Accepted, IEEE Transactions on Services Computing

Satisfiability Analysis of Workflows with Control-Flow Patterns and Authorization Constraints

Ping Yang [1], Xing Xie [2], Indrakshi Ray [2], Shiyong Lu [3]
[1]*Department of Computer Science, State University of New York at Binghamton, Binghamton, New York, 13902, USA*
[2]*Department of Computer Science, Colorado State University, Fort Collins, CO, 80523-1873, USA*
[3]*Department of Computer Science, Wayne State University, Detroit, Michigan 48202, USA*

***Abstract*—Workflow security has become increasingly important and challenging in today's open service world. While much research has been conducted on various security issues of workflow systems, the workflow satisfiability problem, which asks whether a set of users together can complete a workflow, is recently identified as an important research problem that needs more investigation. In this paper, we study the computational complexity of the problem along two directions: one is by considering either one path or all paths of a workflow, and the other is by considering the possible patterns in a workflow. We have shown that the general workflow satisfiability analysis problem is intractable. This result motivates us to consider restrictions on workflow control-flow patterns and access control policies, and to identify tractable cases of practical interest.**

Index Terms – workflow satisfiability analysis, access control

I. Introduction

Workflow security has become increasingly important and challenging in today's open service world [20], [6], [26], [10]. On one hand, a business process (workflow) can comprise numerous autonomous services that are exposed by different distributed service providers [3]. On the other hand, business partners from different enterprises can participate in the execution of an inter-organizational workflow collaboratively and cooperatively [26], [31]. While much research has been conducted on various security issues of workflow systems [20], [6], [28], [17], [4], [5], [30], [16], the workflow satisfiability problem [29], [9], [8], which asks whether a set of users can complete a workflow under the restrictions placed by an access control policy, is recently identified as an important research problem that needs more investigation.

To motivate our research, consider a simplified workflow for course registration and a role-based access control (RBAC) policy for the workflow illustrated in Figure 1. In order to register for a graduate course, a student first selects courses (task T_1) and then registers for the selected courses (task T_2). The registration needs to be approved by the graduate director. Tasks T_3 and T_4 are used to approve or disapprove the registration, respectively; these two tasks are connected through exclusive or denoted by $\#$, which means that either T_3 or T_4, but not both, can be executed. If the registration is approved, a process on behalf of the student registration officer will add the corresponding courses to the database (task T_5) and inform the student that the registration is approved (task T_6); otherwise, the process will inform the student that the registration is not approved (task T_6). In this workflow, T_1 is a *composite task* consisting of subtasks $T_{11}, \ldots, T_{1n}$, each of which is used to register for one course. These subtasks are connected through $|^{\leq 4}$ which specifies that the student can register for at most 4 courses. Other tasks are *atomic tasks*. This workflow can be executed multiple times by the same or different user. Each execution of the workflow is called an *execution run* of the workflow.

In the RBAC policy for this workflow, UA is a set of user-role assignments, PA is a set of permission-role assignments, and $=$, $\neq$, and $Cardin$ denote binding, separation, and cardinality constraints, respectively. The binding constraint $= (T_1, T_2)$ specifies that tasks T_1 and T_2 must be executed by the same user in each execution run of the workflow, i.e., the user who selects a course must be the user who registers for the course. The separation constraint $\neq (T_2, T_3)$ specifies that tasks T_2 and T_3 must be executed by different users in each execution run of the workflow, i.e., a user cannot both register for a course and approve the registration of the same course. The cardinality constraint $Cardin(u_1, 4)$ specifies that u_1 can execute no more than 4 tasks in each execution run of the workflow: since u_1 must execute T_2 to register for courses, u_1 can execute at most 3 subtasks of T_1 to select at most 3 courses for registration.

Design flaws or specification errors in a workflow access control policy may result in the leak of confidential data to unauthorized users. Data integrity may also be violated if unauthorized users modify the data. Therefore, correct specification of a workflow access control policy is critical to protect the confidentiality and the integrity of the data processed by the workflow. Workflow satisfiability analysis helps system administrators and workflow designers to understand an access control policy and detect potential flaws in the policy. For example, if all users together are not able to complete the execution of a workflow under the restrictions imposed by a workflow access control policy, then there might be errors in the access control specification. In practice, many workflows, especially those designed for processing scientific datasets, may contain hundreds of tasks, and the interactions of workflow tasks may be non-trivial due to the composition of complex control-flow patterns. As a result, it is challenging to check whether a set of users can complete the execution of a workflow by simple manual inspection alone. This motivates us to develop automated algorithms to address the problem.

A number of researchers have investigated the *workflow*

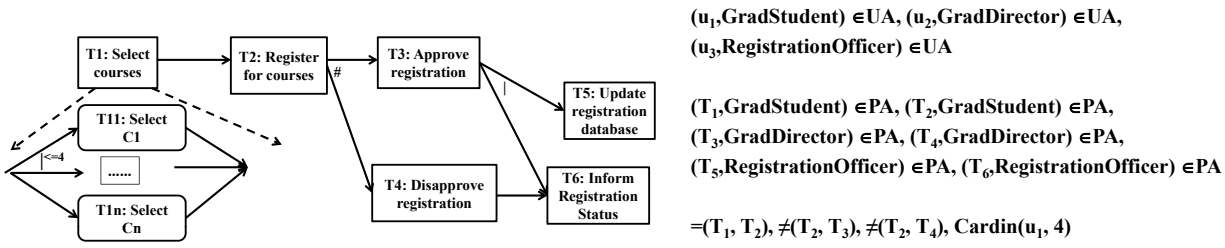

Fig. 1. A sample workflow for course registration.

satisfiability problem (WSP) [9], [29]. However, in their work, the workflow is specified as a partial order of tasks, which can only model the sequential and parallel split patterns. This paper considers a richer set of workflow patterns, including sequential, parallel split, exclusive or, multiple split, multiple split with upper/lower bound, condition, and loop. Considering exclusive-or significantly increases the complexity of WSP: the workflow defined in [9], [29] contains only one execution path; while with exclusive or, the number of execution paths may be exponential to the size of the workflow.

In this paper, we study the computational complexity of the workflow satisfiability problem for workflows with various control flow patterns and role-based access control policies with various constraints. In particular, we investigate the following subclasses of the workflow satisfiability problem: (1) Existential (Universal) workflow satisfiability, which asks whether a set of users together can complete the execution of one (all) of the paths of a workflow under the restrictions imposed by an RBAC policy; (2) Existential (Universal) workflow satisfiability with task constraints, which asks whether a set of users together can complete the execution of one (all) of the paths of a workflow that satisfy a task constraint, under the restrictions imposed by an RBAC policy; and (3) Existential (Universal) minimum role satisfiability, which computes a minimum set of roles that together can complete the execution of all tasks in one (all) of the paths of a workflow under the restrictions imposed by an RBAC policy. Our main contributions are summarized below.

- We have shown that the general workflow satisfiability analysis problem is intractable and identified a subset of tractable subclasses of the problem.
- We have shown that the existential workflow satisfiability with task constraints is NP-complete. We have also presented algorithms and complexity results for several subclasses of existential and universal workflow satisfiability problems with task constraints.
- We have presented algorithms for solving universal and existential minimum role satisfiability problems.

Organization: The rest of the paper is organized as follows. Section II provides definitions for workflow and role-based access control. Section III presents algorithms and complexity results for analysis problems related to workflow satisfiability. Section IV discusses related work and Section V concludes the paper with some pointers to future directions.

II. Preliminaries

A. Syntax and Execution Paths of the Workflow

Workflow is used to describe steps of business or scientific procedures, which helps automate and speed up business procedure and scientific discovery. A workflow consists of a number of tasks that are connected through various control-flow patterns specifying execution order of tasks.

Definition 1 [Workflow] *Let W be a workflow and T be an atomic task. The syntax of the workflow considered in this paper is given below.*

$$\begin{aligned} W \quad ::= \quad & T \mid W_1 \otimes W_2 \mid W_1 \# W_2 \mid W_1 \mid W_2 \\ & \mid |^{k} (W_1, \ldots, W_n) \mid |^{\leq k} (W_1, \ldots, W_n) \\ & \mid |^{\geq k} (W_1, \ldots, W_n) \mid \textit{if C then } W_1 \textit{ else } W_2 \\ & \mid \textit{while(C)}\{W_1\}W_2 \end{aligned}$$

The above workflow contains a subset of basic control-flow patterns defined in [21], as well as three new control-flow patterns proposed by us: $|^{k}$, $|^{\geq k}$, and $|^{\leq k}$. $\otimes$ represents the sequence pattern; $W_1 \otimes W_2$ specifies that W_2 is enabled after W_1 completes execution. $\#$ represents exclusive choice; $W_1 \# W_2$ specifies that either W_1 or W_2, but not both, can be executed in each workflow run. $\mid$ denotes parallel split with synchronization; $W_1 \mid W_2$ specifies that both W_1 and W_2 need to be completed in order to execute the next task. $|^{k} (W_1, \ldots, W_n)$, $|^{\leq k} (W_1, \ldots, W_n)$ and $|^{\geq k} (W_1, \ldots, W_n)$ ($k \leq n$) represent multiple split, multiple split with upper-bound, and multiple split with lower-bound, respectively; these three patterns specify that k, at least k, and at most k W_is need to be completed in order to execute the next task, respectively. if C then W_1 else W_2 denotes the conditional pattern and while$(C)\{W_1\}W_2$ specifies loops. For example, the workflow in Figure 1 can be specified as $|^{\leq 4} (T_{11}, \ldots, T_{1n}) \otimes T_2 \otimes ((T_3 \otimes (T_5 \mid T_6)) \# (T_4 \otimes T_6))$.

Below, we define a set of all execution paths of a workflow. An execution path of a workflow specifies a set of tasks executed in an execution run of the workflow as well as the execution order of tasks.

Definition 2 [Execution paths of a workflow] *Let $paths(W)$ denote a set of all execution paths of a workflow W. $paths(W)$ is defined in Figure 2.*

Rule 1 states that, if W is an atomic task T, then W contains only one execution path $\{T\}$. Rule 2 states that every

1. $paths(T) = \{\{T\}\}$, where T is an atomic task.
2. $paths(W_1 \otimes W_2) = \{append(p_1, p_2) \mid p_1 \in paths(W_1) \wedge p_2 \in paths(W_2)\}$ where $append(p_1, p_2)$ appends p_2 to the end of p_1.
3. $paths(W_1 \# W_2) = paths(\text{if } C \text{ then } W_1 \text{ else } W_2) = paths(W_1) \bigcup paths(W_2)$
4. $paths(W_1 \mid W_2) = union(\{interleave(p_1, p_2) \mid p_1 \in paths(W_1) \wedge p_2 \in paths(W_2)\})$ where
   - $interleave(p_1, p_2) = add(first(p_1), interleave(tail(p_1), p_2)) \cup add(first(p_2), interleave(p_1, tail(p_2)))$,
   - $first(p)$ returns the first task in p,
   - $tail(p)$ returns all but the first task in p,
   - $add(t, p)$ adds task t to the begining of every path in p, and
   - $union(\{S_1, \ldots, S_n\}) = S_1 \cup \ldots \cup \ldots S_n$.
5. $paths(|^k (W_1, \ldots, W_n)) = union(\{interleaveK(p_1, \ldots, p_k) \mid p_1 \in paths(W_1'), \ldots, p_k \in paths(W_k')\})$ where
   - $\{W_1', \ldots, W_k'\} \subseteq \{W_1, \ldots, W_n\}$ and $W_1' \neq \ldots \neq W_k'$
   - $interleaveK(p_1, \ldots, p_k) = add(first(p_1), interleave(tail(p_1), \ldots, p_k))$ $\cup \ldots \cup add(first(p_k), interleave(p_1, \ldots, tail(p_k)))$.
6. $paths(|^{\leq k} (W_1, \ldots, W_n)) = paths(|^1 (W_1, \ldots, W_n)) \cup \ldots \cup paths(|^k (W_1, \ldots, W_n))$
7. $paths(|^{\geq k} (W_1, \ldots, W_n)) = paths(|^k (W_1, \ldots, W_n)) \cup \ldots \cup paths(|^n (W_1, \ldots, W_n))$
8. $paths(\text{while}(C)\{W_1\}W_2) = paths(W_2) \bigcup \{append(p_1, p_2) \mid p_1 \in paths(W_1) \wedge p_2 \in paths(W_2)\}$

Fig. 2. The algorithm for computing a set of all execution paths of workflow W.

execution path of $W_1 \otimes W_2$ contains one execution path of W_1 followed by one execution path of W_2. Rule 3 defines execution paths of $W_1 \# W_2$ and if $(C)W_1$ else W_2. Since either W_1 or W_2, but not both, can be executed, an execution path of $W_1 \# W_2$ and if $(C)W_1$ else W_2 is either an execution path of W_1 or an execution path of W_2. Rule 4 computes all execution paths of $W_1 \mid W_2$ by interleaving the execution of paths of W_1 and W_2. In Rule 5, we compute all execution paths of $|^k (W_1, \ldots, W_n)$ by selecting k out of n W_is and then interleaving the execution of paths of the selected W_is. Rules 6 and 7 specify that $path(|^{\leq k} (W_1, \ldots, W_n))$ contains all execution paths of $|^1 (W_1, \ldots, W_n), \ldots,$ and $|^k (W_1, \ldots, W_n)$, and $path(|^{\geq k} (W_1, \ldots, W_n))$ contains all execution paths of $|^k (W_1, \ldots, W_n), \ldots,$ and $|^n (W_1, \ldots, W_n)$. Rule 8 defines execution paths of while loop; each path is either an execution path of W_2 (when the condition is false) or an execution path of W_1 followed by an execution path of W_2 (when the condition is true). For example, the set of all execution paths of $T_1 \otimes (T_2 \# T_3) \otimes |^1 (T_4, T_5, T_6)$ is $\{\{T_1, T_2, T_4\}$ $\{T_1, T_2, T_5\}$ $\{T_1, T_2, T_6\}$ $\{T_1, T_3, T_4\}$ $\{T_1, T_3, T_5\}$ $\{T_1, T_3, T_6\}\}$.

B. Role-Based Access Control for Workflows

Role-Based Access Control (RBAC) is a tuple $< U, R, P, UA, PA >$, where U is a finite set of users, R is a finite set of roles, P is a finite set of permissions, $UA \subseteq U \times R$ contains a set of user-role assignments, and $PA \subseteq P \times R$ contains a set of permission-role assignments. The user-role assignment $(u, r) \in UA$ specifies that user u is a member of role r, and the permission-role assignment $(p, r) \in PA$ specifies that role r is granted permission p.

We consider the following constraints on RBAC:

- *Cardinality constraints:* $Cardin(u, n)$, which specifies that a user u can execute at most n different workflow tasks in each execution run of the workflow.
- *Binding constraints:* $= (T_1, T_2)$*(called equality constraints in [29])*, which specifies that, if both tasks T_1 and T_2 are executed in an execution run of the workflow, then T_1 and T_2 must be executed by the same user.
- *Separation of duty constraints:* $\neq (T_1, T_2)$*(called inequality constraints in [29])*, which specifies that, if both tasks T_1 and T_2 are executed in an execution run of the workflow, then T_1 and T_2 must be executed by different users.

C. Task Constraints

A task constraint specifies the presence of a task in an execution path of a workflow. The notion of task constraint is formally defined below.

Definition 3 [Task Constraint] *A task constraint TC is defined as follows.*

$$TC \quad ::= \quad T \mid \neg T \mid TC_1 \vee TC_2 \mid TC_1 \wedge TC_2$$

T ($\neg T$) specifies that the execution path must (must not) contain task T. $TC_1 \vee TC_2$ specifies that the execution path must satisfy either TC_1 or TC_2, and $TC_1 \wedge TC_2$ specifies that the execution path must satisfy both TC_1 and TC_2.

III. The Workflow Satisfiability Problem

Let W be a workflow, ψ be an RBAC policy for W, U be a set of workflow users, and TC be a task constraint. In this section, we consider the following analysis problems related to the workflow satisfiability problem.

- *Existential (Universal) workflow satisfiability* WSP_E *(*WSP_U*)*, which asks if users in U together can complete the execution of one (all) paths of W under the restrictions imposed by ψ.
- *Existential workflow satisfiability with task constraint* WSP_{ET}, which asks if there *exists* $p \in paths(W)$ such that p satisfies TC and users in U together can complete the execution of all tasks in p under the restrictions imposed by ψ.
- *Universal workflow satisfiability with task constraint* WSP_{UT}, which asks if users in U together can complete the execution of *all* paths of W that satisfy TC, under the restrictions imposed by ψ.
- *Existential (universal) minimum role satisfiability* MRS_E *(*MRS_U*)*, which computes a minimum set of roles that together can complete the execution of all tasks in one (all) of the execution paths of W under the restrictions imposed by ψ.

A. Classification of Problem Instances

Due to the intractability of the workflow satisfiability problem in the general case, we consider a variety of restrictions on workflow satisfiability analysis problem instances. We consider three categories of restrictions defined below.

- Restricting workflow control-flow patterns
  - all patterns: the workflow contains all patterns.
  - $P_1, P_2, \ldots, P_n$ where $P_i \in \{\otimes, |, \#, |^k|^{\leq k}, |^{\geq k}$, if, while$\}$: the workflow contains only patterns P_1, P_2, ..., and P_n.
- Restricting access control policies
  - all constraints: the policy contains all constraints.
  - no constraint: the policy does not contain constraints.
  - $C_1, \ldots, C_n$ where $C_i \in \{B, C, S\}$, and B, C, and S represent binding, cardinality, and separation of duty constraints, respectively: the policy contains only constraints C_1, ..., and C_n.
- Restricting task constraints
  - $\wedge, \vee$: the task constraint contains both $\wedge$ and $\vee$.
  - $\wedge$: the task constraint contains only $\wedge$.
  - $\vee$: the task constraint contains only $\vee$.

Figure 3 summarizes our complexity results on WSP_E, WSP_U, WSP_{ET} and WSP_{UT}. Each box in the figure represents one problem class. The problem classes are arranged in a hierarchy. An edge from class P_1 to P_2 indicates that P_2 is a specialization of P_1, i.e., every hardness result for P_2 also applies to P_1 and the algorithm for P_1 can be used to solve P_2. Each result in the box is associated with a theorem number; for example, Th1 refers to Theorem III.1.

Some observations follow: (1) When the RBAC policy does not contain any constraints, WSP_E and WSP_U are solvable in polynomial time; (2) When the workflow contains all control-flow patterns, restricting the RBAC policy to contain only cardinality constraints does not make WSP_E and WSP_U tractable; (3) When the workflow contains only $\otimes$ and $|$, restricting the RBAC policy to contain only binding or cardinality constraints makes WSP_E and WSP_U tractable; and (4) When the access control policy does not contain any constraints, restricting the task constraint to contain only $\vee$ makes WSP_{ET} tractable.

B. Existential Workflow Satisfiability Problem: WSP_E

Given a workflow W, an RBAC policy ψ for W, and a set of workflow users U, WSP_E asks if there exists $p \in paths(W)$ such that users in U together are able to execute all tasks in p under the restrictions imposed by ψ.

The WSP_E problem can be solved by computing a set of all execution paths of W and then checking if there exists an execution path such that all tasks in the path can be executed by users in U. This approach, however, is very inefficient since the number of execution paths of a workflow may be exponential to the number of tasks in the workflow. For example, in the workflow $(T_1 \# T_2) \otimes \ldots \otimes (T_{2n-1} \# T_{2n})$, the number of execution paths is 2^n, which is exponential to the number of tasks in the workflow.

Below, we formally define an instance of the WSP_E problem and a solution to the problem.

Definition 4 (Instance of WSP_E) *An instance of WSP_E is defined as $E(W, \psi, U)$ where W is a workflow, ψ is an RBAC policy, and U is a set of users.*

Definition 5 (solution to WSP_E) *Let $EI = E(W, \psi, U)$ be a WSP_E instance and $tasks(p)$ be a set of all tasks in path p. A set of user-task assignments $A = \{(u_1, T_1), \ldots, (u_n, T_n)\}$ is a solution to EI if*

- *$T_1, \ldots, T_n$ are tasks in W,*
- *$T_1 \neq \ldots \neq T_n$,*
- *for each $(u, T) \in A$, there exists a role r such that ψ contains $(u, r) \in UA$ and $(T, r) \in PA$, and*
- *there exists $p \in paths(W)$ such that*
  - *$tasks(p) \subseteq \{T_1, \ldots, T_n\}$, and*
  - *$\{(u, T) \mid (u, T) \in A \wedge T \in tasks(p)\}$ conforms to constraints in ψ.*

$\{(u, T) \mid (u, T) \in A \wedge T \in tasks(p)\}$ in Definition 5 computes a set of all user-task assignments in A that assign users to tasks in path p.

B.1. Proofs for NP-hardness of WSP_E

Theorem III.1 *WSP_E for workflows containing only $\otimes$ and $\#$, and RBAC policies containing only cardinality constraints is NP-hard.*

Proof: Below, we show that the problem is NP-hard by reduction from the 3-CNF satisfiability problem, which is known to be NP-hard. Let $F = F_1 \wedge F_2 \wedge \ldots \wedge F_n$ be a 3-CNF formula. We construct a WSP_E instance $E(W, \psi, U)$ as follows.

- The workflow W is constructed from F as follows:
  - $\wedge$ is mapped to $\otimes$,
  - $\vee$ is mapped to $\#$,
  - each positive literal l_i is mapped to a task T_i, and
  - each negative literal $\neg l_i$ is mapped to a task T'_i.
- The RBAC policy ψ is constructed from F as follows:
  - For every literal l_i in F, if l_i appears positively in F, then $(u_i, r_i) \in UA$ and $(T_i, r_i) \in PA$ are added to ψ. If l_i appears negatively in F, then $(u_i, r_i) \in UA$ and $(T'_i, r_i) \in PA$ are added to ψ.
  - If a literal l_i appears both positively and negatively in F, then a cardinality constraint $Cardin(u_i, 1)$ is added to ψ, which specifies that user u_i can execute at most one task, i.e. either T_i or T'_i.
- $U = \{u_1, \ldots, u_m\}$ where m is the number of literals in F.

The corresponding WSP_E problem wsp_e is: does there exist $p \in paths(W)$ such that a set of users $\{u_1, \ldots, u_m\}$ together can complete the execution of all tasks in p under ψ. The worst-case complexity of the reduction is $O(|F|)$.

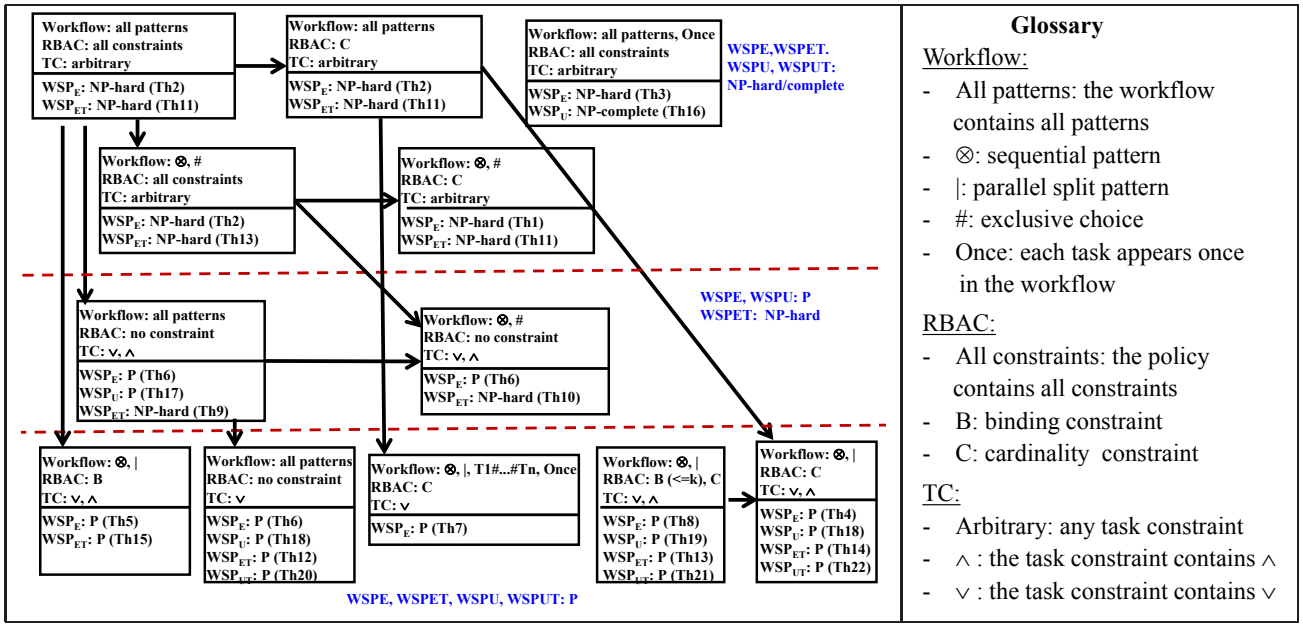

Fig. 3. Summary of Complexity Results of WSP_E, WSP_U, WSP_{ET}, and WSP_{UT}.

Let $W = W_1 \otimes \ldots \otimes W_n$. Below, we show that F is satisfiable iff wsp_e has a solution.

Proof for "only if": Assume that F is true under the set of assignments $A = \{(l_1, v_1), \ldots, (l_m, v_m)\}$. Let $tasks(W)$ represent a set of all tasks of W. Below, we prove that $A' = \{(u_i, T_i) \mid (l_i, true) \in A \wedge T_i \in tasks(W)\} \cup \{(u_i, T_i') \mid (l_i, false) \in A \wedge T_i' \in tasks(W)\}$ is a solution to wsp_e by contradiction. Assume that A' is not a solution, then either A' violates the cardinality constraint or there exists W_i such that all tasks in W_i are not assigned to any user. Because a literal l_j cannot be both true and false in A, u_j can execute either T_j or T_j', but not both. Therefore, A' does not violate the cardinality constraint. We now assume that all tasks in W_i are not assigned to any user. For every task in W_i, if the task is T_{ik}, then l_{ik} appears positively in F_i (from the reduction) and $v_{ik} = false$ (from the definition of A' and the fact that T_{ik} is not assigned to any user, and thus not in A'). Similarly, for every task in W_i, if the task is T_{ik}', then l_{ik} appears negatively in F_i and $v_{ik} = true$. As a result, F_i is false under A, which is a contradiction. Thus, A' is a solution to wsp_e.

Proof for "if": Assume that wsp_e has a solution $A' = \{(u_1, t_1), \ldots, (u_m, t_m)\}$ where t_i is either T_i or T_i'. Below, we show that F is true under the set of assignments $A = \{(l_i, true) \mid t_i = T_i\} \cup \{(l_i, false) \mid t_i = T_i'\}$ by contradiction. Assume that F is false under A, then there exists an F_i which is false under A. If l_{ik} appears positively in F_i, then l_{ik} is false and hence $(u_{ik}, T_{ik}) \notin A$. If l_{ik} appears negatively in F_i, then l_{ik} is true and hence $(u_{ik}, T_{ik}') \notin A$. As a result, all tasks in W_i are not assigned to any user and hence A' is not a solution of wsp_e, which is a contradiction. Therefore, the problem is NP-hard. The theorem is proved. ■

Note that the reduction in Theorem III.1 allows each task to appear in the workflow more than once. It is not clear if the problem is still NP-hard if each task appears in the workflow only once. It is also not clear if the problem is in NP.

The following theorem is a corollary of Theorem III.1, which shows that WSP_E is NP-hard for more restricted problem classes.

Theorem III.2 *WSP_E is NP-hard for the following: (1) workflows containing only $\otimes$ and $\#$, and RBAC policies containing all constraints, (2) workflows containing all control-flow patterns and RBAC policies containing all constraints, (3) workflows containing all control-flow patterns and RBAC policies containing only cardinality constraints.*

Theorem III.3 *WSP_E for workflows where each task appears once and RBAC policies containing all constraints is NP-hard.*

Proof: Let $E(W, \psi, U)$ be an instance of the problem. Below, we show that the problem is NP-hard. First, we apply the reduction in Theorem III.1. Next, we rename every task in W to a unique name that does not appear in W and use $name(T)$ to represent a set of all unique names corresponding to T. We then add binding constraints $\{= (t_1, t_2) \mid t_1 \in name(T) \wedge t_2 \in name(T) \wedge t_1 \neq t_2\}$ and $\{(t_i, r) \in PA \mid t_i \in name(T) \wedge (T, r) \in PA\}$ to the RBAC policy ψ. Finally, we add separation of duty constraints $\{\neq (t_1, t_2) \mid t_1 \in name(T) \wedge t_2 \in name(T')\}$ to ψ. The proof of the correctness of the reduction is similar to that of Theorem III.1. ■

B.2 Proofs for Polynomial-time WSP_E

Theorem III.4 *WSP_E for workflows containing only $\otimes$ and $|$, and RBAC policies containing only cardinality constraints is solvable in polynomial time.*

Proof: Let $E(W, \psi, U)$ be an instance of the problem. Below, we give a polynomial algorithm which reduces WSP_E to the maximum bipartite matching problem.

First, we construct a bipartite graph $G_b = \langle S_u \cup S_t, E \rangle$, where S_u and S_t are sets of all vertices corresponding to users and workflow tasks, respectively. let V_u represent the vertex corresponding to user u and V_T represent the vertex corresponding to task T. There is an edge $(V_u, V_T) \in E$ if there exists a role r such that ψ contains $(u, r) \in UA$ and $(T, r) \in PA$, i.e., user u has permission to execute task T. Next, the algorithm duplicates every vertex V_u $n-1$ times if ψ contains a cardinality constraint $Cardin(u, n)$. If ψ does not contain a cardinality constraint for u, then we duplicate V_u $|W|-1$ times, where $|W|$ is the number of tasks in W. The algorithm then adds edges from the new nodes to tasks that have edges with V_u. Finally, we apply the maximum bipartite matching algorithm [19] to compute the maximum matching between S_u and S_t. The algorithm returns true if the size of the maximum matching is equal to the number of tasks.

Below, we use an example to illustrate our algorithm. Consider the workflow $(T_1 \mid T_3) \otimes T_2 \otimes T_4 \otimes T_5$. Assume that there are two users u_1 and u_2, u_1 has permission to execute task T_1, T_2, and T_3, and u_2 has permission to execute T_3, T_4 and T_5. Also, assume that $Cardin(u_1, 2) \in \psi$ and $Cardin(u_2, 3) \in \psi$.

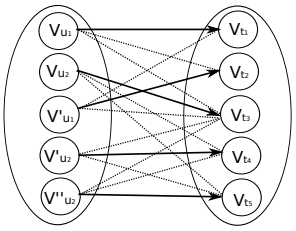

Fig. 4. Example of Maximum Bipartite Matching.

First, we construct a bipartite graph, which contains two user vertices V_{u_1} and V_{u_2}, and five task vertices V_{t_1}, V_{t_2}, V_{t_3}, V_{t_4} and V_{t_5}. There are six edges in the original bipartite graph: $V_{u_1} \rightarrow V_{t_1}$, $V_{u_1} \rightarrow V_{t_2}$, $V_{u_1} \rightarrow V_{t_3}$, $V_{u_2} \rightarrow V_{t_3}$, $V_{u_2} \rightarrow V_{t_4}$, $V_{u_2} \rightarrow V_{t_5}$. Next, we duplicate V_{u_1} once which results in V'_{u_1}, and duplicate u_2 twice which results in new nodes V'_{u_2} and V''_{u_2}. In addition, edges are also created from the new nodes to tasks that are connected with the corresponding user nodes. Finally, we compute the maximum matching of the graph, which is equal to 5 and hence the algorithm returns true. Figure 4 gives the constructed bipartite graph, in which the solid edges represent one maximum matching: u_1 executes tasks T_1 and T_2, and u_2 executes T_3, T_4, and T_5.

Because a user can execute at most $|W|$ tasks, there are at most $|W|$ duplicates for each user. Thus, G_B contains at most $|U||W| + |W|$ vertices and $|U||W|^2$ edges. The worst-case complexity of computing the maximum bipartite matching is $O(\sqrt{(|U||W| + |W|)} \cdot |U||W|^2))$, which can be simplified as $O(|U|^{\frac{3}{2}}|W|^{\frac{5}{2}})$. Therefore, the worst-case complexity of the algorithm is $O(|U|^{\frac{3}{2}}|W|^{\frac{5}{2}})$ and hence the theorem holds. ■

Theorem III.5 *WSP_E for workflows containing $\otimes$ and $|$, and RBAC policies containing only binding constraints is solvable in polynomial time.*

Proof: Let $E(W, \psi, U)$ be an instance of the problem. Algorithm 1 gives a polynomial algorithm for solving the problem. The algorithm is similar to that in Theorem 5 of [29], except that we consider transitive relationship among binding constraints, while they do not.

Algorithm 1 Algorithm for solving WSP_E in Theorem III.5.

```
1:  Procedure wspbind(W, ψ, U)
2:  s = ∅;
3:  for all task T in workflow W do
4:      auth(T) = {u | there exists a role r such that (u, r) ∈ UA ∧ (T, r) ∈ PA};
5:  end for
6:  for all = (T1, T2) ∈ ψ do
7:      flag = 0;
8:      for all e ∈ s do
9:          if T1 ∈ e then
10:             e1 = e ∪ {T2}; s = (s \ {e}) ∪ {e1}; flag = 1;
11:         else if T2 ∈ e then
12:             e1 = e ∪ {T1}; s = (s \ {e}) ∪ {e1}; flag = 1;
13:         end if end if
14:     end for
15:     if flag == 0 then s = s ∪ {{T1, T2}}; end if
16: end for
17: for all e ∈ s do
18:     intersect = ∅;
19:     for all T ∈ e do intersect = intersect ∩ auth(T); end for
20:     if intersect == ∅ then return false;
21:     else for all T ∈ e do auth(T) = intersect; end for
22: end for
```

First, we compute, for each task T, a set $auth(T)$ of users who are authorized to execute T without considering binding constraints (Lines 3 – 5). Next, we compute a set s, where each element in s contains a set of tasks that need to be executed by the same user under binding constraints (Lines 6 – 16). Finally, for every set $\{T_{i1}, \ldots, T_{in}\} \in s$, we replace $auth(T_{i1}), \ldots,$ and $auth(T_{in})$ with $auth(T_{i1}) \cap \ldots \cap auth(T_{in})$ (Lines 17 – 22). Let b be the number of binding constraints. Computing initial values of $auth(T)$ (Lines 3–5) takes $O(|W||\psi|)$, computing s takes $O(b^2 log|W|)$, and computing intersection of $intersect$ and $auth(T)$ (Line 19) takes $|U|log|U|$. Therefore, the worst-case complexity of the algorithm is $O(|W||\psi| + b|W|log|W| + b|W||U|log|U|)$. ■

Theorem III.6 *WSP_E for workflows containing all control-flow patterns and RBAC policies that do not contain constraints is solvable in polynomial time.*

Proof: Let $E(W, \psi, U)$ be an instance of the problem. The algorithm is given in Figure 5 (function $wspesatall(W, U, \psi)$).

Rules 1 – 4 are straightforward. Rule 5 handles $|^k (W_1, \ldots, W_n)$ and $|^{\geq k} (W_1, \ldots, W_n)$: if there exist $\{W'_1, \ldots, W'_k\} \subseteq \{W_1, \ldots, W_n\}$ and $p_i \in paths(W'_i)$ such that, for every i, $tasks(p_i) \subseteq \{T \mid (u, r) \in UA$ and $(T, r) \in PA$ are in ψ for some $u \in U\}$, then the algorithm returns true. $|^{\leq}$ is handled similarly in Rule 6. Because checking if a task can be executed by a user takes $O(|\psi|)$ and each task in the workflow is processed only once, the worst-case complexity of the algorithm is $O(|U||\psi||W|)$, ■

Theorem III.7 *WSP_E for workflows containing $\otimes$, $|$, and $\#$ of the form $T_1 \# \ldots \# T_n$, where each task appears only once, and RBAC policies containing only cardinality constraints is solvable in polynomial time.*

1. $wspesatall(T,U,\psi) = \begin{cases} true & \text{there exists } u \in U \text{ and a role } r \text{ such that } \psi \text{ contains } (u,r) \in UA \wedge (T,r) \in PA \\ false & \text{otherwise} \end{cases}$
2. $wspesatall(W_1 \# W_2, U, \psi) = wspesatall(\text{if } C \text{ then } W_1 \text{ else } W_2, U, \psi) = wspesatall(W_1, U, \psi) \vee wspesatall(W_2, U, \psi)$
3. $wspesatall(W_1 \otimes W_2, U, \psi) = wspesatall(W_1 \mid W_2, U, \psi) = wspesatall(W_1, U, \psi) \wedge wspesatall(W_2, U, \psi)$
4. $wspesatall(\text{while } (C)\{W_1\}W_2, U, \psi) = wspesatall(W_2, U, \psi)$
5. $wspesatall(|^k\ (W_1, \ldots, W_n), U, \psi) = wspesatall(|^{\geq k}\ (W_1, \ldots, W_n), U, \psi)$
$= \begin{cases} true & |\ \{W_i \mid wspsatall(W_i, U, \psi) == true\}\ | \geq k \\ false & \text{otherwise} \end{cases}$
6. $wspesatall(|^{\leq k}\ (W_1, \ldots, W_n), U, \psi) = \begin{cases} true & wspsatall(W_i, U, \psi) == true \text{ for some } W_i \\ false & \text{otherwise} \end{cases}$

Fig. 5. The algorithm for proving Theorem III.6.

Proof: Let $E(W, \psi, U)$ be an instance of the problem. The algorithm is described below. First, we construct a bipartite graph, which is similar to the one constructed in Theorem III.4 except that, (1) we create a vertex $V_{T_{1n}}$ for $T_1 \# \ldots \# T_n$ (instead of a vertex for each T_i) since only one of the T_is can be executed in every workflow run, and (2) we add edges from nodes representing users who can execute $T_1, \ldots,$ or T_n to $V_{T_{1n}}$. We then compute the maximum matching of the graph. Since each task appears only once in W, a user u can execute at most m sub-workflows of the form $T_1 \# \ldots \# T_n$ in every workflow run if the cardinality constraint for the user is $Cardin(u, m)$. Therefore, if the size of the maximum matching is equal to the number of vertices representing tasks, then the problem is true. Because a user can execute at most $|W|$ tasks, there are at most $|W|$ duplicates for each user. Since there are at most $|W|$ vertices representing tasks, the graph contains at most $|U||W| + |W|$ vertices and $|U||W|^2$ edges. Since the worst-case complexity of computing the maximum bipartite matching is $O(|U|^{\frac{3}{2}}|W|^{\frac{5}{2}})$, the worst-case complexity of the algorithm is $O(|U|^{\frac{3}{2}}|W|^{\frac{5}{2}})$. ■

Below, we show that, when the number of binding constraints is limited to a constant k, WSP_E is solvable in polynomial time.

Theorem III.8 *WSP_E for workflows containing $\otimes$ and $|$, and RBAC policies containing only binding and cardinality constraints where the number of binding constraints is less than a constant k, is solvable in polynomial time.*

Proof: Let $E(W, \psi, U)$ be an instance of the problem. The algorithm is given below. First, we apply lines 2–16 of Algorithm 1 to compute set s. Next, for each set $e \in s$, we pick a user, say u, who is authorized to execute all tasks in e under the restriction placed by cardinality constraints and assign u to all tasks in e. If such a user does not exist, then the algorithm returns false. If u is assigned to m tasks and ψ contains $Cardin(u, n)$, then we replace $Cardin(u, n)$ with $Cardin(u, n - m)$. Finally, we apply the algorithm in Theorem III.4 to compute the maximum matching for users in $U \setminus \{u \mid Cardin(u, 0) \in \psi\}$ and tasks that have not been assigned to any user. If no matching is found, we pick another set of users that are authorized to execute tasks in binding constraints and repeat the above process. Since computing s takes $O(|W||\psi|) + O(k^2 log|W|)$, finding a user who is authorized to execute all tasks in e takes $O(|\psi|)$, computing the maximum bipartite matching is $O(|U|^{\frac{3}{2}}|W|^{\frac{5}{2}})$, and we need to try at most $|U|^k$ combinations of users, the worst-case complexity of the algorithm is $O(|W||\psi| + k^2 log|W| + |U|^k(k|\psi| + |U|^{\frac{3}{2}}|W|^{\frac{5}{2}}))$. ■

C. Existential Workflow Satisfiability with Task Constraints WSP_{ET}

Let W be a workflow, U be a set of users, ψ be an RBAC policy, and TC be a task constraint. WSP_{ET} asks if there exists $p \in paths(W)$ such that p satisfies TC and users in U together can complete the execution of p under the restriction place by ψ.

Definition 6 (Task constraint satisfiability) *Let W be a workflow, $p \in paths(W)$, TC be a task constraint, and $TC' = subst(TC, (\{T \mapsto true | T \in p\} \cup \{T \mapsto false | T \notin p\} \cup \{\neg T \mapsto true | T \notin p\} \cup \{\neg T \mapsto false | T \in p\}))$, where $subst(TC, \{t_1 \mapsto v_1, \ldots, t_n \mapsto v_n\})$ substitutes t_i in TC with v_i. We say that p satisfies TC if and only if TC' is true.*

Below, we formally define an instance of the WSP_{ET} problem and a solution to the WSP_{ET} problem.

Definition 7 (WSP_{ET} instance) *A WSP_{ET} instance is defined as $ET(W, \psi, U, TC)$ where W is a workflow, ψ is an RBAC policy, U is a set of users, and TC is a task constraint.*

Definition 8 (Solution to WSP_{ET}) *Let $ETI = ET(W, \psi, U, TC)$ be a WSP_{ET} instance, TC be a task constraint, $p \in paths(W)$, and $tasks(p)$ be a set of all tasks in p. We say that a set of user-task assignments $A = \{(u_1, T_1), \ldots, (u_n, T_n)\}$ is a solution to ETI iff*

- *$T_1, \ldots, T_n$ are tasks in W,*
- *$T_1 \neq \ldots \neq T_n$,*
- *for each $(u, T) \in A$, there exists a role r such that ψ contains $(u, r) \in UA$ and $(T, r) \in PA$, and*
- *there exists $p \in path(W)$ such that*
  - *$tasks(p) \subseteq \{T_1, \ldots, T_n\}$,*
  - *p satisfies TC, and*
  - *$\{(u, T) \mid (u, T) \in A \wedge T \in tasks(p)\}$ conforms to constraints in ψ.*

C.1. Proofs for NP-Hardness Results of WSP_{ET}

Theorem III.9 *WSP_{ET} for workflows containing all patterns, RBAC policies containing no constraints, and task constraint containing both $\wedge$ and $\vee$ is NP-hard.*

$wspetnoand(W, U, \psi, TC) = tsatnoand((elim(W, U, \psi), TC)$.

$elim$ is defined as follows.

1. $elim(T, U, \psi) = \begin{cases} T & \text{there exists } u \in U \text{ such that } \psi \textbf{ contains } (u, r) \in UA \text{ and } (T, r) \in PA \\ null & \text{otherwise} \end{cases}$
2. $elim(W_1 \# W_2, U, \psi) = \begin{cases} elim(W_1, U, \psi) & elim(W_2, U, \psi) = null \\ elim(W_2, U, \psi) & elim(W_1, U, \psi) = null \\ elim(W_1, U, \psi) \# elim(W_2, U, \psi) & \text{otherwise} \end{cases}$
3. $elim(W_1 \mid W_2, U, \psi) = \begin{cases} null & elim(W_2, U, \psi) = null \vee elim(W_1, U, \psi) = null \\ elim(W_1, U, \psi) \mid elim(W_2, U, \psi) & \text{otherwise} \end{cases}$
4. $elim(W_1 \otimes W_2, U, \psi) = \begin{cases} null & elim(W_2, U, \psi) = null \vee elim(W_1, U, \psi) = null \\ elim(W_1, U, \psi) \otimes elim(W_2, U, \psi) & \text{otherwise} \end{cases}$
5. $elim(|^k\ (W_1, \ldots, W_n, U, \psi) = \begin{cases} |^k\ (W'_1, \ldots, W'_m) \text{ where} \\ \{W'_1, \ldots, W'_m\} = \{elim(W_i, U, \psi) \mid elim(W_i, U, \psi) \neq null\} & | \{elim(W_i, U, \psi) \mid elim(W_i, U, \psi) \neq null\} | \geq k \\ null & \text{otherwise} \end{cases}$
6. $elim(|^{\geq k}\ (W_1, \ldots, W_n, U, \psi) = \begin{cases} |^{\geq k}\ (W'_1, \ldots, W'_m) \text{ where} \\ \{W'_1, \ldots, W'_m\} = \{elim(W_i, U, \psi) \mid elim(W_i, U, \psi) \neq null\} & | \{elim(W_i, U, \psi) \mid elim(W_i, U, \psi) \neq null\} | \geq k \\ null & \text{otherwise} \end{cases}$
7. $elim(|^{\leq k}\ (W_1, \ldots, W_n, U, \psi) = \begin{cases} |^{\leq k}\ (W'_1, \ldots, W'_m) \text{ where} \\ \{W'_1, \ldots, W'_m\} = \{elim(W_i, U, \psi) \mid elim(W_i, U, \psi) \neq null\} & | \{elim(W_i, U, \psi) \mid elim(W_i, U, \psi) \neq null\} | \geq 1 \\ null & \text{otherwise} \end{cases}$
8. $elim(\text{if } (C) W_1 \text{ else } W_2, U, \psi) = \begin{cases} null & elim(W_1, U, \psi) == null \wedge elim(W_2, U, \psi) == null \\ \text{if } (C)\ elim(W_1, U, \psi) \text{ else } elim(W_2, U, \psi) & \text{otherwise} \end{cases}$
9. $elim(\text{while } (C)\{W_1\}\{W_2\}, U, \psi) = \begin{cases} null & elim(W_2, U, \psi) == null \\ \text{while } (C)\{elim(W_1, U, \psi)\} elim(W_2, U, \psi) & \text{otherwise} \end{cases}$

$tsatnoand$ is defined as follows, where t_i is T or $\neg T$ for some task T.

1. $tsatnoand(null, T') = tsatnoand(null, \neg T') = false$
2. $tsatnoand(T, t_1 \vee \ldots \vee t_n) = tsatnoand(T, t_1) \vee \ldots \vee tsatnoand(T, t_n)$
3. $tsatnoand(T, T') = \begin{cases} true & T == T' \\ false & \text{otherwise} \end{cases}$
4. $tsatnoand(T, \neg T') = \begin{cases} true & T \neq T' \\ false & \text{otherwise} \end{cases}$
5. $tsatnoand(W_1 \# W_2, T') = tsatnoand(W_1 \mid W_2, T') = tsatnoand(W_1 \otimes W_2, T')$
   $= tsatnoand(\text{if } (C) \text{ then } W_1 \text{ else } W_2, T')$
   $= tsatnoand(\text{while } (C)\ \{W_1\}\ W_2, T') = tsatnoand(W_1, T') \vee tsatnoand(W_2, T')$
6. $tsatnoand(W_1 \# W_2, \neg T') = tsatnoand(\text{if } (C) \text{ then } W_1 \text{ else } W_2, \neg T') = tsatnoand(W_1, \neg T') \vee tsatnoand(W_2, \neg T')$
7. $tsatnoand(W_1 \mid W_2, \neg T') = tsatnoand(W_1 \otimes W_2, \neg T') = tsatnoand(W_1, \neg T') \wedge tsatnoand(W_2, \neg T')$
8. $tsatnoand(\text{while } (C)\ \{W_1\}\ W_2, \neg T') = tsatnoand(W_2, \neg T')$
9. $tsatnoand(|^k\ (W_1, \ldots, W_n), T') = tsatnoand(|^{\leq k}\ (W_1, \ldots, W_n), T') = tsatnoand(|^{\geq k}\ (W_1, \ldots, W_n), T')$
   $= \begin{cases} true & tsatnoand(W_i, T') == true \text{ for some } W_i \\ false & \text{otherwise} \end{cases}$
10. $tsatnoand(|^k\ (W_1, \ldots, W_n), \neg T') = tsatnoand(|^{\geq k}\ (W_1, \ldots, W_n), \neg T') = \begin{cases} true & | \{W_i \mid tsatnoand(W_i, \neg T') == true\} | \geq k \\ false & \text{otherwise} \end{cases}$
11. $tsatnoand(|^{\leq k}\ (W_1, \ldots, W_n), \neg T') = \begin{cases} true & | \{W_i \mid tsatnoand(W_i, \neg T') == true\} | \geq 1 \\ false & \text{otherwise} \end{cases}$

Fig. 6. The algorithm for proving Theorem III.12.

Proof: Below, we show that the problem is NP-hard by providing a polynomial time reduction from the 3-CNF satisfiability problem to the problem. Let $F = F_1 \wedge F_2 \wedge \ldots \wedge F_n$ be a 3-CNF formula. We construct a WSP_{ET} instance $ET(W, \psi, U, TC)$ as follows.

- The task constraint TC is constructed from F by replacing each literal l_i in F with a task T_i.
- The workflow $W = ((T_1 \# T'_1) \otimes \ldots \otimes (T_m \# T'_m))$ is constructed from F, where m is the number of literals in F, all T'_is do not appear in TC, and $T'_1 \neq \ldots \neq T'_m$.
- $U = \{u\}$
- The RBAC policy ψ is constructed as follows. First, we add $(u, r) \in UA$ to ψ. Next, for every T_i and T'_i in W, we add $(T_i, r) \in PA$ and $(T'_i, r) \in PA$ to ψ.

The corresponding WSP_{ET} problem wsp_{et} is: does there exist $p \in paths(W)$ such that p satisfies TC and user u can complete the execution of p under the restriction place by ψ?

Let $TC = C_1 \wedge \ldots \wedge C_n$. Below, we show that F is satisfiable iff wsp_{et} has a solution.

Proof for "only if": Assume that F is true under a set of assignments $\{(l_1, v_1), \ldots, (l_m, v_m)\}$. We show that the path p of W, constructed as follows, satisfies TC under ψ: If v_i is true, then p contains T_i; otherwise, p contains T'_i. Assume that this is not the case, then p does not satisfy C_i for some $1 \leq i \leq n$. If T_j appears positively in C_i, then T'_j is in p and hence l_j is false. If T_j appears negatively in C_i, then T_j is in p and hence l_j is true. As a result, F_i is false, which is a contradiction.

Proof for "if": Assume that there exists $p \in paths(W)$ that satisfies TC. Then p satisfies all C_is. If T_i appears in p, then we assign l_i true. If T'_i appears in p, then we assign l_i false. We now show that F is true under the above assignment by contradiction. Assume that this is not the case, i.e., some F_i is not true under the above assignment. If a literal l_j appears positively in F_i, then l_j is false and hence T'_j is in p. If a literal l_j appears negatively in F_i, then l_j is true and hence T_j is in p. As a result, p does not satisfy C_i and hence does not satisfy TC, which is a contradiction. ■

Theorem III.10 *WSP_{ET} for workflows containing only $\otimes$ and $\#$, RBAC policies containing no constraints, and task constraint containing both $\wedge$ and $\vee$ is NP-hard.*

Proof: The proof directly follows that of Theorem III.9. ■

The following theorem is a corollary of theorems in Section III-B.

Theorem III.11 *WSP_{ET} is NP-hard for the following with arbitrary task constraints: (1) workflows containing only $\otimes$ and $\#$, and RBAC policies containing only cardinality constraints; (2) workflows containing all patterns, and RBAC policies containing only cardinality constraints; (3) workflows containing only $\otimes$ and $\#$, and RBAC policies containing all constraints; (4) workflows containing all control-flow patterns, and RBAC policies containing all constraints.*

C.2. Proofs for Polynomial-Time WSP_{ET}

Theorem III.12 *WSP_{ET} for workflows containing all patterns, RBAC policies containing no constraints, and task constraints that do not contain $\wedge$ is solvable in polynomial time.*

Proof: Let $ET(W, \psi, U, TC)$ be an instance of the problem. Figure 6 gives a polynomial algorithm for solving the problem. The algorithm is defined as a function $wspetnoand(W, U, \psi, TC)$, which first calls $elim(W, U, \psi)$ to eliminate all execution paths of W that cannot be completed by users in U under ψ (i.e., $elim(W, U, \psi)$ returns the largest sub-workflow of W in which all execution paths can be completed by users in U), and then calls $tsatnoand((elim(W, U, \psi), TC)$ to check if there exists $p \in paths(elim(W, U, \psi))$ that satisfies the task constraint TC. $null$ in function $elim$ represents a (sub)-workflow that cannot be completed by any user. The problem is true iff $wspetnoand(W, U, \psi, TC)$ returns true.

For example, consider the workflow $W = T_1\#(T_2 \mid T_3)$, the RBAC policy $\psi = \{(u, r) \in UA, (T_1, r) \in PA, (T_2, r) \in PA\}$, the task constraint $TC = \neg T_1 \vee T_2$, and $U = \{u\}$. Since no user has permission to execute T_3, $T_2 \mid T_3$ cannot be completed by any user and hence $elim(W, U, \psi)$ returns T_1. Next, we compute $tsatnoand(T_1, \neg T_1 \vee T_2)$, which returns $false$. Thus, the problem is false.

Since it takes $O(|\psi|)$ to check if there exists a user who has permission to execute task T and each task is processed once in function $elim$, the worst-case complexity of $elim$ is $O(|W||\psi|)$. Since each task and each constraint are processed once in $tsatnoand$, the worst-case complexity of $tsatnoand$ is $O(|W||TC|)$. Therefore, the worst-case complexity of the algorithm is $O(|W||\psi| + |W||TC|)$. ■

Theorem III.13 *WSP_{ET} for workflows containing only $\otimes$ and $|$, RBAC policies containing only binding and cardinality constraints where the number of binding constraints is less than a constant k, and task constraints containing both $\wedge$ and $\vee$ is solvable in polynomial time.*

Proof: When the workflow W contains only $\otimes$ and $|$, there is only one execution path in the workflow. We first apply the algorithm in Theorem III.8 to check if all tasks of W can be executed by users in U under the restriction placed by an RBAC policy ψ. If not, the algorithm returns false. Otherwise, for every T in the task constraint TC, if T appears in W, then we replace T with true; otherwise, we replace T with false. WSP_{ET} is true if and only if the resulting task constraint is true. Since the worst-case complexity of the algorithm in Theorem III.8 is $O(|W||\psi| + k^2 log|W| + |U|^k(k|\psi| + |U|^{\frac{3}{2}}|W|^{\frac{5}{2}}))$, checking if a task T appears in W takes $O(|W|)$, and replacing T in TC with true/false takes $O(|TC|)$, the worst-case complexity of the algorithm is $O(|W||\psi| + k^2 log|W| + |U|^k(k|\psi| + |U|^{\frac{3}{2}}|W|^{\frac{5}{2}}) + |TC||W|)$. Thus, the problem is solvable in polynomial time. ■

The following theorem is a corollary of Theorem III.13.

Theorem III.14 *WSP_{ET} is solvable in polynomial time for the following workflows and RBAC policies, and task constraints containing both $\wedge$ and $\vee$: (1) workflows containing only $\otimes$ and $|$, RBAC containing no constraints; (2) workflows containing only $\otimes$ and $|$, RBAC containing only binding constraints where the number of binding constraints is less than a constant k; and (3) workflows containing only $\otimes$ and $|$, RBAC containing only cardinality constraints.*

D. Universal Workflow Satisfiability Problem: WSP_U

WSP_U asks if a set of users U together are able to complete the execution of all paths of a workflow W under the restriction placed by an RBAC policy ψ.

Definition 9 (Instance of WSP_U) *An instance of WSP_U is defined as $WU(W, \psi, U)$ where W is a workflow, ψ is an RBAC policy, and U is a set of users.*

Definition 10 (solution to WSP_U) *Let $UI = WU(W, \psi, U)$ be a WSP_U instance and $tasks(W)$ be a set of all tasks in workflow W. A set of user-task assignments $A = \{(u_1, T_1), \ldots, (u_n, T_n)\}$ is a solution to UI if*

- $\{T_1, \ldots, T_n\} = tasks(W)$,
- $T_1 \neq \ldots \neq T_n$,
- *for each $(u, T) \in A$, there exists a role r such that ψ contains $(u, r) \in UA$ and $(T, r) \in PA$, and*
- *for every $p \in paths(W)$, $\{(u, T) \mid (u, T) \in A \wedge p$ contains $T\}$ conforms to constraints in ψ.*

D.1. NP-Completeness Results for WSP_U

In this section, we show that WSP_U for workflows where each task appears only once, and RBAC policies containing all constraints is NP-complete. Before we prove the theorem, we first show that checking if there exists $p \in paths(W)$ such that p contains both tasks T_1 and T_2 is polynomial.

Lemma III.15 *Let W be a workflow, and T_1 and T_2 be two tasks in W. The problem of checking if there exists*

1. $contain(T, T_1) = \begin{cases} true & T = T_1 \\ false & \text{otherwise} \end{cases}$
2. $contain(W_1 \# W_2, T) = contain(\text{if } C \text{ then } W_1 \text{ else } W_2, T) = contain(W_1 \otimes W_2, T) = contain(W_1 \mid W_2, T) =$
$contain(\text{while } (C)\{W_1\}W_2, T) = contain(W_1, T) \vee contain(W_2, T)$
3. $contain(|^k\ (W_1, \ldots, W_n), T) = contain(|^{\geq k}\ (W_1, \ldots, W_n), T) = contain(|^{\leq k}\ (W_1, \ldots, W_n), T)$
$= \begin{cases} true & \text{there exists a } W_i \text{ such that } contain(W_i, T) = true \\ false & otherwise \end{cases}$
4. $contain(T, T_1 \wedge T_2) = false$
5. $contain(W_1 \# W_2, T_1 \wedge T_2) = contain(\text{if } C \text{ then } W_1 \text{ else } W_2, T_1 \wedge T_2) = contain(W_1, T_1 \wedge T_2) \vee contain(W_2, T_1 \wedge T_2)$
6. $contain(W_1 \otimes W_2, T_1 \wedge T_2) = contain(W_1 \mid W_2, T_1 \wedge T_2) = contain(\text{while } (C)\{W_1\}W_2, T_1 \wedge T_2$
$= contain(W_1, T_1 \wedge T_2) \vee contain(W_2, T_1 \wedge T_2) \vee (contain(W_1, T_1) \wedge contain(W_2, T_2)) \quad \vee (contain(W_2, T_1) \wedge contain(W_1, T_2))$
7. $contain(|^{\geq k}\ (W_1, \ldots, W_n), T_1 \wedge T_2) = contain(|^k\ (W_1, \ldots, W_n), T_1 \wedge T_2)(k > 1) = contain(|^{\leq k}\ (W_1, \ldots, W_n), T_1 \wedge T_2)(k > 1)$
$= \begin{cases} true & \text{there exists a } W_i \text{ such that } contain(W_i, T_1 \wedge T_2) = true \vee \\ & \text{there exist } W_i \text{ and } W_j \text{ such that } contain(W_i, T_1) = true \wedge contain(W_j, T_2) = true \\ false & \text{otherwise} \end{cases}$
8. $contain(|^1\ (W_1, \ldots, W_n), T_1 \wedge T_2) = contain(|^{\leq 1}\ (W_1, \ldots, W_n), T_1 \wedge T_2) = \begin{cases} true & \text{there exists a } W_i \text{ such that } contain(W_i, T_1 \wedge T_2) = true \\ false & \text{otherwise} \end{cases}$

Fig. 7. The algorithm for proving Lemma III.15.

$p \in paths(W)$ such that p contains both tasks T_1 and T_2 is polynomial.

Proof: Figure 7 gives a polynomial algorithm for solving the problem. Rules 1 – 5 are straightforward. Rule 6 specifies that there exists $p \in paths(W_1 \otimes W_2)$ such that p contains both T_1 and T_2 if (1) there exists $p \in paths(W_1) \cup path(W_2)$ such that p contains both T_1 and T_2, (2) there exists $p_1 \in paths(W_1)$ and $p_2 \in paths(W_2)$ such that p_1 contains T_1 and p_2 contains T_2, or (3) there exists $p_1 \in paths(W_1)$ and $p_2 \in paths(W_2)$ such that p_1 contains T_2 and p_2 contains T_1. $\mid$ and while loop are handled using the same way. Rules 7 and 8 handle $|^k$, $|^{\leq k}$, and $|^{\geq k}$. The worst-case complexity of the algorithm is $O(|W|log|W|)$. ■

Theorem III.16 *WSP_U for workflows where each task appears only once, and RBAC policies containing all constraints is NP-complete.*

Proof: First, we show that the problem is in NP. Let $WU(W, \psi, U)$ be an instance of the WSP_U problem and A be a set of user-task assignments in which each task in W appears once. To prove that the problem is in NP, we show that checking if A is a solution to the problem can be done in polynomial time. It is easy to see that, for each $(u, T) \in A$, checking if there exists a role r such that ψ contains $(u, r) \in UA$ and $(T, r) \in PA$ can be done in $O(|\psi|)$. Below, we show that, it takes polynomial time to check if $\{(u, T) \mid (u, T) \in A \wedge p \text{ contains } T\}$ conforms to constraints in ψ, for every $p \in paths(W)$.

Binding constraints: For every binding constraint $= (T_1, T_2)$, we first check if there exists $p \in paths(W)$ such that p contains both T_1 and T_2. This can be done in polynomial time according to Lemma III.15. If not, then A satisfies the constraint. Otherwise, we check if T_1 and T_2 are assigned the same user in A; if so, A satisfies the binding constraint. Assume that there are b binding constraints. The worst-case complexity of the algorithm is $O(b|A||W|log|W|)$.

Separation of duty constraints: For every separation of duty constraint $\neq (T_1, T_2)$, we first check if there exists $p \in paths(W)$ such that p contains both T_1 and T_2. If not, then A satisfies the constraint. Otherwise, we check if T_1 and T_2 are assigned different users in A; if so, A satisfies the constraint. Assume that there are s separation of duty constraints. The worst-case complexity of the algorithm is $O(s|A||W|log|W|)$.

Cardinality constraints: A satisfies a cardinality constraint $Cardin(u, n)$ if there does not exist $p \in paths(W)$ such that $|\{T \mid T \in p \wedge (u, T) \in A\}| > n$. Let $S_0 = \{(u, 0) \mid u \in U\}$. Figure 8 provides an algorithm for computing the maximum number of tasks executed by each user in all execution paths. If there exist an execution path, a user u and $Cardin(u, m) \in \psi$ such that the maximum number of tasks executed by u is greater than m, then the cardinality constraint does not hold.

Rule 1 states that, given an atomic task T, if A contains (u, T), then we replaces $(u, 0)$ with $(u, 1)$, which means that u executes one task. Rule 2 states that, for every user u, the maximum number of tasks executed by u in $W_1 \# W_2$ is the maximum number of (1) the maximum number of tasks executed by u in W_1, and (2) the maximum number of tasks executed by u in W_2. if (C) then W_1 else W_2 is handled in the same manner. Let $|S_{max}| = |S_1| > |S_2| ? |S_1| : |S_2|$. $max(S_1, S_2)$ can be computed in $O(|S_{max}|log|S_{max}|)$ by first sorting S_1 and S_2 and then, for each user u, returns (u, n) in S_1 and S_2 that has a larger n.

In rule 3, the maximum number of tasks executed by a user in $W_1 \otimes W_2$ is the sum of the maximum number of tasks executed by the user in W_1 and the maximum number of tasks executed by the user in W_2. Similar to max, $sum(S_1, S_2)$ can be computed in $O(|S_{max}|log|S_{max}|)$ where $|S_{max}| = |S_1| > |S_2| ? |S_1| : |S_2|$. $\mid$ and while loop are handled similarly.

Rule 4 handles $|^k$ and $|^{\leq k}$. First, for every W_i, we compute and sort $wspusatnp(W_i)$. Next, for every user u, we sort all (u, n) in $wspusatnp(W_i)$ in descending order and compute the sum of the first k number. $kmaxsum(k, S_1, \ldots, S_m)$ can be computed in $|S_{kmax}|log|S_{kmax}| + |U|mlogm$ where $|S_{kmax}|$ is the largest $|S_i|$ for $1 \leq i \leq m$. $|^{\geq k}$ is handled similarly in Rule 5. Note that Rules 3 – 5 are correct only if each task appears once in the workflow. For example, consider the workflow $(T_1 \otimes T_2) \mid T_2$, in which task T_2 appears twice. Assume that $(u, T_1) \in A$ and $(u, T_2) \in A$, then u will execute two different tasks T_1 and T_2 in each workflow run. However, Rule 3 would return $\{(u, 3)\}$.

Since each task in the workflow is processed once, each operation (i.e., max, sum, $kmaxsum$) is performed at most

1. $wspusatnp(T, A) = \{(u, 1)\} \cup (S_0 \setminus \{(u, 0)\})$, where $(u, T) \in A$.
2. $wspusatnp(W_1 \# W_2, A) = wsat(\text{if } C \text{ then } W_1 \text{ else } W_2, A) = max(S_1, S_2)$, where
   - $S_1 = wspusatnp(W_1, A)$,
   - $S_2 = wspusatnp(W_2, A)$, and
   - $max(S_1, S_2) = \{(u, n) \mid (u, n_1) \in S_1, (u, n_2) \in S_2, n = (n_1 > n_2)?n_1 : n_2\}$
3. $wspusatnp(W_1 \otimes W_2, A) = wspusatnp(W_1 \mid W_2, A) = wspusatnp(\text{while } (C)\{W_1\}W_2, A) = sum(S_1, S_2)$, where
   - $S_1 = wspusatnp(W_1, A)$,
   - $S_2 = wspusatnp(W_2, A)$, and
   - $sum(S_1, S_2) = \{(u, n) \mid (u, n_1) \in S_1, (u, n_2) \in S_2, n = n_1 + n_2)\}$.
4. $wspusatnp(|^k\ (W_1, \ldots, W_m), A) = wspusatnp(|^{\leq k}\ (W_1, \ldots, W_m), A) = kmaxsum(k, wspusatnp(W_1, A), \ldots, wspusatnp(W_m, A))$ where
   - $kmaxsum(k, S_1, \ldots, S_m) = \{(u_i, kmax(k, N_i)) \mid u_i \in U \wedge N_i = \{n | (u_i, n) \in (S_1 \cup \ldots \cup S_m)\}$ and
   - $kmax(k, N_i)$ returns the sum of the maximum k numbers in N_i.
5. $wspusatnp(|^{\geq k}\ (W_1, \ldots, W_m), A) = kmaxsum(m, wspusatnp(W_1, A), \ldots, wspusatnp(W_m, A))$

Fig. 8. The algorithm for proving Theorem III.16.

$|W|$ times, and the size of $wspusatnp(W, A)$ is $|U|$, the worst-case complexity of the above algorithm is $O(|W||U|log|U| + |U||W|^2 log|W|)$. Therefore, the problem is in NP.

Below, we use an example to illustrate the above algorithm. Consider the workflow $(|^2\ (T_1, T_2, T_3)\#((T_4 \mid T_5) \otimes T_6)) \otimes T_7$ and a set of user-task assignments $A = \{(u_1, T_1), (u_2, T_2), (u_2, T_3), (u_1, T_4), (u_1, T_5), (u_2, T_6), (u_1, T_7)\}$. Assume that $Cardin(u_1, 2) \in \psi$ and $Cardin(u_2, 2) \in \psi$. The algorithm works as follows. First, we process $(|^2\ (T_1, T_2, T_3)\#((T_4 \mid T_5) \otimes T_6))$. $wspusatnp(|^2\ (T_1, T_2, T_3), A)$ returns $S_1 = \{(u_1, 1), (u_2, 2)\}$ and $wspusatnp((T_4 \mid T_5) \otimes T_6, A)$ returns $S_2 = \{(u_1, 2), (u_2, 1)\}$. We then compute $max(S_1, S_2)$, which is $\{(u_1, 2), (u_2, 2)\}$. Next, we computes $wspusatnp(T_7, A)$, which results in $S_3 = \{(u_1, 1), (u_2, 0)\}$. We then computes the sum of $max(S_1, S_2)$ and S_3, which results in $\{(u_1, 3), (u_2, 2)\}$. Since $Cardin(u_1, 2) \in \psi$, the cardinality constraint is violated. There are two paths that violate the cardinality constraint: $T_4 \cdot T_5 \cdot T_6 \cdot T_7$ and $T_5 \cdot T_4 \cdot T_6 \cdot T_7$.

The NP-hardness proof directly follows Theorem 6 in [29], which shows that WSP_U for workflows containing only $\otimes$ and $|$, and RBAC containing only separation of duty constraints is NP-hard. Therefore, the theorem holds. ■

D.2 Polynomial-Time Results for WSP_U

Theorem III.17 *WSP_U for workflows containing all control-flow patterns and RBAC policies that do not contain constraints is solvable in polynomial time.*

Proof: Let $U(W, \psi, U)$ be an instance of the problem. The problem is true iff all tasks of workflow W can be executed by some user under ψ. The worst-case complexity of the algorithm is $O(|W||\psi|)$. ■

Theorem III.18 *WSP_U for workflows containing only $\otimes$ and $|$, and RBAC policies containing only cardinality constraints is solvable in polynomial time.*

Proof: The proof directly follows Theorem III.4, because there is only one execution path in the workflow if the workflow contains only $\otimes$ and $|$. ■

Theorem III.19 *WSP_U for workflows containing only $\otimes$ and $|$, and RBAC policies containing both binding and cardinality constraints, where the number of binding constraints is less than a constant k, is solvable in polynomial time.*

Proof: The proof directly follows Theorem III.8, because there is only one execution path in the workflow if the workflow contains only $\otimes$ and $|$. ■

E. Universal Workflow Satisfiability with Task Constraints WSP_{UT}

WSP_{UT} asks if a set of users U together are able to complete the execution of all paths of a workflow W that satisfy a task constraint TC under the restrictions place by an RBAC policy ψ.

Definition 11 (Instance of WSP_{UT}) *An instance of WSP_{UT} is defined as $UT(W, \psi, U, TC)$, where W is a workflow, ψ is an RBAC policy, U is a set of users, and TC is a task constraint.*

Definition 12 (solution to WSP_{UT}) *Let $UTI = UT(W, \psi, U, TC)$ be a WSP_{UT} instance, TC be a task constraint, and $tasks(W)$ be a set of all tasks of workflow W. A set of user-task assignments $A = \{(u_1, T_1), \ldots, (u_n, T_n)\}$ is a solution to UTI if*

- $\{T_1, \ldots, T_n\} \subseteq tasks(W)$,
- $T_1 \neq \ldots \neq T_n$,
- *for each $(u, T) \in A$, there exists a role r such that ψ contains $(u, r) \in UA$ and $(T, r) \in PA$,*
- *for every $p \in paths(W)$ that satisfies TC, $\{(u, T) \mid (u, T) \in A \wedge p$ contains $T\}$ conforms to constraints in ψ.*

Theorem III.20 *WSP_{UT} for workflows containing all patterns, RBAC containing no constraints, and task constraints that do not contain $\wedge$ is solvable in polynomial time.*

Proof: Let $UTI = UT(W, \psi, U, TC)$ be a WSP_{UT} instance. The problem is true iff there does not exist $p \in paths(W)$ such that p satisfies TC and users in U cannot complete the execution of p. The algorithm is given below. First, the algorithm computes all execution paths of W that cannot be completed by users in U under ψ by computing $W' = W - elim(W, \psi)$ where $elim$ is defined in Figure 6. Next, the algorithm applies $tsatnoand$ in Figure 6 to check if there exists a path in W' that satisfies the task constraint. If so, the algorithm returns false;

otherwise, the algorithm returns true. When RBAC does not contain constraint, computing $W - elim(W)$ takes $O(|W||\psi)$ and computing $tsatnoand$ takes $O(|W||TC|)$. Therefore, the problem is solvable in polynomial time. ■

Theorem III.21 *WSP_{UT} for workflows containing only $\otimes$ and $|$, RBAC containing only binding and cardinality constraints where the number of binding constraints is less than a constant k , and task constraints containing both $\wedge$ and $\vee$ is solvable in polynomial time.*

Proof: Let $UT(W, \psi, U, TC)$ be a WSP_{UT} instance. The algorithm is given below. First, the algorithm checks if W satisfies TC as follows. For every T in TC, if T appears in W, then we replace T with true; otherwise, we replace T with false. W satisfies TC iff the resulting task constraint is true. If W satisfies TC, then the algorithm checks if the workflow satisfies the cardinality and binding constraints using the algorithm in Theorem III.8. Since checking if a task T appears in W takes $O(|W|)$, replacing T in TC with true/false takes $O(|TC|)$, and the worst-case complexity of the algorithm in Theorem III.8 is $O(|W||\psi| + k^2 log|W| + |U|^k(k|\psi| + |U|^{\frac{3}{2}}|W|^{\frac{5}{2}}))$, the worst-case complexity of the algorithm is $O(TC| + |W||\psi| + k^2 log|W| + |U|^k(k|\psi| + |U|^{\frac{3}{2}}|W|^{\frac{5}{2}}))$. ■

The following theorem is a corollary of Theorem III.21.

Theorem III.22 *WSP_{UT} is solvable in polynomial time for the following workflows and RBAC policies, and task constraints containing both $\wedge$ and $\vee$: (1) workflows containing only $\otimes$ and $|$, RBAC containing no constraints; (2) workflows containing only $\otimes$ and $|$, RBAC containing only binding constraints where the number of binding constraints is less than a constant k; and (3) workflows containing only $\otimes$ and $|$, RBAC containing only cardinality constraints.*

F. Minimum Role Satisfiability

Another challenge for managing access control policies is that, an administrator needs to determine what roles must be assigned to workflow users in order to complete a workflow. To address this, we consider two types of minimum role satisfiability analysis problem: the existential minimum role satisfiability problem MRS_E and the universal minimum role satisfiability problem MRS_U. MRS_E (MRS_U) computes a minimum set of roles that together can complete the execution of one (all) of the paths of a workflow W under the restriction placed by an RBAC policy ψ. While MRS_E and MRS_U provide information about the minimum set of roles for user assignment to ensure the completion of a workflow execution, in practice, we often need to assign users to roles in multiple minimum sets or assign multiple users to each role in one minimum set to satisfy RBAC constraints and allow the absence of some users during execution.

Because constraints in RBAC impose restrictions on users, instead of roles, they do not affect the minimum role satisfiability problem and hence are not considered in this section.

F.1 The Universal Minimum Role Satisfiability Problem

Definition 13 (Solution to MRS_U) *Let $I = MU(W, \psi)$ be an MRS_U instance. A set S_u of roles is a solution to I if*

- *for every task T in W, there exists a role $r \in S_u$ such that ψ contains $(T, r) \in PA$; and*
- *there does not exist $S'_u \subset S_u$ such that S'_u is a solution to I.*

Theorem III.23 *MRS_U is solvable in polynomial time.*

Proof: Let $MU(W, \psi)$ be an instance of the problem. Algorithm 2 gives a polynomial algorithm for solving the problem. First, the algorithm calls $compRoles(W, \psi)$ to compute $Rset$, which is a set of roles that have permission to execute all tasks of W, and $RTset$, which is the corresponding set of role-task assignments. Next, the algorithm removes the first element r from $Rset$ and calls $compTask(r, Rset, RTset)$ to check if the rest of roles in $Rset$ together have permission to execute all tasks assigned to r. If so, the algorithm assigns such tasks to corresponding roles; otherwise, the algorithm adds r to $MRset$ and repeat the above process. The algorithm returns $MRset$, which is a solution to MRS_U. Since it takes $O(|W|log|W|)$ to compute intersection of two sets in Line 16 and the size of $Tset$ is $|W|$, the worst-case complexity of $compRoles$ is $O(|W||\psi| + |W|^2 log|W|)$. Since the worst-case complexity of $compTask$ is $O(|\psi||W|)$, the worst-case complexity of the algorithm is $O(|W||\psi| + |W|^2 log|W| + |\psi|^2|W|)$. ■

Note that there may be more than one solution to the MRS_U problem. Algorithm 3 gives an algorithm for computing all solutions to MRS_U. First, the algorithm checks if a set R of all roles together have permission to complete the execution of all paths of workflow W (Line 5). If not, the algorithm returns $\emptyset$. Otherwise, the algorithm checks if R is a solution to the problem using function $minumim$ (Line 6). If so, the algorithm adds R to $MRall$; otherwise, for every subsets of R whose size is $|R| - 1$, the algorithm repeats the above process until all solutions are computed. In the worst-case, the algorithm needs to process all subsets of R. Since the number of all subsets of R is $2^{|R|}$ and the worst-case complexity of $minimum$ is $O(|R|(|\psi| + |W|log|W|))$, the worst-case complexity of the algorithm is $O(2^{|R|}|R|(|\psi|+|W|log|W|))$.■

F.2 The Existential Minimum Role Satisfiability Problem

Definition 14 (Solution to MRS_E) *Let $I = ME(W, \psi)$ be an MRS_E instance. A set S_e of roles is a solution to I if*

- *there exists $p \in paths(W)$ such that for every task T in p, there exists a role $r \in S_e$ such that ψ contains $(T, r) \in PA$; and*
- *there does not exist $S'_e \subset S_e$ such that S'_e is a solution to I.*

Let $ME(W, \psi)$ be an instance of the MRS_E problem. Algorithm 4 gives an algorithm for solving the problem. The algorithm first calls $compRoles$ (defined in Algorithm 2) to compute a set $Rset$ of roles that have permission to execute of all paths of the workflow and the corresponding set of user-task assignments. Next, the algorithm computes a solution

Algorithm 2 Algorithm for computing one solution to MRS_u.

Procedure $mrsu(W, \psi)$
$(Rset, RTset) = compRoles(W, \psi)$; $MRset = \emptyset$;
while $Rset \neq \emptyset$ **do**
  remove the first element r from $Rset$;
  $RTset_1 = compTask(r, Rset, RTset)$;
  if $RTSet_1 = \emptyset$ **then** $MRset = MRset \cup \{r\}$;
  else
    $RTset = (RTset - \{(r,T) \mid (r,T) \in RTset\}) \cup RTset_1$;
  end if
end while
return $MRset$;

Procedure $compRoles(W, \psi)$
$Tset$ = all tasks in W; $Rset = RTset = \emptyset$;
while $Tset \neq \emptyset$ **do**
  remove the first element T from $Tset$;
  if there exists a role r such that $(T, r) \in PA$ **then**
    $Rset = Rset \cup \{r\}$; $Tset_1 = \{T_1 \mid (T_1, r) \in PA\} \cap Tset$;
    $RTset = RTset \cup \{(r, T_1) \mid T_1 \in Tset_1\}$;
  $Tset = Tset - Tset_1$;
  else return $(\emptyset, \emptyset)$; **end if**
end while
return $(Rset, RTset)$;

Procedure $compTask(r, Rset, RTset)$
$Tset_r = \{T \mid (r,T) \in RTset\}$; $RTset_1 = \emptyset$;
while $Tset_r \neq \emptyset$ **do**
  remove the first element T from $Tset_r$;
  if there exists $r' \in Rset$ such that $(T, r') \in PA$ **then**
  $RTset_1 = RTset_1 \cup \{(r', T)\}$;
  else return $\emptyset$; **end if**
end while
return $RTset_1$;

Algorithm 3 Algorithm for computing all solutions to MRS_u.

Procedure $mrsuAll(W, \psi)$
$R = \{\{r \mid (r,T) \in \psi\}\}$; $MRall = \emptyset$; $S = R$;
while $S \neq \emptyset$ **do**
  remove the first set $rset$ from S;
  if $\{T \mid (r,T) \in \psi \wedge r \in rset\} = tasks(W)$ **then**
    if $minimum(rset)$ **then** $MRall = MRall \cup \{rset\}$;
    else $S = S \cup \{rset \setminus \{r\} \mid r \in rset\}$; **end if**
  end if
end while
return $MRall$;
Procedure $minimum(rset)$
if there exists $r \in rset$ such that $\{T \mid (T, r') \in \psi \wedge r' \in (rset \setminus \{r\})\} = tasks(W)$ **then**
  return false;
else return true; **end if**

Algorithm 4 Algorithm for computing one solution to MRS_E.

Procedure $mrse(W, \psi)$
$(Rset, \_) = compRoles(W, \psi)$;
$allsub = \{sub \mid sub \subseteq Rset\}$;
sort $allsub$ in ascending order;
while $allsub \neq \emptyset$ **do**
  remove the first element sub_1 from $allsub$;
  if $satRoles(W, sub_1, \psi) = true$
  then return sub_1; **end if**
end while

to the problem from $Rset$. Function $satRoles(W, sub_1, \psi)$ checks if there exists $p \in path(W)$ such that roles in sub_1 together can execute all tasks in p, an algorithm similar to that of Theorem III.6 (with users replaced with roles). Since the worst-case complexity of $compRoles$ and $satRoles$ are $O(|W||\psi| + |W|^2 log|W|)$ and $O(|\psi||W|)$, respectively, the worst-case complexity of the algorithm is $O(|W||\psi| + |W|^2 log|W| + 2^{|R|}|\psi||W|)$.

To compute all solutions to MRS_E, we can apply the algorithm for computing all solutions to MRS_U to every path of the workflow.

IV. Related Work

Several researchers have considered the workflow satisfiability problem (WSP). Wang and Li [29] presented algorithms and complexity results for WSP of R^{2}BAC, which extends RBAC with binary relations between users. They have also shown that the workflow resilient problem is intractable. Crampton et al. [9] considered the workflow satisfiability problem in the presence of delegation under constrained RBAC. In [8], they have also presented fixed parameterized tractable algorithms for the workflow satisfiability problem. Our work is different from theirs as follows. Firstly, none of them considered cardinality constraints presented in this paper, while our work did not consider some of the constraints they considered, such as $\exists$ and $\forall$ in [29]. Secondly, in their work, the workflow is specified as a partial order of tasks. In our work, the workflow is specified using control-flow patterns; some of the patterns such as exclusive or $\#$, multiple split $|^k$, multiple split with upper bound $|^{\leq k}$, multiple split with lower bound $|^{\geq k}$, and while loop cannot be represented using partial order among tasks. Thirdly, because they did not consider $\#$ and $|^k$, the workflow they considered contains only one execution path. However, when considering $\#$ and $|^k$, the number of execution paths of the workflow may be exponential to the size of the workflow, which significantly complicates the problem. Fourthly, they did not consider minimum role satisfiability problems, and we do not consider the workflow resilient problem and the delegation.

A number of researchers have also proposed techniques for modeling, analysis, and verification of workflows [32], [1], [2], [7], [27], [12], [11], [24]. However, they did not consider the workflow satisfiability problem. Luo et al [18] proposed algorithms and complexity results for analysis of workflow provenance dependencies; the analysis problems they considered are different from ours.

Analysis of access control policies [15], [22], [25], [14], [23], [13] has also been recognized as an important problem, which checks whether an access control policy conforms to given security properties (e.g. reachability, availability, containment). However, none of them considered the analysis problems considered in this paper.

V. Conclusion and Future Work

In this paper, we present algorithms and complexity results for solving various analysis problems related to the workflow

satisfiability. We have shown that several subclasses of existential and universal workflow satisfiability analysis problems are NP-complete or NP-hard. We have also identified a few restrictions on workflow patterns and RBAC policies under which these problems are solvable in polynomial time.

One direction for future work is to develop more efficient algorithms for computing all solutions to universal and existential minimum role satisfiability problems. In addition, delegation causes temporary transfer or grant of privileges to other users. It introduces a new set of constraints that interact in subtle ways with the workflow and other authorization constraints. We plan to investigate this as part of our future work. Finally, we plan to evaluate the effectiveness and performance of our algorithms on real world workflows.

Acknowledgement: This work was supported in part by NSF Grant CNS-0855204.

REFERENCES

[1] W. Aalst and A. Hofstede. Verification of workflow task structures: A petri-net-based approach. *Information Systems*, 25(1):43–69, 2000.

[2] N. R. Adam, V. Atluri, and W. Huang. Modeling and analysis of workflows using petri nets. *Intelligent Information Systems*, 10(2):131–158, 1998.

[3] C. A. Ardagna, S. D. C. di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, and M. Verdicchio. Expressive and deployable access control in open web service applications. *IEEE Transactions on Services Computing*, 4(2):96–109, 2011.

[4] Y. Badr, F. Biennier, and S. Tata. The integration of corporate security strategies in collaborative business processes. *IEEE Transactions on Services Computing*, 4(3):243–254, 2011.

[5] E. Bertino, E. Ferrari, and V. Atluri. A flexible model supporting the specification and enforcement of role-based authorization in workflow management systems. In *ACM Workshop on Role-Based Access Control*, pages 1–12, 1997.

[6] A. Chebotko, S. Lu, S. Chang, F. Fotouhi, and P. Yang. Secure abstraction views for scientific workflow provenance querying. *IEEE Transactions on Services Computing*, 3(4):322–337, 2010.

[7] Y. Choi, X. Zhao, and K. Han. Hierarchical reachability analysis for workflow-nets. In *International Conference on Computer Supported Cooperative Work in Design*, pages 556–561, 2006.

[8] J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity of the workflow satisfiability problem. In *ACM conference on Computer and communications security*, pages 857–868, 2012.

[9] J. Crampton and H. Khambhammettu. Delegation and satisfiability in workflow systems. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, 2008.

[10] E. Damiani, S. Proctor, and A. Singhal. Guest editorial: Security and dependability in SOA and business processes. *IEEE Transactions on Services Computing*, 4(4):255–256, 2011.

[11] H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1996.

[12] R. Eshuis and R. Wieringa. Verification support for workflow design with UML activity graphs. In *International Conference on Software Engineering*, pages 166–176, 2002.

[13] M. Gofman, R. Luo, J. He, Y. Zhang, and P. Yang. Incremental information flow analysis of role based access control. In *International Conference on Security and Management*, pages 397–403, 2009.

[14] M. Gofman, R. Luo, and P. Yang. User-role reachability analysis of evolving administrative role based access control. In *European Symposium on Research in Computer Security (ESORICS)*, 2010.

[15] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[16] P. Hung and K. Karlapalem. A secure workflow model. In *Proc. of the Australasian information security workshop conference on ACSW frontiers*, Adelaide, Australia, 2003.

[17] L. Lowis and R. Accorsi. Vulnerability analysis in SOA-based business processes. *IEEE Transactions on Services Computing*, 4(3):230–242, 2011.

[18] R. Luo, P. Yang, S. Lu, and M. I. Gofman. Analysis of scientific workflow provenance access control policies. In *The 9th IEEE International Conference on Services Computing (SCC)*, 2012.

[19] S. Micali and V. V. V. An o(v(1/2)e) algorithm for finding maximum matching in general graph. In *Foundations of Computer Science, 21st Annual Symposium*, 1980.

[20] F. Montagut and R. Molva. Bridging security and fault management within distributed workflow management systems. *IEEE Transactions on Services Computing*, 1(1):33–48, 2008.

[21] N. Russell, Arthur, W. M. P. van der Aalst, and N. Mulyar. Workflow control-flow patterns: A revised view. Technical report, BPMcenter.org, 2006.

[22] R. Sandhu. Transaction control expressions for separation of duties. In *Proc. of the Fourth Computer Security Applications Conference*, pages 282–286, 1988.

[23] A. Sasturkar, P. Yang, S. Stoller, and C. Ramakrishnan. Policy analysis for administrative role based access control. *Theoretical Computer Science*, 412(44):6208–6234, 2011.

[24] M. P. Singh. Semantical considerations on workflows: An algebra for intertask dependencies. In *In Proceedings of the International Workshop on Database Programming Languages*, 1995.

[25] S. Stoller, P. Yang, C. R. Ramakrishnan, and M. Gofman. Efficient policy analysis for administrative role based access control. In *ACM Conference on Computer and Communication Security (CCS)*, pages 445–455. ACM Press, 2007.

[26] S. Tata, K. Klai, and N. O. A. M'Bareck. Coopflow: A bottom-up approach to workflow cooperation for short-term virtual enterprises. *IEEE Transactions on Services Computing*, 1(4):214–228, 2008.

[27] F. L. Tiplea and D. C. Marinescu. Structural soundness of workflow nets is decidable. *Inf. Process. Lett.*, 96(2):54–58, 2005.

[28] S. Tjoa, S. Jakoubi, G. Goluch, G. Kitzler, S. Goluch, and G. Quirchmayr. A formal approach enabling risk-aware business process modeling and simulation. *IEEE Transactions on Services Computing*, 4(2):153–166, 2011.

[29] Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur.*, 13(4):40:1–40:35, 2010.

[30] S. Wu, A. Sheth, J. Miller, and Z. Luo. Authorization and access control of application data in workflow systems. *Journal of Intelligent Information Systems*, 18(1):71–94, 2002.

[31] J. Xu, D. Zhang, L. Liu, and X. Li. Dynamic authentication for cross-realm SOA-based business processes. *IEEE Transactions on Services Computing*, 5(1):20–32, 2012.

[32] P. Yang, S. Lu, M. Gofman, and Z. Yang. Information flow analysis of scientific workflows. *Journal of Computer and System Sciences (JCSS)*, 76(6):390–402, 2010.

Ping Yang is an Assistant Professor in Department of Computer Science at State University of New York at Binghamton. She received her PhD from Stony Brook University. Her research focuses on security, privacy, workflow, and formal methods.

Xing Xie is a Ph.D. student in the Computer Science Department at Colorado State University. His research interests include security, database systems, workflow, and formal methods.

Indrakshi Ray is an associate professor in the Computer Science Department at Colorado State University. She obtained her Ph.D. from George Mason University. Her research interests include security and privacy, database, e-commerce and formal methods.

Shiyong Lu is an Associate Professor in the Department of Computer Science at Wayne State University and the Director of the Scientificc Workflow Research Laboratory. He received his Ph.D. in computer science from Stony Brook University. His research interests focus on scientific workflows and their applications.