_2$.