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

Ping Yang 1, Xing Xie 2, Indrakshi Ray 2, Shiyong Lu 3

1Department of Computer Science, State University of New York at Binghamton, Binghamton, New York, 13902, USA
2Department of Computer Science, Colorado State University, Fort Collins, CO, 80523-1873, USA
3Department 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 T1) and then registers for the selected courses (task T2). The registration needs to be approved by the graduate director. Tasks T3 and T4 are used to approve or disapprove the registration, respectively; these two tasks are connected through exclusive or denoted by $\oplus$, which means that either T3 or T4, 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 T5) and inform the student that the registration is approved (task T6); otherwise, the process will inform the student that the registration is not approved (task T0). In this workflow, T1 is a composite task consisting of subtasks $T_{1_1}, \ldots, T_{1_n}$, 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
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.

\[
W := \begin{array}{c}
T | W_1 \otimes W_2 | W_1 \# W_2 | W_1 | W_2 \\
| k (W_1, \ldots, W_n) | | \leq k (W_1, \ldots, W_n) \\
| \geq k (W_1, \ldots, W_n) | \text{if } C \text{ then } W_1 \text{ else } W_2 \\
| \text{while}(C)\{W_1\}W_2
\end{array}
\]

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. \( | \) denotes parallel split with synchronization; \( W_1 | W_2 \) specifies that both \( W_1 \) and \( W_2 \) need to be completed in order to execute the next task. \( | k (W_1, \ldots, W_n) \), \( \geq k (W_1, \ldots, W_n) \) and \( \leq k (W_1, \ldots, W_n) \) \((k \leq n)\) 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 \( k \) \( W_i \)'s 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 \( \text{while}(C)\{W_1\}W_2 \) specifies loops. For example, the workflow in Figure 1 can be specified as \( \leq 4 \) \((T_{11}, \ldots, T_{1n}) \otimes T_2 \otimes ((T_3 \otimes (T_5 | T_6)) \# (T_4 \otimes T_6))\).

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
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 \(n\) workflow paths and then interleaving the execution of the selected workflows. Rules 6 and 7 specify that \(path(\lceil k \rceil (W_1, \ldots, W_n))\) contains all execution paths of \([1] (W_1, \ldots, W_n)\), and \([k] (W_1, \ldots, W_n)\), and \(path(\lceil k \rceil (W_1, \ldots, W_n))\) contains all execution paths of \([k] (W_1, \ldots, W_n)\), and \([n] (W_1, \ldots, W_n)\). Rule 8 defines execution paths of workflow \(W\), where each path is either an execution path of \(W_1\) followed by 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 \(T_1 \otimes (T_2 \# T_3) \otimes (T_4, T_5, T_6)\) is \\{
(T_1, T_2, T_3) \{T_1, T_2, T_5\} \\
{T_1, T_2, T_6} \{T_1, T_3, T_4\} \{T_1, T_3, T_6\} \{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 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:** \(Card(u, n)\), which specifies that a user \(u\) can execute at most \(n\) different workflow tasks in each execution run of the workflow.
- **Binding constraints:** \(\{T_1, T_2\}\) \(\text{called equality constraints in [29]}\), 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:** \(\neq \{T_1, T_2\}\) \(\text{called inequality constraints in [29]}\), 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:**

\[ TC := T \mid \neg T \mid TC_1 \lor TC_2 \mid TC_1 \land TC_2 \]

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

III. THE WORKFLOW SATISFIABILITY PROBLEM

Let \(W\) be a workflow, \(\psi\) 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 \(\psi\).
- **Existential workflow satisfiability with task constraint \(WSP_{ET} (WSP_{UT})\)**, 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 \(\psi\).
- **Universal workflow satisfiability with task constraint \(WSP_{UT} (WSP_{ET})\)**, which asks if users in \(U\) together can complete the execution of all paths of \(W\) that satisfy \(TC\), under the restrictions imposed by \(\psi\).
- **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 \(\psi\).
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 $P_i \in \{\otimes, \#, \}, k \leq k$, if, while: the workflow contains only patterns $P_1, P_2, \ldots$ and $P_n$.

- Restricting access control policies
  - all constraints: the policy contains all constraints.
  - no constraint: the policy does not contain constraints.
  - $C_1, C_2, \ldots, C_n$ where $C_i \in \{B, C, S\}$, and $B, C, S$ represent binding, cardinality, and separation of duty constraints, respectively: the policy contains only constraints $C_1, \ldots$ and $C_n$.

- Restricting task constraints
  - $\land, \lor$: the task constraint contains both $\land$ and $\lor$.
  - $\land$: the task constraint contains only $\land$.
  - $\lor$: the task constraint contains only $\lor$.

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 $\lor$ makes $WSP_{ET}$ tractable.

B. Existential Workflow Satisfiability Problem: $WSP_E$

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

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, $\psi$ 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 $\psi$ contains $(u, r) \in UA$ and $(T, r) \in PA$, and
- there exists $p \in \text{paths}(W)$ such that $
\{(u, T) \mid (u, T) \in A \land T \in \text{tasks}(p)\}$ conforms to constraints in $\psi$.

$\{(u, T) \mid (u, T) \in A \land T \in \text{tasks}(p)\}$ 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 $F = F_1 \land F_2 \land \ldots \land F_n$ 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:
  - $\land$ is mapped to $\otimes$,
  - $\lor$ is mapped to $\#$,
  - each positive literal $l_i$ is mapped to a task $T_i$, and
  - each negative literal $-l_i$ is mapped to a task $T'_i$.

- The RBAC policy $\psi$ 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 $\psi$. If $l_i$ appears negatively in $F$, then $(u_i, r_i) \in UA$ and $(T'_i, r_i) \in PA$ are added to $\psi$.
  - If a literal $l_i$ appears both positively and negatively in $F$, then a cardinality constraint $\text{Cardin}(u_i, 1)$ is added to $\psi$, 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 $p \in \text{paths}(W)$ such that a set of users $\{u_1, \ldots, u_m\}$ together can complete the execution of all tasks in $p$ under $\psi$. The worst-case complexity of the reduction is $O(|F|)$. 

Let $W = W_1 \ldots \otimes W_t$. Below, we show that $F$ is only once. It is also not clear if the problem is in NP.

The following theorem is a corollary of Theorem III.1.

Theorem III.2. WSP is NP-hard for the following:

1. workflows containing only control-flow patterns and RBAC policies containing only cardinality constraints.

Proof. Let $E(W, \psi, U)$ be an instance of the problem. Below, we give a polynomial algorithm which reduces $WSP$ to a problem is still NP-hard if task appears in the workflow more than once and it is not clear if the problem is still NP-hard if task appears in the workflow more than once.

Below, we give a polynomial algorithm which reduces $WSP$ to a problem is still NP-hard if task appears in the workflow more than once.

Therefore, if task appears in the workflow more than once it is not clear if the problem is still NP-hard if task appears in the workflow more than once.
First, we construct a bipartite graph $G_b = (S_u \cup S_t, E)$, 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 $\psi$ 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 $\psi$ contains a cardinality constraint $\text{Cardin}(u, n)$. If $\psi$ 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 $(T_1 | T_3) \odot T_2 \odot T_4 \odot T_5$. Assume that there are two users $u_1$ and $u_2$, $u_1$ has permission to execute tasks $T_1$, $T_2$, and $T_3$, and $u_2$ has permission to execute $T_3$, $T_4$ and $T_5$. Also, assume that $\text{Cardin}(u_1, 2) \in \psi$ and $\text{Cardin}(u_2, 3) \in \psi$.

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: $V_{u_1} \rightarrow V_{T_1}$, $V_{u_1} \rightarrow V_{T_2}$, $V_{u_1} \rightarrow V_{T_3}$, $V_{u_2} \rightarrow V_{T_1}$, $V_{u_2} \rightarrow V_{T_2}$, $V_{u_2} \rightarrow V_{T_3}$, $V_{u_2} \rightarrow V_{T_4}$, $V_{u_2} \rightarrow V_{T_5}$. Next, we duplicate $V_{u_1}$ once which results in $V_{u_1}''$, and duplicate $V_{u_2}$ 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:

![Bipartite Matching](image)

**Algorithm 1** Algorithm for solving $\text{WSP}_E$ in Theorem III.5.

1: Procedure $\text{wspsatall}(W, \psi, U)$
2: $s = \emptyset$
3: for all task $T$ in workflow $W$ do
4: $\text{auth}(T) = \{u \mid \text{there exists a role } r \text{ such that } (u, r) \in UA \land (T, r) \in PA\}$
5: end for
6: for all $\{T_1, T_2\} \in \psi$ do
7: $\text{flag} = 0$
8: for all $e \in s$ do
9: if $T_1 \in e$ then
10: $e_1 = e \cup \{T_2\}$; $s = (s \setminus \{e\}) \cup \{e_1\}$; $\text{flag} = 1$
11: else if $T_2 \in e$ then
12: $e_2 = e \cup \{T_1\}$; $s = (s \setminus \{e\}) \cup \{e_2\}$; $\text{flag} = 1$
13: end if
14: end for
15: if $\text{flag} == 0$ then $s = s \cup \{(T_1, T_2)\}$; end if
16: end for
17: for all $e \in s$ do
18: $\text{intersect} = \emptyset$
19: for all $T \in e$ do
20: $\text{intersect} = \text{intersect} \cap \text{auth}(T)$; end for
21: if $\text{intersect} == \emptyset$ then return false;
22: else for all $T \in e$ do $\text{auth}(T) = \text{intersect}$; end for
23: end for

First, we compute, for each task $T$, a set $\text{auth}(T)$ 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 $\{T_1, \ldots, T_n\} \in s$, we replace $\text{auth}(T_1)$, ..., and $\text{auth}(T_n)$ with $\text{auth}(T_{i_1}) \cap \ldots \cap \text{auth}(T_{i_n})$ (Lines 17 – 22). Let $b$ be the number of binding constraints. Computing initial values of $\text{auth}(T)$ (Lines 3–5) takes $O(|W||\psi|)$, computing $s$ takes $O(b^2|\log|W|))$, and computing intersection of $\text{intersect}$ and $\text{auth}(T)$ (Line 19) takes $|U||\log|U|\). Therefore, the worst-case complexity of the algorithm is $O(|W||\psi| + b|W||\log|W| + b||U||\log|U|)$.

**Theorem III.6** $\text{WSP}_E$ 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 $\text{wspsatall}(W, \psi, U)$).

Rules 1 – 4 are straightforward. Rule 5 handles $|k^1| \leq |W_1, \ldots, W_n|$ and $|k^2| \leq |W_1, \ldots, W_n|$; if there exist $\{W^1, \ldots, W^k\} \subseteq \{W_1, \ldots, W_n\}$ and $p_i \in \text{paths}(W_i)$ such that, for every $i$, tasks $\{p_i\} \subseteq \{T \mid (u, r) \in UA \land (T, r) \in PA\}$ are in $\psi$ 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** $\text{WSP}_E$ for workflows containing $\odot$, $\otimes$, 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. \( \text{wpesatall}(T, U, \psi) = \begin{cases} \text{true} & \text{if there exists } u \in U \text{ and a role } r \text{ such that } \psi \text{ contains } (u, r) \in U A \land (T, r) \in PA \\ \text{false} & \text{otherwise} \end{cases} \)

2. \( \text{wpesatall}(W_1 \# W_2, U, \psi) = \text{wpesatall}(W_1, U, \psi) \lor \text{wpesatall}(W_2, U, \psi) \)

3. \( \text{wpesatall}(W_1 \times W_2, U, \psi) = \text{wpesatall}(W_1 | W_2, U, \psi) \lor \text{wpesatall}(W_1, U, \psi) \land \text{wpesatall}(W_2, U, \psi) \)

4. \( \text{wpesatall}(\text{while } (C)(W_1) W_2, U, \psi) = \text{wpesatall}(W_1, U, \psi) \)

5. \( \text{wpesatall}(T_i (W_1, \ldots, W_n), U, \psi) = \text{wpesatall}(|\mathcal{T}(W_1, \ldots, W_n), U, \psi) \)

6. \( \text{wpesatall}(| \mathcal{T}(W_1, \ldots, W_n), U, \psi) = \begin{cases} \text{true} & \text{if } \{W_i | \text{wpesatall}(W_i, U, \psi) == \text{true} \} \geq k \\ \text{false} & \text{otherwise} \end{cases} \)

---

**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 \( V_{T_i} \) for \( T_i \neq \ldots \neq T_n \) (instead of a vertex for each \( T_i \)) since only one of the \( T_i \)s 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 \( V_{T_i} \). 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 \( T_i \neq \ldots \neq T_n \) in every workflow run if the cardinality constraint for the user is \( \text{Cardin}(u, m) \). 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 \( O(|U|^2 |W|^2) \), the worst-case complexity of the algorithm is \( O(|U|^2 |W|^2) \).

Below, we show that, when the number of binding constraints is limited to a constant \( k \), \( \text{WSP}_E \) is solvable in polynomial time.

**Theorem III.8** \( \text{WSP}_E \) for workflows containing \( \times \) 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 \( \psi \) contains \( \text{Cardin}(u, n) \), then we replace \( \text{Cardin}(u, n) \) with \( \text{Cardin}(u, n - m) \). Finally, we apply the algorithm in Theorem III.4 to compute the maximum matching for users in \( U \setminus \{u \ | \ \text{Cardin}(u, 0) \in \psi \} \) 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 \( O(|U|^2 |W|^2) \), and we need to try at most \(|U|^k \) combinations of users, the worst-case complexity of the algorithm is \( O(|W||\psi| + k^2 \log |W| + |U|^k (k|\psi| + |U|^2 |W|^2)) \).

**C. Existential Workflow Satisfiability with Task Constraints \( \text{WSP}_{ET} \)**

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

**Definition 6** (Task constraint satisfiability) Let \( W \) be a workflow, \( p \in \text{paths}(W) \), \( TC \) be a task constraint, and \( TC' = \text{subst}(TC, \{(T \mapsto \text{true}|T \in p) \cup \{T \mapsto \text{false}|T \notin p \} \cup \{\neg T \mapsto \text{true}|T \notin p \} \cup \{\neg T \mapsto \text{false}|T \in p \}) \), where \( \text{subst}(TC, \{t_1 \mapsto v_1, \ldots, t_n \mapsto v_n\}) \) 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 \( \text{WSP}_{ET} \) problem and a solution to the \( \text{WSP}_{ET} \) problem.

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

**Definition 8** (Solution to \( \text{WSP}_{ET} \)) Let \( ET(W, \psi, U, TC) \) be a \( \text{WSP}_{ET} \) instance, \( TC \) be a task constraint, \( p \in \text{paths}(W) \), and \( \text{tasks}(p) \) 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 \( ET(W, \psi, U, TC) \) if

- \( T_1, \ldots, T_n \) are tasks in \( W \),
- \( T_i \neq \ldots \neq T_n \),
- for each \( (u, T) \in A \), there exists a role \( r \) such that \( \psi \) contains \( (u, r) \in U A \) and \( (T, r) \in PA \), and
- there exists \( p \in \text{path}(W) \) such that
  - \( \text{tasks}(p) \subseteq \{T_1, \ldots, T_n\} \),
  - \( p \) satisfies \( TC \), and
  - \( \{(u, T) | (u, T) \in A \land T \in \text{tasks}(p)\} \text{ conforms to constraints in } \psi \).

**C.1. Proofs for NP-Hardness Results of \( \text{WSP}_{ET} \)**

**Theorem III.9** \( \text{WSP}_{ET} \) for workflows containing all patterns, RBAC policies containing no constraints, and task constraint containing both \( \land \) and \( \lor \) is NP-hard.
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 $F = F_1 \land F_2 \land \ldots \land F_n$ be a 3-CNF formula. We construct a $\text{WSP}_{ET}$ instance $E(W, \psi, U, TC)$ 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 $W = (\langle T_1 \# T' \rangle \ldots \langle T_m \# T_m' \rangle)$ is constructed from $F$, where $m$ is the number of literals in $F$, all $T_i$'s do not appear in $TC$, and $T_i \neq \ldots \neq T_m'$.
- $U = \{ u \}$
- The RBAC policy $\psi$ is constructed as follows. First, we add $(u, r) \in UA$ to $\psi$. Next, for every $T_i$ and $T_i'$ in $W$, we add $(T_i, r) \in PA$ and $(T_i', r) \in PA$ to $\psi$.

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

Let $TC = C_1 \land \ldots \land C_m$. Below, we show that $\hat{F}$ is satisfiable if $wsp_{et}$ has a solution.

Proof for “only if”: Assume that $F$ is true under a set of assignments $\{(t_1, v_1), \ldots, (t_m, v_m)\}$. We show that the path $p$ of $W$, constructed as follows, satisfies $TC$ under $\psi$: 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 $p \in \text{paths}(W)$ that satisfies $TC$. Then $p$ satisfies all $C_i$s. 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 $\land$ and $\lor$ 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 $\land$ is solvable in polynomial time.

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

For example, consider the workflow $W = T_1 \#(T_2 | T_3)$, the RBAC policy $\psi = \{(u,r) \in UA, (T_1,r) \in PA, (T_2,r) \in PA\}$, the task constraint $T = T_1 \lor T_2$, and $U = \{u\}$. Since no user has permission to execute $T_3$, $T_2 | T_3$ cannot be completed by any user and hence $elim(W,U,\psi)$ returns $T_1$. Next, we compute $tsatnoand(T_1, \neg T_1 \lor T_2)$, 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 $\land$ and $\lor$ is solvable in polynomial time.
1. contain(T,T) = \{true \text{ if } T = T_1 \}
2. contain(W_1 \oplus W_2, T) = contain(W_1, T) \land contain(W_2, T)
3. contain(1, W_1, \ldots, W_n) \land T) = contain(\leq k, W_1, \ldots, W_n) \land T) \lor contain(1, W_1, \ldots, W_n)
4. contain(T, T_1 \land T_2) = contain(T_1 \land T_2)
5. contain(W_1 \oplus W_2, T_1 \land T_2) = contain(W_1 \oplus W_2, T_1 \land T_2)
6. contain(W_1 \oplus W_2, T_1 \land T_2) = contain(W_1 \oplus W_2, T_1 \land T_2)
7. contain(1, W_1, \ldots, W_n) \land T_1 \land T_2) = contain(1, W_1, \ldots, W_n, T_1 \land T_2) \land k \geq 1)
8. contain(W_1 \oplus W_2, T_1 \land T_2) = contain(1, W_1, \ldots, W_n, T_1 \land T_2) \land contain(\leq k, W_1, \ldots, W_n, T_1 \land T_2)

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

Proof: Figure 7 gives a polynomial algorithm for solving the problem. Rules 1 – 5 are straightforward. Rule 6 specifies that there exists p ∈ paths(W_1 ⊕ W_2) such that p contains both T_1 and T_2 if (1) there exists p ∈ paths(W_1) \cup paths(W_2) such that p contains both T_1 and T_2, (2) there exists p_1 ∈ paths(W_1) and p_2 ∈ paths(W_2) such that p_1 contains T_1 and p_2 contains T_2, or (3) there exists p_1 ∈ paths(W_1) and p_2 ∈ paths(W_2) such that p_1 contains T_2 and p_2 contains T_1. 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, ψ, 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) ∈ A, checking if there exists a role r such that ψ contains (u, r) ∈ UA and (T, r) ∈ PA can be done in O(|ψ|).

Binding constraints: For every binding constraint = (T_1, T_2), we first check if there exists p ∈ 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 ≠ (T_1, T_2), we first check if there exists p ∈ 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 ∈ paths(W) such that |\{T | T ∈ p \land (u, T) ∈ A\}| > n. Let S_0 = \{\{u, 0\} | u ∈ 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) ∈ ψ 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 replace (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 ⊕ 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|, and while loop are handled similarly.

Rule 4 handles k and |k|. For every user u, we compute and sort wspusatnp(W_u). Next, for every user u, we sort all (u, n) in wspusatnp(W_u) in descending order and compute the sum of the first k number. kmaxsum(k, S_1, \ldots, S_m) can be computed in |S_{max}|log|S_{max}| + U|log|q where |S_{max}| is the largest |S_i| for 1 ≤ i ≤ m. |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 ⊗ T_2) | T_3, in which task T_3 appears twice. Assume that (u, T_1) ∈ A and (u, T_2) ∈ 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
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 $\psi$. The worst-case complexity of the algorithm is $O(|W||\psi|)$. ■

Theorem III.18 WSP$_U$ for workflows containing only $\otimes$ and $\uparrow$, 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 $\uparrow$. ■

Theorem III.19 WSP$_U$ for workflows containing only $\otimes$ and $\uparrow$, 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 $\uparrow$. ■

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 placed by an RBAC policy $\psi$.

Definition 11 (Instance of WSP$_{UT}$) An instance of WSP$_{UT}$ is defined as $UT(W,\psi,U,TC)$, where $W$ is a workflow, $\psi$ 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 $\psi$ contains $(u,r)$ in $UA$ and $(T,r) \in PA$,
- for every $p \in$ paths($W$) that satisfies $TC$, R {$(u,T) | (u,T) \in A \land p \text{ contains } T$} conforms to constraints in $\psi$.

Theorem III.20 WSP$_{UT}$ for workflows containing all patterns, RBAC containing no constraints, and task constraints that do not contain $\land$ 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 $\psi$ by computing $W' = W - \text{elim}(W,\psi)$ where $\text{elim}$ is defined in Figure 6. Next, the algorithm applies $\text{tsatnoand}$ 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 $W - \text{elim}(W)$ takes $O(|W||\psi|)$ and computing $t\text{structand}$ takes $O(|W||TC|)$. Therefore, the problem is solvable in polynomial time.

**Theorem III.21** $WSP_{UT}$ for workflows containing only $\otimes$ and $\mid$, RBAC containing only binding and cardinality constraints where the number of binding constraints is less than a constant $k$, and task constraints containing both $\land$ and $\lor$ 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 $T$ 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 $O(|W||\psi| + k^2\log|W| + |U|^2(k|\psi| + |U|^\log|W|^2))$, the worst-case complexity of the algorithm is $O(|TC| + |W||\psi| + k^2\log|W| + |U|^2(k|\psi| + |U|^\log|W|^2))$.

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 $\land$ and $\lor$: (1) workflows containing only $\otimes$ and $\mid$, RBAC containing no constraints; (2) workflows containing only $\otimes$ and $\mid$, RBAC containing only binding constraints where the number of binding constraints is less than a constant $k$; and (3) workflows containing only $\otimes$ and $\mid$, 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 each (all) of the paths of a workflow $W$ under the restriction placed by an RBAC policy $\psi$. 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 $\psi$ 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 checks if $\psi$ is an instance. A set $E \subset R^*$ of roles is a solution to $\psi$ if

1. The Universal Minimum Role Satisfiability Problem
2. RBAC containing no constraints; (2) workflows containing only binding and cardinality constraints; (3) workflows containing only cardinality constraints.

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 $\text{minimum}$ (Line 6). If so, 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 $\text{compRoles}$ is $O(|W||\psi| + |W|^2|\log|W|)$. Since the worst-case complexity of $\text{compTask}$ is $O(|\psi||W|)$, the worst-case complexity of the algorithm is $O(|W||\psi| + |W|^2|\log|W| + |\psi|^2|W|)$.

**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 $p \in \text{paths}(W)$ such that for every task $T$ in $p$, there exists a role $r \in S_e$ such that $\psi$ 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 $\text{compRoles}$ (defined in Algorithm 2) to compute a set $Rset$ of roles that have permission to execute all paths of the workflow and the corresponding set of user-task assignments. Next, the algorithm computes a solution
to the problem from $\mathit{Rset}$. Function $\mathit{satRoles}(W, \mathit{sub}_1, \psi)$ checks if there exists $p \in \mathit{path}(W)$ such that roles in $\mathit{sub}_1$ 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 $\mathit{compRoles}$ and $\mathit{satRoles}$ 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 $\mathit{MRS}_E$, we can apply the algorithm for computing all solutions to $\mathit{MRS}_U$ 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 R²BAC, 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 $|\|^k$, multiple split with lower bound $|\|^k$, 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...
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


Ping Yang is an Assistant Professor in Department of Computer Science at 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, database, e-commerce and formal methods.

Shiyoung Lu is an Associate Professor in the Department of Computer Science at Wayne State University and the Director of the Scientific 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.