CERIAS Tech Report 2007-28

SATISFIABILITY AND RESILIENCY IN WORKFLOW SYSTEMS

by Qihua Wang and Ninghui Li

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

Satisfiability and Resiliency in Workflow Systems

Qihua Wang Ninghui Li
{wangq, ninghui}@cs.purdue.edu
Center for Education and Research in Information Assurance and Security
and Department of Computer Science
Purdue University

Abstract

We propose the role-and-relation-based access control ($\mathsf{R^2BAC}$*) model for workflow systems. In* $\mathsf{R^2BAC}$*, in addition to a user's role memberships, the user's relationships with other users help determine whether the user is allowed to perform a certain step in a workflow. For example, a constraint may require that two steps must not be performed by users who have a conflict of interest. We also study the workflow satisfiability problem, which asks whether a set of users can complete a workflow. We show that the problem is* $\mathbf{NP}$*-complete for* $\mathsf{R^2BAC}$*, and is* $\mathbf{NP}$*-complete for any workflow model that supports certain simple types of constraints (e.g., constraints that state certain two steps must be performed by two different users). After that, we apply tools from parameterized complexity theory to better understand the complexities of this problem. We show that the problem is fixed-parameter tractable when the only relations used are* $=$ *and* $\neq$*, and is fixed-parameter intractable when user-defined binary relations can be used. Finally, we study the resiliency problem in workflow systems, which asks whether a workflow can be completed even if a number of users may be absent. We formally define three levels of resiliency in workflow systems, namely, static resiliency, decremental resiliency and dynamic resiliency, and study computational problems related to these notions of resiliency.*

1 Introduction

Workflow systems are used in numerous domains, including production, purchase order processing, and various management tasks. Workflow authorization systems have gained popularity in the research community [1, 3, 5, 10, 11]. A workflow divides a task into a set of well-defined sub-tasks (called *steps* in the paper). Security policies in workflow systems are usually specified using authorization constraints. One may specify, for each step, which users are authorized to perform it. In addition, one may specify the constraints between users who perform different steps in the workflow. For example, one may require that two steps must be performed by different users for the purpose of separation of duty [4]. Oftentimes, constraints in workflow authorization systems need to refer to relationships among users. For example, the rationale under a separation of duty policy that requires 2 users to perform the task is that this deters and controls fraud, as the collusion of 2 users are required for a fraud to occur. However, when two users are close relatives, then collusion is much more likely. To achieve the objective of deterring and controlling fraud, the policy should require that two different steps in a workflow must be performed by users who are not in conflict of interest with each other. In different environments, the conflict-of-interest relation need to be defined differently. For instance, inside an organization's system, relationships such as close relatives (e.g., spouses and parent-child) can be maintained and users who are close relatives may be considered to be in conflict of interest. In a peer-review setting, conflict of interest may be based on past collaborations, common institutions, etc. For another example, one university may have a policy that a graduate student's study plan must be first approved by the student's advisor and then by the graduate officer in the student's department. To specify such a constraint, one needs to define and refer to the advisor-student binary relation.

In this paper, we introduce the role-and-relation-based access control ($\mathsf{R^2BAC}$) model for workflow systems. The model is role-based in the sense that individual steps of a workflow are authorized for roles. The model is relation-based in the sense that user-defined binary relations can be used to specify constraints and an authorized user is prevented from performing a step unless the user satisfies these constraints. $\mathsf{R^2BAC}$ is a natural step beyond Role-Based Access Control (RBAC) [9], especially in the setting of workflows. As a role defines a set of users, which can be viewed as a unary relation among the set of all users, a binary relation is the natural next step.

One fundamental problem in any workflow authorization systems is the *workflow satisfiability problem* (WSP), which asks whether a workflow can be completed in a certain system configuration. We show that WSP is $\mathbf{NP}$-complete in R^{2}BAC. Furthermore, we show that the intractability is inherent in any workflow authorization systems that support some simple kinds of constraints. In particular, we show that WSP is $\mathbf{NP}$-hard in any workflow system that supports *either* constraints that require two steps must be performed by different users *or* constraints that require one step must be performed by a user who also performs at least one of several other steps. Such intractability results are somewhat surprising and discouraging, because the constraints involved are simple and natural. It is also unsatisfying as such results do not shed light on the computation cost one has to pay by introducing additional expressive features such as user-defined binary relations, since the complexity of WSP is $\mathbf{NP}$-complete with or without them. Finally, the practical significance of such intractability results is unclear, as in real-world workflow systems, the number of steps should be small.

To address these issues, we apply tools from *parameterized complexity* [6] to WSP. Parameterized complexity is a measure of computational complexity of problems with multiple input parameters. Parameterized complexity enables us to perform finer-grained study on the computational complexity of WSP. We show that if only equality and inequality relations are used and the number of steps in the workflow is treated as a parameter, WSP is fixed-parameter tractable. More specifically, the problem can be solved in $O(f(k)n)$, where f is a function, k is the number of steps in the workflow, and n is the size of the problem. As the number of steps is relatively small in practice, this result shows that it is possible to solve WSP efficiently, when only equality and inequality relations are used. Also, we show that if user-defined relations are allowed, WSP is fixed-parameter intractable. More specifically, WSP is $W[1]$-hard and is in the complexity class $W[2]$; both of $W[1]$ and $W[2]$ are parameterized complexity classes within $\mathbf{NP}$. This illustrates that while supporting user-defined binary relations increases the expressive power, it also introduces a computational cost. We note that a naive algorithm solving WSP in R^{2}BAC takes time $O(kn^{k+1})$, which may be acceptable when k is small. The complexity $O(kn^{k+1})$ is not considered fixed-parameter tractable because one cannot separate n and k in the complexity to the form of $f(k)n^{\alpha}$, where $f(k)$ is independent of n and α is a constant independent of k. We also note that it is also possible to develop algorithms with heuristic optimizations that can solve WSP efficiently for practical instances; the study of such algorithms is beyond the scope of this paper.

In many situations, it is not enough to ensure that a workflow can be completed in the current system configuration. In particular, when the workflow is designed to complete a critical task, it is necessary to make sure that the workflow can be completed even if certain users become absent in emergency situations. In other words, *resiliency* is important in workflow systems. The notion of resiliency policies in access control has been recently introduced [8]. Unlike traditional security policies about access control, which focus on ensuring that access is properly *restricted* so that users who should not have access do not get access, resiliency policies aim at ensuring that access is properly *enabled* so that the system is resilient to the absence of users. The goal of resiliency policies is to guarantee that even if a number of users become absent in certain emergent situation, the remaining users can still finish the crucial tasks. An example resiliency policy is as follows: Upon the absence of up to four users, there must still exist three mutually disjoint sets of users such that the users in each set together have all permissions to carry out a critical task. Such a policy would be needed when one needs to be able to send up to three teams of users to different sites to perform a certain task, perhaps in response to some emergent events.

A challenging problem with both theoretical and practical interest is resiliency in workflow systems. Resiliency in workflow systems differs from the resiliency policies proposed in [8] in two aspects. First, due to the existence of authorization constraints, even if a set of users together are authorized to perform all steps in a workflow, it is still possible that they cannot complete the task. Second, as a workflow consists of a sequence of steps and finishing all these steps may take a relatively long time, it is possible that certain users become absent at some point and come back later. In other words, the set of available users may change during the execution of a workflow. Therefore, more refined notions of resiliency for workflow systems are needed. In this paper, we introduce three levels of resiliency in workflow systems and study the complexity of checking resiliency.

The contributions of this paper are as follows:

- We propose the role-and-relation-based access control (R^{2}BAC) model for workflow systems. R^{2}BAC naturally extends RBAC to use binary relations to specify authorization constraints and capture many security requirements commonly encountered in workflows.
- We show that WSP in R^{2}BAC is $\mathbf{NP}$-complete in general. We also show that WSP remains $\mathbf{NP}$-hard for any workflow model that supports one of two simple kinds of constraints. Such results are inherent to features of workflow authorization systems and are independent from specific modeling approaches.

- We apply tools from the parameterized complexity theory to WSP and show that it is fixed-parameter tractable when only equality and inequality relations are allowed. However, when user-defined binary relations can be used, WSP becomes fixed-parameter intractable. This clearly illustrates the computational cost incurred by having user-defined binary relations and gives algorithmic insights and ideas about solving WSP in the fixed-parameter tractable (but **NP**-complete) case.

  To the best of our knowledge, this paper is the first to use parameterized complexity in access control policy analysis. As a number of policy analysis problems in access control have been shown to be **NP**-complete, we believe that parameterized complexity theory can be fruitfully applied to these problems to shed insight on the causes of hardness in these problems as well as to give new algorithmic insights.

- We formally define three levels of resiliency in workflow systems. In *static resiliency*, up to t users are absent before the execution of an instance of a workflow. We show that checking whether a set of users is statically resilient for a workflow is **NP**-hard and is in $\mathbf{coNP^{NP}}$, a complexity class in the Polynomial Hierarchy. In *decremental resiliency*, users may become absent during the execution of an instance of a workflow, absent users will never come back for the same workflow instance, and at most t users may be absent in the end. *Dynamic resiliency* differs from decremental resiliency in that absent users may come back later and work on the same workflow instance, and at most t users may be absent at any given point of time. We show that checking whether a set of users is decremental resilient or dynamic resilient for a workflow is **PSPACE**-complete.

The remainder of the paper is organized as follows. We introduce the R^{2}BAC model in Section 2. After that, we study the workflow satisfiability problem in Section 3 and study parameterized complexity of the problem in Section 4. We then define and study resiliency problems in workflow systems in Section 5. We discuss related work in Section 6. Finally, we conclude and discuss open problems in Section 7.

2 The Role-and-Relation-Based Access Control Model for Workflow Systems

In this section, we introduce the Role-and-Relation-Based Access Control (R^{2}BAC) model for workflow systems. We start with a motivating example.

Example 1. In an academic institution, submitting a grant proposal to an outside sponsor via the sponsor program services (SPS) is modeled as a workflow with five steps[1] (see Figure 1).

1. `Preparation`: A faculty member prepares a proposal and sends it to the business office of his or her department.
2. `Budget`: An account clerk prepares the budget, checks the proposal, and submits it to the SPS office.
3. `Expert Review`: A regulation expert in the SPS office reviews the proposal to check whether the proposal satisfies various regulations, e.g., those governing export control and human subject research.
4. `Account Review`: An account manager reviews the proposal and the budget.
5. `Submission`: An account manager submits the proposal to the outside sponsor.

In the workflow, steps `expert review` and `account review` may be performed concurrently while all other steps must be carried out sequentially. The step `preparation` can be performed by any personnel who can serve as a primary investigator, while the step `budget` must be carried out by an account clerk. A regulation expert is authorized to review the proposal in the step `expert review`. The privilege to perform steps `account review` and `submission` is granted to account managers.

The workflow has the following constraints.

1. Steps `preparation`, `budget`, `expert review` and `account review` must be performed by four different users.
2. The account clerk who signs the proposal must be in the same department as the faculty member who prepares the proposal.
3. The persons who review the proposal must not have a conflict of interest with the one submitting the proposal.
4. The account manager who reviews the proposal is responsible to submit it to the outside sponsor.

[1]This is a simplified version of the process in the authors' institution, which also requires signatures of the department head and the dean's office.

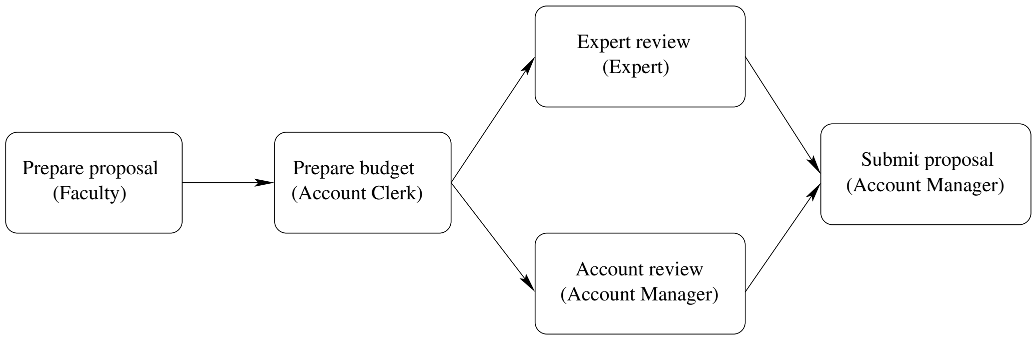

Figure 1. A workflow for grant proposal submission to outside sponsor via the sponsor program services (SPS).

In the above, Constraint 2 reflects certain procedural and duty requirements, while Constraint 1 enforces the principle of separation of duty. Constraint 3 follows the spirit of separation of duty and goes beyond that. Rather than simply requiring that the two steps must be performed by different people, the constraint requires that the people who perform the two steps must not have a conflict of interest. Constraint 4 enforces a binding-of-duty policy [5] by requiring two tasks be performed by the same user.

As security and practical requirements vary from tasks to tasks, the specification of constraints plays a crucial role in the expression of workflow. As demonstrated in Example 1, binary relations play an important role in expressing authorization constraints. Most existing workflow authorization models support only a few pre-defined binary relations, which limits the expressive power of these models. For example, the model proposed in [10] supports only six pre-defined binary relations $\{=, \neq, <, \leq, >, \geq\}$ between users and roles. Hence, there is no way to express relations like "in the same department" or "is a family member". The model in [5] supports user-defined relations. Our role-and-relation-based access control (R^{2}BAC) model for workflow systems extends the model in [5] by explicitly combining roles and relations and by supporting more sophisticated forms of constraints using these relations.

We now introduce formal definitions for R^{2}BAC. Note that $\mathcal{U}$, $\mathcal{R}$ and $\mathcal{B}$ are names of all possible users, roles and binary relations in the system, respectively.

Definition 1 (Configuration)**.** A *configuration* is given by a tuple $\langle U, UR, B\rangle$, where $U \subseteq \mathcal{U}$ is a set of users, $UR \subseteq U \times \mathcal{R}$ is the user-role membership relation and $B = \{\rho_1, \cdots, \rho_m\} \subseteq \mathcal{B}$ is a set of binary relations such that $\rho_i \subseteq U \times U$ $(i \in [1, m])$. For convenience, we assume that when ρ is in B, $\overline{\rho}$ is also in B, and $(u_1, u_2) \in \overline{\rho}$ if and only if $(u_1, u_2) \notin \rho$. Also, $\overline{\overline{\rho}}$ is the same as ρ. Furthermore, we assume that B contains two predefined binary relations "$=$" and "$\neq$", which denote equality and inequality, respectively.

A configuration $\langle U, UR, B\rangle$ defines the environment in which a workflow is to be run. In particular, B should define all the binary relations that appear in any constraint in workflows to be run in the environment.

Definition 2 (Workflow and Constraints)**.** A *workflow* is represented as a tuple $\langle S, \preceq, SA, C\rangle$, where S is a set of steps, $\preceq\, \subseteq S \times S$ defines a partial order among steps in S, $SA \subseteq \mathcal{R} \times S$, and C is a set of constraints, each of which takes one of the following forms:

1. $\langle\rho(s_1, s_2)\rangle$: the user who performs s_1 and the user who perform s_2 must satisfy the binary relation ρ.
2. $\langle\rho(\exists X, s)\rangle$: there exists a step $s' \in X$ such that $\langle\rho(s', s)\rangle$ holds, i.e., the user who performs s' and the user who performs s satisfy ρ.
3. $\langle\rho(s, \exists X)\rangle$: there exists a step $s' \in X$ such that $\langle\rho(s, s')\rangle$ holds.
4. $\langle\rho(\forall X, s)\rangle$: for each step $s' \in X$, $\langle\rho(s', s)\rangle$ must hold.
5. $\langle\rho(s, \forall X)\rangle$: for each step $s' \in X$, $\langle\rho(s, s')\rangle$ must hold.

Intuitively, in a workflow $\langle S, \preceq, SA, C\rangle$, that $s_i \preceq s_j$ $(i \neq j)$ indicates that step s_i must be performed before step s_j. Steps s_i and s_j may be performed *concurrently*, if neither $s_i \preceq s_j$ nor $s_j \preceq s_i$. SA is called *role-step authorization* and $(r, s) \in SA$ indicates that members of role r is authorized to perform step s.

Example 2. Consider the workflow for submitting a grant proposal in Example 1. Let $s_{prepare}$, s_{budget}, s_{xp_review}, s_{ac_review} and s_{submit} denote the five steps in the workflow. The constraints of the workflow can be represented in tuple-based specification as follows.

1. $\langle \neq (s_{budget}, s_{prepare})\rangle$,
   $\langle \neq (s_{xp\_review}, \forall\{s_{prepare}, s_{budget}\})\rangle$,
   $\langle \neq (s_{ac\_review}, \forall\{s_{prepare}, s_{budget}, s_{xp\_review}\})\rangle$

   These require that the first four steps in the workflow must be performed by four different users.

2. $\langle \rho_{same\_dept}(s_{budget}, s_{prepare})\rangle$

   $(u_x, u_y) \in \rho_{same\_dept}$ when u_x and u_y are in the same department. The constraint requires that the person who signs the proposal must be in the same department as the person who prepares it.

3. $\langle \overline{\rho}_{conflict\_interest}(\forall\{s_{xp\_review}, s_{ac\_review}\}, s_{prepare})\rangle$

   $(u_x, u_y) \in \rho_{conflict\_interest}$ when u_x and u_y have a conflict of interest. The constraint requires that the person who reviews the proposal must not have a conflict of interest with the person who prepares it.

4. $\langle = (s_{submit}, s_{ac\_review})\rangle$

   The constraint requires that `account review` and `submission` must be performed by the same person.

Definition 3 (Plans and Partial Plans)**.** A *plan* P for workflow $W = \langle S, \preceq, SA, C\rangle$ is a subset of $\mathcal{U} \times S$ such that, for every step $s_i \in S$, there is exactly one tuple (u_a, s_i) in P, where $u_a \in \mathcal{U}$.

A *partial plan* PP for W is a subset of $\mathcal{U} \times S$ such that, for every step $s_i \in S$, there is at most one tuple (u_a, s_i) in PP, where $u_a \in \mathcal{U}$. And $(u_a, s_i) \in PP$ implies that, for every $s_j \preceq s_i$, there exists $u_b \in \mathcal{U}$ such that $(u_b, s_j) \in PP$.

Intuitively, a plan assigns exactly one user to every step in a workflow, while a partial plan does this for only a portion of the steps in the workflow. Furthermore, if a step is in a partial plan, then its prerequisite steps must also be in the partial plan.

Definition 4 (Valid Plan)**.** Given a workflow $W = \langle S, \preceq, SA, C\rangle$, and a configuration $\Gamma = \langle U, UR, B\rangle$, we say that a user u is an *authorized user* of a step $s \in S$ under Γ if and only if there exists a role r such that $(u, r) \in UR$ and $(r, s) \in SA$.

We say that a plan P *is valid for* W under Γ if and only if for every $(u, s) \in P$, u is an authorized user of s, and no constraint in C is violated. We say that W is *satisfiable* under Γ if and only if there exists a plan P that is valid for W under Γ.

Note that there can be multiple valid plans for a workflow W under a configuration. In fact, it is the existence of multiple valid plans that makes it possible for W to be completed even if a number of users are absent. In situations where the configuration changes during the execution of a workflow instance (e.g. users become absent), we will have to change our plan at runtime and thus constraints need to be checked at runtime as well. If a constraint c contains $\forall$, then it is checked whenever a step restricted by c is to be executed. Other kinds of constraints are checked before the last step restricted by the constraint is to be executed.

Definition 5 (Valid Partial Plan)**.** Given a workflow $\langle S, \preceq, SA, C\rangle$ and a configuration $\langle U, UR, B\rangle$, let $s_1, \cdots, s_m$ be a sequence of steps such that $s_i \not\preceq s_j$ when $i > j$. A partial plan PP is *valid* with respect to the sequence $s_1, \cdots, s_i$ if it assigns one user to each step in $s_1, \cdots, s_i$ and no constraint that is checked before the execution of s_i is violated by PP.

3 The Workflow Satisfiability Problem

One fundamental problem in workflow authorization systems is the Workflow Satisfiability Problem (WSP), which checks whether a workflow W is satisfiable under a configuration Γ. Note that, given configuration $\langle U, UR, B\rangle$, checking whether W is satisfiable under Γ is equivalent to checking whether there is a valid plan for W under Γ. In this section, we study the computational complexity of WSP.

3.1 Computational Complexity of WSP for R^{2}BAC

Theorem 1. WSP is $\mathbf{NP}$-complete in R^{2}BAC.

The proof of Theorem 1 consists of two parts. The first part is Lemma 2, which shows that WSP is in $\mathbf{NP}$ in R^{2}BAC. In the second part, Lemma 3 and Lemma 4 show that WSP is $\mathbf{NP}$-hard in two restricted cases. Proofs for these lemmas are in Appendix A.

Lemma 2. WSP is in $\mathbf{NP}$ in R^{2}BAC.

Intuitively, a nondeterministic Turing can guess a plan and check whether the plan is valid in polynomial time.

Lemma 3. WSP is $\mathbf{NP}$-hard in R^{2}BAC, if the workflow uses constraints of the form $\langle \neq (s_1, s_2) \rangle$.

In the proof of the above lemma (in Appendix A), we use a reduction from the $\mathbf{NP}$-complete GRAPH K-COLORABILITY problem. In the reduction, vertices in a graph are mapped to steps in the workflow, while colors are mapped to users. In the GRAPH K-COLORABILITY problem, the number of vertices is normally much larger than the number of colors. Hence, the number of steps in the constructed workflow is much larger than the number of users, which is rarely the case in practice. Such a phenomenon indicates that classical complexity framework is inadequate to study the complexity of WSP in a real-word setting. This motivates us to apply the tool of parameterized complexity to perform finer-grained study of the complexity of WSP, which will be discussed in Section 4.

Lemma 4. WSP is $\mathbf{NP}$-hard in R^{2}BAC, if the workflow uses constraints of the form $\langle = (s, \exists X) \rangle$.

The proof of this lemma uses a reduction from the $\mathbf{NP}$-complete HITTING SET problem to WSP.

Although WSP is intractable in general in R^{2}BAC, the problem is in $\mathbf{P}$ for certain special cases. Lemma 5 states a tractable case for WSP.

Lemma 5. WSP is in $\mathbf{P}$ in R^{2}BAC, if the workflow only has constraints in the forms of $\langle = (s_1, s_2) \rangle$, $\langle = (s, \forall X) \rangle$ or $\langle = (\forall X, s) \rangle$.

3.2 The Inherent Complexity of Workflow Systems

In Section 3.1, we show that WSP is $\mathbf{NP}$-hard in R^{2}BAC in general. In this section, we stress that the intractability of WSP is inherent to certain fundamental features of workflow authorization systems and independent from modeling approaches. We say that a workflow system supports the feature of *user-step authorization* if it allows one to specify (either directly or indirectly) which users are allowed to perform which steps in the workflow. User-step authorization is probably the most fundamental feature and almost all workflow systems found in existing literatures support such feature. A *user-inequality constraint* states that certain two steps cannot be performed by the same user, i.e., $\langle \neq (s_1, s_2) \rangle$ in R^{2}BAC. An *existence-equality constraint* states that a certain step must be performed by a user who performs at least one step in a given set of steps, i.e., $\langle = (s, \exists X) \rangle$ in R^{2}BAC.

Theorem 6. Checking whether a set of users can complete a workflow is $\mathbf{NP}$-hard for any workflow system that supports user-step authorization and user-inequality constraints.

Proof. The reduction from GRAPH K-COLORABILITY in the proof of Lemma 3 only makes use of user-step authorization and user-inequality constraints offered by R^{2}BAC. Therefore, the reduction also applies to the satisfiability problem for any workflow system that supports these two features. □

Note that user-inequality constraints are widely used in existing literatures to enforce separation of duty in workflow systems. Many workflow models [3, 10, 5] support such type of constraints.

Theorem 7. Checking whether a set of users can complete a workflow is $\mathbf{NP}$-hard for any workflow system that supports user-step authorization and existence-equality constraints.

Proof. The reduction from HITTING SET in the proof of Lemma 4 only makes use of user-step authorization and existence-equality constraints offered by R^{2}BAC. Therefore, the reduction also applies to the satisfiability problem for any workflow system that supports these two features. □

Note that existence-equality constraints are a natural way to enforce the general form of binding of duty policies, which require a step be performed by one of those users who have performed some prerequisite steps.

4 Beyond Intractability of WSP

In Section 3, we have shown that WSP is $\mathbf{NP}$-complete in R^{2}BAC for the general case as well as the two special cases where only a simple form of constraints are used. Such results are, however, unsatisfying, as they do not shed light on the computation cost associated with introducing additional expressive features such as user-defined binary relations, since the complexity of WSP is $\mathbf{NP}$-complete in all the three cases. Such a phenomenon indicates that classical computational complexity does not precisely capture the computational difficulty of different cases of WSP. Furthermore, the practical significance of such intractability results is unclear. The input to WSP consists of many aspects, such as the number of steps in the workflow, the number of constraints and the number of users in the configuration etc. In practice, some aspects of the input will not take a large value. For instance, even though the number of users may be large, the number of steps in the workflow is expected to be small. An interesting question arises is whether WSP can be solved efficiently given the restriction that the number of steps is small.

To address these issues, we apply tools from the theory of parameterized complexity [6] to WSP.

4.1 Why Parameterized Complexity?

Parameterized complexity is a measure of complexity of problems with multiple input parameters. The theory of parameterized complexity was developed in the 1990s by Rod Downey and Michael Fellows. It is motivated, among other things, by the observation that there exist hard problems that (most likely) require exponential runtime when complexity is measured in terms of the input size only, but that are computable in a time that is polynomial in the input size and exponential in a (small) parameter k. Hence, if k is fixed at a small value, such problems can still be considered 'tractable' despite their traditional classification as 'intractable'.

In classical complexity, a decision problem is specified by two items of information: (1) the input to the problem, and (2) the question to be answered. In parameterized complexity, there are three parts of a problem specification: (1) the input to the problem, (2) the aspects of the input that constitute the parameter, and (3) the question to be answered. Normally, the parameter is selected because it is likely to be confined to a small range in practice. The parameter provides a systematic way of specifying restrictions of the input instances. Some $\mathbf{NP}$-hard problems can be solved by algorithms that are exponential only in a fixed parameter while polynomial in the size of the input. Such an algorithm is called a *fixed-parameter tractable* algorithm. More specifically, an algorithm for solving a problem is a fixed-parameter tractable algorithm, if when given any input instance of the problem with parameter k, the algorithm takes time $O(f(k)n^{\alpha})$, where n is the size of the input, k is the parameter, α is a constant (independent of k), and f is an arbitrary function.

If a problem has a fixed-parameter tractable algorithm, then we say that it is a fixed-parameter tractable problem and belongs to the class $\mathbf{FPT}$. For example, the $\mathbf{NP}$-complete VERTEX COVER asks, given a graph G and an integer k, whether there is a size-k set V' of vertices, such that every edge in G is adjacent to at least one vertex in V'. This problem is in $\mathbf{FPT}$ when taking k as the parameter, as there exists a simple algorithm with running time of $O(2^k n)$, where n is the size of G. Note that not all intractable problems are in $\mathbf{FPT}$. For instance, the $\mathbf{NP}$-complete DOMINATING SET problem is fixed-parameter intractable. Given a graph G and an integer k, DOMINATING SET asks whether there is a size-k set V' of vertices such that every vertex in G is either in V' or is connected to a vertex in V' by an edge. For DOMINATING SET, there is no significant alternative to trying all size-k subsets of vertices in G and there are $O(n^k)$ such subsets, where n is the number of vertices.

Finally, we would like to point out that a problem in $\mathbf{FPT}$ does not necessarily mean that it can be efficiently solved as long as the parameter is small. Note that $f(k)$ may be a function that grows very fast over k. For instance, an $O(k^{k^k} n)$ algorithm is not practical even if k is as small as 5, just as we cannot claim that a problem in $\mathbf{P}$ can be solved efficiently when the best algorithm takes time $O(n^{100})$. However, showing that a problem is in $\mathbf{FPT}$ has significant impact as experiences have shown that improvement on fixed-parameter tractable algorithms are oftentimes possible. For instance, when VERTEX COVER was first observed to be solvable in $O(f(k)n^3)$, $f(k)$ was such a function that the algorithm is utterly impractical even for $k = 1$. An $O(2^k n)$ algorithm was proposed later, and then an algorithm with running time $O(kn + (4/3)^k k^2)$ was revealed. Right now, VERTEX COVER is well-solved for input of any size, as long as the parameter value is $k \leq 60$. Parameterized complexity offers a fresh angle into designing algorithms for such problems.

In this paper, we only study which subcases of WSP are in $\mathbf{FPT}$ and which are not. Improvement on the fixed-parameter tractable algorithms for the $\mathbf{FPT}$ cases is beyond the scope of this paper.

4.2 Fixed Parameter Tractable Cases of WSP

As the number of steps in a workflow is likely to be small in practice, we select the number of steps as the parameter for WSP. We first show that a special case of WSP in which only the $\neq$ relation is allowed is in $\mathbf{FPT}$. The proof gives a fixed-parameter tractable algorithm and illustrates the intuition why this problem is in $\mathbf{FPT}$.

Lemma 8. WSP in R^{2}BAC is in $\mathbf{FPT}$, if $\neq$ is the only binary relation used by constraints in the workflow. In particular, given a workflow W and a configuration Γ, WSP can be solved in time $O(k^{k+1}n)$, where k is the number of steps in W and n is the size of the entire input to the problem.

Proof. A constraint using binary relation $\neq$ requires a certain step to be performed by a user who does not perform certain other step(s). Since there are k steps in W, if step s is authorized to no less than k users in U, then we can always find an authorized user of s, who is not assigned to any other steps in W. In other words, we only need to consider those steps that are authorized to less than k users in U, and there are at most k such steps. We construct partial plans for these steps by trying all combinations of authorized users and there are no more than k^k such combinations. Verifying whether a plan is valid can be done in $O(kn)$, as there are $O(n)$ constraints and each constraints restricts at most k steps. Therefore, checking whether U can complete W can be done in time $O(k^{k+1}n)$. $\square$

Theorem 9. WSP is in $\mathbf{FPT}$ in R^{2}BAC, if = and $\neq$ are the only binary relations used by constraints in the workflow.

This Theorem subsumes Lemma 8. Proof of this theorem is given in Appendix B. The proof uses Lemma 8.

4.3 WSP is Fixed Parameterized Intractable in General

A natural question to ask is whether WSP is still in $\mathbf{FPT}$ when user-defined binary relations are allowed in the workflow. We show that the answer is "no". Similar to proving a problem is intractable in classical complexity framework, we prove that a problem is fixed-parameter intractable by reducing another fixed-parameter intractable problem to the target problem. To preserve fixed-parameter tractability, we need to use a kind of reduction different from the classical ones used in $\mathbf{NP}$-completeness proofs. We say that L reduces to L' by a *fixed-parameter reduction* if given an instance $\langle x, k\rangle$ for L, one can compute an instance $\langle x' = g_1(\langle x, k\rangle), k' = g_2(k)\rangle$ in time $O(f(k)|x|^\alpha)$ such that $\langle x, k\rangle \in L$ if and only if $\langle x', k'\rangle \in L'$, where g_1 and g_2 are two functions and α is a constant. Note that many classical reductions are not fixed-parameter reduction as they do not carry enough structure, and lead to lose of control for the parameter.

Under parameterized complexity, each problem falls somewhere in the hierarchy: $\mathbf{P} \subseteq \mathbf{FPT} \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq \mathbf{NP}$. If a problem is $W[1]$-hard, then it is believed to be fixed-parameter intractable. To understand the classes $W[t]$, we can start by viewing a 3CNF formula as a (boolean) decision circuit, consisting of one input for each variable and structurally a large *and* gate taking inputs from a number of small *or* gates. (Some wires in the circuit may include a negation.) The *or* gates are small in that each of them takes 3 inputs, and the *and* gate is large in that it takes an unbounded number of inputs. The *weft* of a decision circuit is the maximum number of large gates on any path from the input variable to the output line. The *weighted satisfiability* problem for decision circuits asks whether a decision circuit has a weight k satisfying assignment (i.e., a satisfying assignment in which at most k variables are assigned true). The class $W[t]$ includes all problems that are fixed parameter reducible to the weighted satisfiability problem for decision circuits of weft t.

The following theorem states that WSP is fixed-parameter intractable in R^{2}BAC when user-defined binary relations are allowed in the workflow.

Theorem 10. WSP is $W[1]$-hard in R^{2}BAC if user-defined binary relations are used in constraints.

The proof to the above theorem is given in Appendix B. In the proof, we reduce the $W[1]$-complete INDEPENDENT SET problem to WSP.

We conclude from Theorem 9 and Theorem 10 that supporting user-defined binary relations introduces additional complexity to WSP in R^{2}BAC. Parameterized complexity reveals such a fact that is hidden by classical complexity framework and allows us to better understand the source of complexity of WSP in R^{2}BAC. We point out that a naive algorithm solving WSP for R^{2}BAC, which enumerates all possible plans and verifies each of them, takes time $O(kn^{k+1})$, which may be acceptable when k is small. We also note that it is possible to develop algorithms with heuristic optimizations that can solve WSP efficiently for practical instances; the study of such algorithms is beyond the scope of this paper.

Finally, we provide an upperbound for WSP in R^{2}BAC in the parameterized complexity framework. The proof to Theorem 11 is given in Appendix B.

Theorem 11. WSP in $\mathsf{R}^2\mathsf{BAC}$ is in $W[2]$.

It remains open whether WSP is $W[1]$-complete or $W[2]$-complete.

5 Resiliency in Workflow Systems

We have studied the workflow satisfiability problem (WSP) in previous sections. In many situations, it is not enough to ensure that a workflow is satisfiable in the current system configuration. In particular, when the workflow is designed to complete a critical task, it is necessary to guarantee that even if certain users are absent unexpectedly, the workflow can still be completed. Resiliency is a property of those system configurations that can satisfy the workflow even with absence of some users.

In this section, we define and study resiliency in workflow systems. The workflow model we use is $\mathsf{R}^2\mathsf{BAC}$. Before giving formal definitions of resiliency in workflow systems, let us consider several possible scenarios.

1. The execution of instances of a workflow is done in a relatively short period of time, say within fifteen minutes. Although it is possible that certain users are absent before the execution of a workflow instance, it is unlikely that available users become absent during the execution of the workflow instance. In other words, the set of users who are available for a workflow instance is stable.
2. The execution of instances of a workflow takes a relatively long period of time, say within one day. Some users may not come to work on the day when a workflow instance is executed. Furthermore, some users may have to leave at some point (e.g. between the execution of two steps) before the workflow instance is completed and will not come back to work until the next day. In such a situation, the set of users available to the workflow instance becomes smaller and smaller over time. Such a scenario would also be possible in potentially hazardous situations such as battlefield and fire-fighting.
3. The execution of instances of a workflow takes a long period of time. For example, only a single step of the workflow is performed each day. Since the set of users who come to work may differ from day to day, the set of available users may differ from step to step.

We capture the above three scenarios by proposing three levels of resiliency in workflow systems. They are *static (level-1) resiliency*, *decremental (level-2) resiliency* and *dynamic (level-3) resiliency*. In static resiliency, a number of users are absent before the execution of a workflow instance, while remaining users will not be absent during the execution; in decremental resiliency, users may be absent before or during the execution of a workflow instance, and absent users will not become available again; in dynamic resiliency, users may be absent before or during the execution of a workflow instance and absent users may become available again. In all cases, we assume that the number of absent users at any point is bounded by a parameter t. We now give formal definitions of the three levels of resiliency.

Definition 6 (Static Resiliency)**.** Given a workflow W and an integer $t \geq 0$, a configuration $\langle U, UR, B\rangle$ is *statically resilient* for W up to t absent users if and only if for every size-t subset U' of U, W is satisfiable under $\langle (U - U'), UR, B\rangle$.

Intuitively, a configuration is statically resilient for a workflow if the workflow is still satisfiable after removing t users from the configuration.

Definition 7 (Decremental Resiliency)**.** Given a workflow $W = \langle S, \preceq, SA, C\rangle$ and an integer t, a configuration $\langle U, UR, B\rangle$ is *decrementally resilient* for W up to t absent users, if and only if Player 1 can always win the following two-person game when playing optimally.

Initialization: $PP \leftarrow \emptyset$, $U_0 \leftarrow U$, $S_0 \leftarrow S$, $t_0 \leftarrow t$ and $i \leftarrow 1$.

Round i of the Game:

1. Player 2 selects a set U'_{i-1} such that $|U'_{i-1}| \leq t_{i-1}$.
   $U_i \leftarrow (U_{i-1} - U'_{i-1})$ and $t_i \leftarrow (t_{i-1} - |U'_{i-1}|)$.
2. Player 1 selects a step $s_{a_i} \in S_{i-1}$ such that $\forall s_b(s_b \prec s_{a_i} \Rightarrow s_b \notin S_{i-1})$.
   Player 1 selects a user $u \in U_i$.
   $PP \leftarrow PP \cup \{(u, s_{a_i})\}$ and
   $S_i \leftarrow (S_{i-1} - \{s_{a_i}\})$.
   If PP is not a valid partial plan with respect to the sequence $s_{a_1}, \cdots, s_{a_i}$, then Player 1 loses.

3. If $S_i = \emptyset$, then Player 1 wins; otherwise, let $i \leftarrow (i+1)$ and the game goes on to the next round.

In each round, Player 2 may remove a certain number of users and then Player 1 has to pick a remaining step that is ready to be performed and assign an available user to it. The total number of users Player 2 may remove throughout the game is bounded by t. A configuration is decrementally resilient for a workflow if there is always a way to complete the workflow no matter when and which users are removed, as long as the total number of absent users is bounded by t.

Also, in Definition 7, we assume that Player 1 plays optimally, which implies that in each round, Player 1 has to consider not only the next step but also all future steps.

Example 3. There is a workflow $W = \langle S, \preceq, SA, C\rangle$ and a configuration $\langle U, UR, B\rangle$, where $S = \{s_1, s_2\}$, $s_1 \preceq s_2$, $C = \{\langle \neq (s_1, s_2)\rangle\}$, $SA = \{(r_1, s_1), (r_2, s_2)\}$, and $UR = \{(Alice, r_1), (Alice, r_2), (Bob, r_1), (Carl, r_2)\}$. All users in $U = \{Alice, Bob, Carl\}$ are available before the execution of s_1. Consider the following two choices of user assignment for s_1.

1. $Alice$ is assigned to perform s_1: If $Carl$ becomes absent after the execution of s_1, then $Alice$ is the only user authorized to perform s_2. However, assigning $Alice$ to s_2 violates the constraint $\langle \neq (s_1, s_2)\rangle$. That is to say, the remaining users cannot complete the workflow.
2. Bob is assigned to perform s_1: In this case, no matter which single user becomes absent after the execution of s_1, we can always find an authorized user (either $Alice$ or $Carl$) to perform s_2 without violating the constraint $\langle \neq (s_1, s_2)\rangle$.

Thus it is clear that having Bob perform s_1 is a better choice than having $Alice$ with respect to resiliency. Actually, it can be proved that this configuration is decrementally resilient for W up to one absent user.

Definition 8 (Dynamic Resiliency)**.** Given a workflow $W = \langle S, \preceq, SA, C\rangle$ and an integer t, a configuration $\langle U, UR, B\rangle$ is *dynamically resilient* for W up to t absent users, if and only if Player 1 can always win the following two-person game when playing optimally.

Initialization: $PP \leftarrow \emptyset$, $S_0 \leftarrow S$ and $i \leftarrow 1$.

Round i of the Game:

1. Player 2 selects a set U'_{i-1} of up to t users.
   $U_i \leftarrow (U - U'_{i-1})$.
2. Player 1 selects a step $s_{a_i} \in S_{i-1}$ such that $\forall s_b (s_b \prec s_{a_i} \Rightarrow s_b \notin S_{i-1})$.
   Player 1 selects a user $u \in U_i$.
   $PP \leftarrow PP \cup \{(u, s_{a_i})\}$ and
   $S_i \leftarrow (S_{i-1} - \{s_{a_i}\})$.
   If PP is not a valid partial plan with respect to the sequence $s_{a_1}, \cdots, s_{a_i}$, then Player 1 loses.
3. If $S_i = \emptyset$, then Player 1 wins; otherwise, let $i \leftarrow (i+1)$ and the game goes on to the next round.

Intuitively, Player 2 may temporarily remove up to t users from the configuration at the beginning of each round. Then, Player 1 has to select a remaining step that is ready to be performed and assign an available user to it. After that, the configuration is restored and the next round of the game starts.

The following theorem states a relationship among the three levels of resiliency in workflow systems: dynamic (level-3) resiliency is stronger than decremental (level-2) resiliency, which is in turn stronger than static (level-1) resiliency.

Theorem 12. Given a workflow W, a configuration Γ and an integer t, the following are true.

- If Γ is dynamically resilient for W up to t absent users, then it is also decrementally resilient for W up to t absent users.
- If Γ is decrementally resilient for W up to t absent users, then it is also statically resilient for W up to t absent users.

But the reverse of either of the above statements is not true.

Proof. The game defining dynamic resiliency allows Player 2 to play any strategy he/she can in the game defining decremental resiliency. The same relation holds between the game defining decremental resiliency and the game defining static resiliency. Therefore, the theorem holds. □

5.1 Computational Complexities of Checking Resiliency

Theorem 13. Checking whether a configuration Γ is statically resilient for a workflow W up to t users, which is called the *Static Resiliency Checking Problem* (SRCP), is $\mathbf{NP}$-hard and is in $\mathbf{coNP}^{\mathbf{NP}}$.

Proof. When $t = 0$, SRCP degenerates to WSP. Since WSP is $\mathbf{NP}$-complete, SRCP is $\mathbf{NP}$-hard.

Next, we prove that the problem is in $\mathbf{coNP}^{\mathbf{NP}}$. From Lemma 2, checking whether a workflow is satisfiable under a configuration $\langle U, UR, B\rangle$ is in $\mathbf{NP}$. We now construct a nondeterministic oracle Turing machine M that decides the complement of the problem. Assume that M has access to an $\mathbf{NP}$ oracle N which checks whether a workflow is satisfiable under a configuration. M nondeterministically selects a set U' of t users and asks N whether the workflow is satisfiable under $\langle (U - U'), UR, B\rangle$. If the answer is "yes", M returns "no"; otherwise, M returns "yes". In this case, M returns "yes" if and only if the answer to the SRCP instance is "no". In general, SRCP is in $\mathbf{coNP}^{\mathbf{NP}}$. $\square$

It remains open whether SRCP is $\mathbf{coNP}^{\mathbf{NP}}$-complete or not. Readers who are familiar with computational complexity theory will recognize that $\mathbf{coNP}^{\mathbf{NP}}$ is a complexity class in the Polynomial Hierarchy. (See Appendix C for a brief introduction to the Polynomial Hierarchy.) Because the Polynomial Hierarchy collapses when $\mathbf{P} = \mathbf{NP}$, showing that an $\mathbf{NP}$-hard decision problem is in the Polynomial Hierarchy, although is not equivalent to showing that the problem is $\mathbf{NP}$-complete, has the same consequence: the problem can be solved in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

Theorem 14. Checking whether a configuration Γ is decremental resilient for a workflow W up to t users, which is called the *Decremental Resiliency Checking Problem* (CRCP), is $\mathbf{PSPACE}$-complete.

Theorem 15. Checking whether a configuration Γ is dynamically resilient for a workflow W up to t users, which is called the *Dynamic Resiliency Checking Problem* (DRCP), is $\mathbf{PSPACE}$-complete.

Please refer to Appendix D for proofs of Theorem 14 and 15. In the proofs, we reduce the $\mathbf{PSPACE}$-complete QUANTIFIED SATISFIABILITY problem to CRCP or DRCP. Intuitively, we use user-step assignments in workflow to simulate truth assignments for boolean variables.

Note that given a workflow $W = \langle S, \preceq, SA, C\rangle$, there may not exist a set of users that is decrementally or dynamically resilient for W even just up to one absent user, when the equality relation is used. For instance, assume that $S = \{s_1, s_2\}$, $s_1 \preceq s_2$ and $C = \{\langle = (s_1, s_2)\rangle\}$. Constraint $\langle = (s_1, s_2)\rangle$ requires s_1 and s_2 be performed by the same user. (Such constraints appear in [5] under the name binding-of-duty constraints.) If the user who executed s_1 becomes absent before the execution of s_2, then there is no way to finish the workflow without violating the constraint no matter which users remain available. This illustrate that bind-of-duty constraints can make it difficult to achieve decremental or dynamic resiliency. This problem can be addressed either by not using such constraints in settings where such resiliency is desirable, or by introducing concepts such as delegation, where one user can act on behalf of another user in the user's absence. The study of using delegation for resiliency is interesting future work.

6 Related Work

Bertino et al. [3] introduced a language to express workflow authorization constraints as clauses in a logic programming language. The language supports a number of predefined relations for constraint specification. Bertino et al. [3] also proposed searching algorithms to assign users to complete a workflow. This work does not support user-defined binary relations, nor does it formally study computational complexity of the workflow satisfiability problem. Tan et al. [10] studied the consistency of authorization constraints in workflow systems. The model in [10] supports six predefined binary relations: $\{=, \neq, <, \leq, >, \geq\}$, but not user-defined relations. Atluri and Huang [1] proposed a workflow authorization model that focuses on temporal authorization. This model does not support constraints about users performing different steps in a task. Atluri and Warner [2] proposed a model that supports conditional delegation in workflow systems. Delegation is a potential mechanism to achieve resiliency. In this paper, we consider resiliency without using delegation. We plan to extend our definitions on resiliency to take delegation into account and study how to use delegation to achieve resiliency in workflow systems. Furthermore, in [11], Warner and Atluri considered authorization constraints that span multiple instances of a workflow. Their model supports predefined relations with emphasis on inter-instance constraints. Inter-instance problems in workflow systems is an interesting research area. The models in [2, 11] do not support user-defined relations. Finally, Kang et al. [7] investigated access control mechanisms for inter-organizational workflow. Their workflow model authorizes steps to roles and supports dynamic constraints. However, they do not explicitly point out how constraints are specified and what kinds of constraints are supported besides separation of duty. Their paper mainly focuses on infrastructure design and implementation.

The workflow authorization model proposed by Crampton [5] is probably the one that is most closely related to R^{2}BAC. The model in [5] supports user-defined binary relations; however, it does not support quantifiers in constraints, so that constraints of the form $\langle \rho(\exists X, s) \rangle$ cannot be expressed in that model. Crampton [5] also studied the workflow satisfiability problem and presented a polynomial time algorithm for their model. However, the algorithm is incorrect.[2] Each constraint in [5] relates two steps in an workflow. The algorithm (Figure 2 in [5]) tries to gradually reduce the set of users that can be applied to each step. One first calculates the set of authorized users for each individual step, and then for each constraint that involves steps s_1 and s_2, one remove from the sets for steps s_1 and s_2 those users that cannot be paired with a user satisfying the constraint. If no set can be reduced further and no set is empty, the algorithm declares that a workflow is satisfiable. The problem with this algorithm is that, while it ensures that each individual constraint can be satisfied, it does not ensure the combination of them can. For a counter example, consider a workflow with 4 steps and 3 users, where every user is authorized to perform every step. The constraints are such that no two steps can be performed by the same user. Obviously, a valid execution assignment would not exist. However, the algorithm would return true. As we have pointed out in Theorem 6, the workflow satisfiability problem is $\mathbf{NP}$-hard in general for any workflow model that supports user-inequality constraints. Since the model in [5] supports such type of constraints, a polynomial time algorithm for the satisfiability problem in their model could not exist.

None of the work mentioned above have given the computational complexity results of the Workflow Satisfiability Problem, whereas we give a clear characterization using parameterized complexity. Also, the resiliency problem in workflow has not been studied before in the literature.

The concept of resiliency policies in access control is first formally proposed by Li et al. [8]. To our knowledge, this paper is the first to define and study resiliency problems in workflow systems. There are major difference between resiliency in workflow systems and the resiliency policies proposed in [8], and we have discussed the differences in Section 1.

7 Conclusion and Future Work

We have proposed a role-and-relation-based model (R^{2}BAC) for workflow systems, and have shown that the workflow satisfiability problem in R^{2}BAC is $\mathbf{NP}$-complete. We have also shown that the problem remains intractable for any workflow model that supports certain simple types of constraints such as user-inequality constraints and existence-equality constraints. We then apply tools from parameterized complexity to better understand the complexities of the problem. Furthermore, we have formally defined three levels of resiliency in workflow systems, namely, static resiliency, decremental resiliency and dynamic resiliency. We have also shown that checking whether a system configuration is statically resilient for a workflow is $\mathbf{NP}$-hard and is in the Polynomial Hierarchy, and the same problems for decremental resiliency and dynamic resiliency are $\mathbf{PSPACE}$-complete.

To our knowledge, this paper is the first to apply parameterized complexity theory to computational problems on access control policy analysis. As a number of policy analysis problems in access control have been shown to be $\mathbf{NP}$-complete, we believe that parameterized complexity theory can be fruitfully applied to these problems to shed insight on the causes of hardness in these problems as well as to give new algorithmic insights.

For the Workflow Satisfiability Problem, one future direction is to improve the fixed-parameter tractable algorithm described in Section 4.2, and another direction is to design and implement algorithms that can solve general case of the problem efficiently in practice. Resiliency is a relatively new concept in access control, and there are a number of interesting resiliency-related research topics. One open problem is how to efficiently generate optimal strategies to achieve decremental or dynamic resiliency in workflow systems. We believe that planning techniques developed in the artificial intelligence community will prove useful. Another direction is to study resiliency in workflow systems where multiple instances of the same workflow may be executed concurrently. Finally, it would be interesting to study how to design and specify delegation policies to achieve resiliency in workflow systems.

References

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

[2] We have verified the bug with the author of [5].

[2] V. Atluri and J. Warner. Supporting conditional delegation in secure workflow management systems. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 49–58, New York, NY, USA, 2005. ACM Press.

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

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

[5] J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies (SACMAT 2005)*, pages 38–47, Stockholm, Sweden, June 2005.

[6] R. Downey and M. Fellows. *Parameterized Complexity*. Springer, 1999.

[7] M. H. Kang, J. S. Park, and J. N. Froscher. Access control mechanisms for inter-organizational workflow. pages 66–74, 2001.

[8] N. Li, M. V. Tripunitara, and Q. Wang. Resiliency policies in access control. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, Nov. 2006.

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

[10] K. Tan, J. Crampton, and C. Gunter. The consistency of task-based authorization constraints in workflow systems. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 155–169, 2004.

[11] J. Warner and V. Atluri. Inter-instance authorization constraints for secure workflow management. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 190–199, 2006.

A Proofs in Section 3.1

Proof to Lemma 2: WSP is in $\mathbf{NP}$ in R^{2}BAC.

Proof. The length of a plan is bounded by the number of steps in the workflow. Given a plan for a workflow, checking whether a user is authorized to perform a step can be done in linear time. Also, checking whether a constraint is satisfied by the plan can be done in polynomial time. Hence, checking whether a plan is valid is in $\mathbf{P}$. A nondeterministic Turing machine can thus guess a plan and check whether it is valid in polynomial time. $\square$

Proof to Lemma 3: WSP is $\mathbf{NP}$-hard in R^{2}BAC, if the workflow uses constraints of the form $\langle \neq (s_1, s_2) \rangle$.

Proof. To prove the problem is $\mathbf{NP}$-hard, we reduce the $\mathbf{NP}$-complete GRAPH K-COLORABILITY problem to this problem. In the GRAPH K-COLORABILITY problem, we are given a graph $G(V, E)$ and an integer k, and asked whether we can assign no more than k colors to vertices in V such that every vertex has one color and vertices n_i and n_j have different colors whenever $(n_i, n_j) \in E$.

Given a graph $G(V, E)$, we construct a workflow $W = \langle S, \preceq, SA, C \rangle$ and a configuration $\Gamma = \langle U, UR, B \rangle$ such that there is a one-to-one correspondence between steps in S and vertices in V. Let $U = \{u_1, \cdots, u_k\}$, where each $u_i \in U$ corresponds to a color. Construct UR and SA in such a way that every user in U is authorized to perform every step in S. For every $(n_i, n_j) \in E$, construct a constraint $\langle \neq (s_i, s_j) \rangle$, which requires that s_i and s_j must be performed by different users. If G is k-colorable, then we can construct a plan P such that s_j is performed by u_i if and only if n_j is assigned the ith color. Since no pair of adjacent vertices have the same color, no pair of steps restricted by a constraint is assigned to the same user in P. Hence, P satisfies all constraints and is a valid plan. Similarly, if there is a valid plan P for W in Γ, we can find a way to color G with no more than k colors according to P. In general, G is k-colorable if and only if W is satisfiable. $\square$

Proof to Lemma 4: WSP is $\mathbf{NP}$-hard in R^{2}BAC, if the workflow uses constraints of the form $\langle = (s, \exists X) \rangle$.

Proof. To prove the problem is $\mathbf{NP}$-hard, we reduce the $\mathbf{NP}$-complete HITTING SET problem to this problem. In the HITTING SET problem, we are given a set Z and a family $F = \{Z_1, \cdots, Z_m\}$ of subsets of Z and asked whether there exists a size-k subset H of Z such that, for every $Z_i \in F$, $H \cap Z_i \neq \emptyset$.

We construct a workflow $W = \langle S \cup A, \preceq, SA, C\rangle$ and a configuration $\Gamma = \langle U, UR, B\rangle$ such that the answer to the HITTING SET problem is "yes" if and only if W is satisfiable under Γ. Let $U = \{u_i \mid e_i \in Z\}$ be a set of users. Let $S = \{s_1, \cdots, s_k\}$ be a set of k steps. Construct UR and SA in such a way that every step in S is authorized to all users in U. Furthermore, let $A = \{a_1, \cdots, a_m\}$ be a set of m steps and $S \cap A = \emptyset$. Construct UR and SA in such a way that u_i is authorized to perform a_j if and only if $e_i \in Z_j$. Construct a set $C = \{c_1, \cdots, c_m\}$ of m constraints, where $c_i = \langle = (a_i, \exists S)\rangle$.

On the one hand, assume that P is a valid plan. Let $H = \{e_i \mid \exists_{s_j}(u_i, s_j) \in P\}$. For every $a_i \in A$, let u_j be the user such that $(u_j, a_i) \in P$. P being valid indicates that u_j is authorized to perform a_i. From our construction, we have $e_j \in Z_i$. Furthermore, for every $i \in [1, m]$, $\langle = (a_i, \exists S)\rangle$ being satisfied indicates that there exists $s_l \in S$ such that $(u_j, s_l) \in P$. And $(u_j, s_l) \in P$ indicates that $e_j \in H$. Therefore, we have $H \cap Z_i = e_j$. In general, for every $Z_i \in F$, $H \cap Z_i \neq \emptyset$. The answer to the HITTING SET problem is "yes".

On the other hand, assume that the answer to the HITTING SET problem is "yes". We now construct a plan P that satisfies the workflow. Without loss of generality, assume that $H = \{e_1, \cdots, e_k\}$. We initialize P to $\emptyset$ and add (u_i, s_i) to P for every $i \in [1, k]$. Recall that s_i is authorized to every user in U. For every $Z_j \in F$, add (u_i, a_j) to P when $H \cap Z_j = e_i$. $e_i \in Z_j$ implies that u_i is authorized to perform a_j. Furthermore, for every $c_j \in C$, $(u_i, a_j) \in P$ and $(u_i, s_i) \in P$ indicate that c_j is satisfied. Therefore, P is a valid plan. □

Proof to Lemma 5: WSP is in $\mathbf{P}$ in R^{2}BAC, if the workflow only has constraints in the forms of $\langle = (s_1, s_2)\rangle$, $\langle = (s, \forall X)\rangle$ or $\langle = (\forall X, s)\rangle$.

Proof. As "=" is transitive, constraint $\langle = (\forall X, s)\rangle$ can be equivalently rewritten as $\langle = (s, \forall X)\rangle$. Furthermore, constraint $\langle = (s_1, s_2)\rangle$ can be equivalently rewritten as $\langle = (s_1, \forall\{s_2\})\rangle$. Therefore, we just need to consider constraints in the form of $\langle = (s, \forall X)\rangle$.

Given a constraint $c = \langle = (s, \forall X)\rangle$, let $ST(c) = \{s\} \cup X$ be the set of steps restricted by c. c requires that all steps in $ST(c)$ be performed by the same users. If there exist two constraints c_1 and c_2 such that $ST(c_1) \cap ST(c_2) \neq \emptyset$, we can replace c_1 and c_2 with a third constraint that requires all steps in $ST(c_1) \cup ST(c_2)$ be performed by the same user. Without loss of generality, we assume that for every pair of constraints (c_1, c_2) in the workflow, we have $ST(c_1) \cap ST(c_2) = \emptyset$. In this case, we may assign users to steps in $ST(c_1)$ independently from steps in $ST(c_2)$. Given a step s, let $AU(s)$ be the set of users authorized to perform step s. Assume that $ST(c_1) = \{s_0, s_1, \cdots, s_m\}$. Steps in $ST(c_1)$ can only be performed by a user in $U' = AU(s_0) \cap AU(s_1) \cap \cdots \cap AU(s_m)$. If $U' = \emptyset$, then c_1 cannot be satisfied. Therefore, to check whether W is satisfiable, we just need to compute set intersection for every constraint in W. And set intersection can be done in polynomial time. □

B Proofs in Section 4

Proof to Theorem 9: WSP is in $\mathbf{FPT}$ in R^{2}BAC, if = and $\neq$ are the only binary relations used by constraints in the workflow.

Proof. Given a workflow $W = \langle S, \preceq, SA, C\rangle$ and a configuration $\Gamma = \langle U, UR, B\rangle$, let k be the number of steps in W. The description of the algorithm is as follow.

1. For every step $s_i \in S$, compute the set $AU(s_i)$ of users authorized to perform s_i according to UR and SA.
2. Process every constraint c in the form of $\langle = (s_1, s_2)\rangle$ and $\langle = (s, \forall X)\rangle$. Let $ST(c)$ be the set of steps appearing in c. Without loss of generality, assume that $ST(c) = \{s_1, \cdots, s_m\}$. Let $U' = \bigcap_{j=1}^{m} AU(s_j)$ be the set of users who are authorized to perform all steps in $ST(c)$. If $U' = \emptyset$, then c is not satisfiable and neither nor W. Otherwise, for every $s_i \in ST(c)$, $AU(s_i)$ is set to be U'.
3. Let C' be the set of all constraints in C that are in the form of $\langle = (s, \exists X)\rangle$. We construct a search tree as follow.

   Label the root of the tree with C' and $UA = \{AU(s_i) \mid s_i \in S\}$. Choose a constraint c from C'. Without loss of generality, assume that $c = \langle = (s_0, \exists\{s_1, \cdots, s_m\})\rangle$ $(m \leq k-1)$. s_0 must be performed by the same user as s_1, or s_2, $\cdots$, or s_m. We create m children of the root corresponding to these m possibilities. Let $U_{0,1} = AU(s_0) \cap AU(s_1)$. If $U_{0,1} = \emptyset$, then the first child of the root is marked as "invalid" and will not be further processed, since it is impossible to find a user who is authorized to perform both s_0 and s_1. Otherwise, the first child is labeled with $C' - \{c\}$ and UA_1, where UA_1 is the same as UA except that $AU(s_0)$ and $AU(s_1)$ are set to be $U_{0,1}$. Intuitively, the set of constraints labeling a node represents the remaining constraints, while the set UA labeling a node represents the user-step authorization that satisfies those constraints that have been processed. The other $m-1$ children of the root are processed similarly. We then recursively process the children of the children of the root and so on until all nodes in the tree have been processed. We then say that the search tree is fully-developed.

In a fully-developed search tree, a leave node that is not marked as "invalid" (in this case, it must have been labeled with an empty set of constraints) is called "alive". If there is no "alive" leave node in the search tree, then it is impossible to satisfy all constraints in C' and thus W is not satisfiable.

Note that there are no more than $k2^{k-1}$ different constraints in the form of $\langle = (s, \exists X) \rangle$ as s and X can take at most k and 2^{k-1} different values, respectively [3]. Therefore, the depth of the fully-developed search tree is no more than $k2^{k-1}$. Furthermore, the number of children of each node is bounded by $k-1$. Hence, the size of the fully-developed search tree is bounded by $(k-1)^{k2^{k-1}}$. Processing each node in the search tree involves computing no more than $k-1$ intersections and can be done in $O(kn)$.

4. For each "alive" leave node v in the search tree, we check whether all constraints using $\neq$ can be satisfied with the user-step authorization UA labeling v. According to Lemma 8, this can be done in $O(k^{k+1}n)$, where n is the size of the entire input to the problem.

In general, the above algorithm finishes in $O(f(k)n)$ where $f(k) = k^{k+1}(k-1)^{k2^{k-1}}$. Hence, the problem is in **FPT**. $\square$

Proof to Theorem 10: WSP is $W[1]$-hard in R^{2}BAC if user-defined binary relations are used in constraints.

Proof. A constraint $\langle \rho(s_1, s_2) \rangle$ can be equivalently represented as $\langle \rho(s_1, \forall\{s_2\}) \rangle$ or $\langle \rho(s_1, \exists\{s_2\}) \rangle$. Hence, we just need to prove that WSP is $W[1]$-hard even if the workflow only has constraints in the form of $\langle \rho(s_1, s_2) \rangle$.

We reduce INDEPENDENT SET to WSP. In INDEPENDENT SET, we need to determine whether there is a size-k independent set in graph $G(V, E)$. An independent set of G is a set of vertices V' such that $V' \subseteq V$ and no pair of vertices in V' are adjacent to each other in G. INDEPENDENT SET with parameter k is $W[1]$-complete.

Given an integer k and a graph $G(V, E)$ where $V = \{v_1, \cdots, v_m\}$, we construct a workflow $W = \langle S, \preceq, SA, C \rangle$ and a configuration $\Gamma = \langle U, UR, B \rangle$, where $U = \{u_1, \cdots, u_m\}$. There is a one-to-one correspondence between users in U and vertices in V. $S = \{s_1, \cdots, s_k\}$ and $s_1 \preceq \cdots \preceq s_k$. Let $UR = \{(u_i, r) \mid i \in [1, m]\}$ and $SA = \{(r, s_i) \mid i \in [1, k]\}$. In other words, every user in U is authorized to perform every step in S. B contains one binary relation ρ, and $\rho = \{(u_i, u_j) \mid i \neq j \wedge (v_i, v_j) \notin E\}$. Intuitively, $(u_i, u_j) \in \rho$ if and only if $u_i \neq u_j$ and the vertices corresponding to the two users are not adjacent to each other in G. For every $i \in [2, k]$, we construct $i-1$ constraints $c_{i,1}, \cdots, c_{i,i-1}$ such that $c_{i,j} = \langle \rho(s_i, s_j) \rangle$ where $j \in [1, i-1]$.

Next, we show that G has a size-k independent set if and only if W is satisfiable under Γ.

On the one hand, without loss of generality, assume that $\{v_1, \cdots, v_k\}$ is an independent set of G. By definition of independent set, we have $(v_i, v_j) \notin E$, for any $i, j \in [1, k]$ and $i \neq j$. We construct a plan $P = \{(u_i, s_i) \mid i \in [1, k]\}$. For any $i, j \in [1, k]$ and $i \neq j$, $(v_i, v_j) \notin E$ implies that $(u_i, u_j) \in \rho$. Therefore, no constraint is violated by P and P is a valid plan.

On the other hand, assume that there is a valid plan P for W. From the construction of ρ, the k steps in W must be performed by k different users. Without loss of generality, assume that $P = \{(u_i, s_i) \mid i \in [1, k]\}$. Since no constraint is violated by P, we have $(u_i, u_j) \in \rho$ for any $i, j \in [1, k]$ and $i < j$. Let $V' = \{v_1, \cdots, v_k\}$. For any pair of vertices (v_i, v_j) where $i, j \in [1, k]$ and $i < j$, $(u_i, u_j) \in \rho$ implies that $(v_i, v_j) \notin E$. Hence, V' is a size-k independent set of G.

Finally, we show that the above reduction is a fixed-parameter reduction. In our reduction, the parameter k of the INDEPENDENT SET instance has the same value as the number of steps in the corresponding WSP instance. Furthermore, the number m of users in the workflow is the same as the number of vertices in the graph. ρ can be generated in quadratic time to the size of G. There are no more than $k^2/2$ constraints in the workflow, and UR contains m items while SA contains k items. In general, the WSP instance can be generated from the INDEPENDENT SET instance in $O(n^2 + k^2)$, where n is the size of graph G. Hence, the reduction is a fixed-parameter reduction. $\square$

Proof to Theorem 11:WSP in R^{2}BAC is in $W[2]$.

[3]The 2^{k-1} upper-bound is loose and can be improved, but it suffices to prove the result we want.

Proof. We can reduce WSP to the weighted satisfiability problem of decision circuits of weft 2 (denoted as WCS[2]). In the following, we encode an WSP instance into a boolean expression that can be represented as a decision circuit of weft 2. And the answer to the WSP instance is "yes" if and only if the answer to the WCS[2] instance is "yes".

Given a workflow $W = \langle S, \preceq, SA, C\rangle$ and a configuration $\Gamma = \langle U, UR, B\rangle$, let $S = \{s_1, \cdots, s_k\}$ and $U = \{u_1, \cdots, u_n\}$. We construct kn variables $v_{i,j}$ where $i \in [1, k]$ and $j \in [1, n]$. Intuitively, setting $v_{i,j}$ to true corresponds to assigning user u_j to s_i.

Let $AU(s)$ be the set of authorized users for step s. For every $s_i \in S$, we construct a clause $H_{s_i} = \bigvee_{u_j \in AU(s_i)} v_{i,j}$, which indicates that s_i must be performed by an authorized user. The length of such a clause is no more than n and there are k such clauses. Note that a weight-k truth assignment satisfying these k clauses must set exactly one $v_{i,j}$ to true for every $i \in [1, k]$, which indicates that every step is assigned to exactly one user.

For every constraint $c \in C$, we construct clauses for c as follows. Given a set $F = \{f_1, \cdots, f_m\}$ of clauses, we define $\bigvee F$ as $f_1 \vee \cdots \vee f_m$ and $\bigwedge F$ as $f_1 \wedge \cdots \wedge f_m$.

- When $c = \langle\rho(s_{i_1}, s_{i_2})\rangle$: Let $F = \{v_{i_1,j_1} \wedge v_{i_2,j_2} \mid u_{j_1} \in AU(s_{i_1}) \wedge u_{j_2} \in AU(s_{i_2}) \wedge (u_{j_1}, u_{j_2}) \in \rho\}$. We construct a clause $H_c = \bigvee F$, which indicates that s_{i_1} and s_{i_2} must be performed by a pair of authorized users that satisfies ρ.
- When $c = \langle\rho(s, \exists X)\rangle$: Without loss of generality, assume that $c = \langle\rho(s_0, \exists\{s_1, \cdots, s_m\})\rangle$. For every $i \in [1, m]$, let $F_i = \{v_{0,j_1} \wedge v_{i,j_2} \mid u_{j_1} \in AU(s_0) \wedge u_{j_2} \in AU(s_i) \wedge (u_{j_1}, u_{j_2}) \in \rho\}$. We construct a clause $H_c = \bigvee F_1 \vee \cdots \vee \bigvee F_m$, where $\bigvee F_i$ indicates that s_0 and s_i must be performed by a pair of authorized users that satisfies ρ.
- When $c = \langle\rho(s, \forall X)\rangle$: Without loss of generality, assume that $c = \langle\rho(s_0, \forall\{s_1, \cdots, s_m\})\rangle$. For every $i \in [1, m]$, let $F_i = \{v_{0,j_1} \wedge v_{i,j_2} \mid u_{j_1} \in AU(s_0) \wedge u_{j_2} \in AU(s_i) \wedge (u_{j_1}, u_{j_2}) \in \rho\}$. We construct a clause $H_c = \bigvee F_1 \wedge \cdots \wedge \bigvee F_m$, where $\bigvee F_i$ indicates that s_0 and s_i must be performed by a pair of authorized users that satisfies ρ.

Let $F = \{H_{s_i} \mid s_i \in S\} \cup \{H_c \mid c \in C\}$. $H = \bigwedge F$ is a clause encoding the WSP instance in the sense that H has a weight-k satisfying truth assignment if and only if W is satisfiable under Γ. H can be represented by a decision circuit using a large "$\wedge$" gate that connects a number of large "$\vee$" gates that connect either a number of variables or a number of small "$\wedge$" gates, each of which connects two variables. The decision circuit is thus a weft 2 decision circuit.

In the above reduction, the number of step in the WSP instance is the same as the weight k of the corresponding WCS[2] instance. There are k H_s clauses, each of which has length no more than n, where n is the size of the WSP instance. And there are n H_c clauses, each of which has length no more than kn^2. Hence, the construction of the decision circuit can be done in $O(kn^3)$. Therefore, the above reduction is a fixed-parameter reduction and WSP is in $W[2]$. □

C Background on Oracle Turing Machines and Polynomial Hierarchy

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

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

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

and for all $k \geq 0$,

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

Some classes in the hierarchy are

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

D Proofs in Section 5.1

Proof to Theorem 14: CRCP is $\mathbf{PSPACE}$-complete.

Proof. The two-person game of decremental resiliency has the following two properties, which indicates that it can be solved in $\mathbf{PSPACE}$.

1. The number of rounds is bounded by a polynomial in the size of the input. In particular, the game must come to a conclusion after at most k rounds, where k is the number of steps in the workflow.
2. Given an intermediate state, which consists of a partial plan, the set of remaining users and the set of unfinished steps, there is a polynomial-space algorithm that constructs all possible combinations of actions of the two users in the next round, and determines if the game is over.

To show PSPACE-hardness, we reduce the $\mathbf{PSPACE}$-complete QUANTIFIED SATISFIABILITY (or QSAT) problem to CRCP. In the QSAT, we are given a boolean expression ϕ in conjunction normal form (CNF), with boolean variables $x_1, \cdots, x_m$. Is it true that there is a truth value for x_1 such that for both truth value of x_2 there exists a truth value for x_3, and so on up to x_m, ϕ is satisfied by the overall truth assignment? In other words,

$$\exists_{x_1} \forall_{x_2} \exists_{x_3} \cdots Q_{x_m} \phi ?$$

where Q is "exists" if m is odd, or "for all" if m is even. Without loss of generality, we assume that m is odd.

The QSAT problem can be modeled as a two-person game, in which Player 1 and Player 2 control the truth assignment of variables in $\{x_i \mid i \in [1,m] \wedge i \text{ is odd}\}$ and $\{x_j \mid j \in [1,m] \wedge j \text{ is even}\}$, respectively. Player 1 tries to satisfy ϕ, while Player 2 tries to prevent this.

Given a QSAT instance $\exists_{x_1} \forall_{x_2} \exists_{x_3} \cdots \exists_{x_m} \phi$ where $\phi = \phi_1 \wedge \cdots \wedge \phi_k$, we construct a CRCP instance. The detailed construction of the CRCP instance is given in Figure 2.

Next, we prove that the answer to the CRCP instance is "yes" if and only if the answer to the QSAT instance is "yes". In the constructed workflow $W = \langle S, \preceq, SA, C \rangle$, S consists of three parts A, B and D. Steps in A determine truth values of variables $x_1, \cdots, x_m$. Intuitively, assigning user u_i (or v_i) to a_i represents setting x_i to "true" (or "false"). Steps in B correspond to the k clauses in ϕ. Steps in D are used to restrict the behaviors of the two players.

We need to to prove the following four claims.

1. For every even number $i \in [1,m]$, Player 2 should remove either u_i or v_i right after the execution of a_{i-1}. In other words, Player 2 controls the user-step assignment for steps in $\{a_i \mid a_i \in A \wedge i \text{ is even}\}$.
2. For every step $a_i \in A$, Player 1 should assign either u_i or v_i to a_i, when Player 2 plays optimally.
3. If Player 1 plays optimally, then steps in D can always be completed.
4. If both players play optimally, all steps in B can be completed if and only if the truth assignment of boolean variables $x_1, \cdots, x_m$ corresponding to the user-step assignment of steps in A satisfies ϕ.

If Claim 1 and Claim 2 are true, then Player 1 and Player 2 control the truth assignment of variables in $\{x_i \mid i \in [1,m] \wedge i \text{ is odd}\}$ and $\{x_j \mid j \in [1,m] \wedge j \text{ is even}\}$, respectively. If Claim 3 and Claim 4 are true, then the workflow instance can be completed if and only if ϕ is satisfied by the truth assignment. In general, the answer to the CRCP instance is "yes" if and only if the answer to the QSAT instance is "yes".

The proofs to the four claims are listed as follows.

Proof to Claim 1: First of all, since the total number of absent users is bounded by t, Player 2 should not remove any of those users with $t+1$ copies. Users in $\{u_i, v_i \mid i \in [1,m] \wedge i \text{ is even}\}$ are unique and are the only users with less than than $t+1$ copies.

Secondly, given an even number i, if Player 2 removes both u_i and v_i, then there must exist an even number $j \in [1,m]$ such that both u_j and v_j are available throughout the game, as Player 2 can remove at most $(m-1)/2$ users. In this case, Player 1 can assign u' to all remaining even steps in A as well as all steps in B, and then assign u_j to d_1 and v_j to d_2. Such an assignment complete the workflow without violating any constraint. Player 1 wins. Therefore, Player 2 should remove either u_i or v_i for every even number i.

Finally, we would like to point out that Player 2 should remove u_i or v_i before the execution of a_i, where i is a even number. If Player 2 does this after the execution of a_i, then Player 1 gains advantage by being able to choose between u_i and v_i for a_i. However, removing u_i or v_i after a_i does not affect future user-step assignment, as it is p_i and q_i rather than u_i and v_i that will be performing steps in B.

Proof to Claim 2: The statement is true when i is odd, since u_i and v_i are the only users authorized to perform a_i. In the following, we only discuss the case when i is even.

From Claim 1, when Player 2 plays optimally, he/she removes either u_i or v_i for every even number $i \in [1, m]$. Given an even number i, without loss of generality, assume that Player 2 removes u_i. In this case, Player 1 may either assign v_i or u' to a_i. If, by contradiction, Player 1 assigns u' to a_i, then according to constraint $\langle \overline{\rho_1}(d_1, \forall A_{even}) \rangle$, Player 1 cannot assign v' to d_1 as $(v', u') \in \rho_1$. Thus, Player 1 has to choose a certain u_j or v_j for d_1, where j is even. By the time d_1 is to be executed, either u_j or v_j must have been removed by Player 2. Without loss of generality, assume that u_j is available and is thus assigned to d_1. According to $\langle \rho_2(d_2, d_1) \rangle$, Player 1 has to assign v_j to d_2, but v_j is not available. Hence, d_2 cannot be completed and Player 1 losses. Therefore, Player 1 must not assign u' to a_i when Player 2 plays optimally. The only choice for Player 1 is to assign v_i to a_i.

Proof to Claim 3: We have shown that if Player 2 does not follow the strategy in Claim 1, then Player 1 can complete all steps in the workflow. When both players play optimally, according to Claim 1 and Claim 2, only users in $\{u_i, v_i \mid i \in [1, m]\}$ are assigned to steps in A. In this case, Player 1 can assign v' to d_1 and u' to d_2 without violating any constraints.

Proof to Claim 4: From Claim 1 and Claim 2, when both players play optimally, only users in $\{u_i, v_i \mid i \in [1, m]\}$ are assigned to steps in A. For any $b_j \in B$, according to constraint $\langle \rho_0(b_j, \exists A) \rangle$, u' cannot be assigned to b_j, as $(u', u_i), (u', v_i) \notin \rho_0$. Furthermore, p_i and q_i correspond to u_i and v_i respectively according to ρ_0. From the construction of SA and UR, p_i (or q_i) is authorized to perform b_j if and only if setting x_i to true (or false) satisfies clause ϕ_j. Hence, Player 1 can assign a user to b_j if and only if the truth assignment determined by the user-step assignment of steps in A satisfies ϕ_j. In general, all steps in B can be completed if and only if ϕ_j is satisfied for every $j \in [1, k]$, which indicates that ϕ is satisfied. □

Proof to Theorem 15: DRCP is $\mathbf{PSPACE}$-complete.

Proof. The proof that DRCP is in $\mathbf{PSPACE}$ is similar to the case of CRCP. In the following, we only prove that the problem is $\mathbf{PSPACE}$-hard.

We reduce the $\mathbf{PSPACE}$-complete QUANTIFIED SATISFIABILITY (or QSAT) problem to DRCP. Given a QSAT instance $\exists_{x_1} \forall_{x_2} \exists_{x_3} \cdots \exists_{x_m} \phi$ where $\phi = \phi_1 \wedge \cdots \wedge \phi_k$, we construct a DRCP instance. The detailed construction of the DRCP instance is given in Figure 3.

We need to prove that the answer to the DRCP instance is "yes" if and only if the answer to the QSAT instance is "yes". In the constructed workflow $W = \langle S, \preceq, SA, C \rangle$, S consists of two parts A and B. Steps in A determine truth values of variables $x_1, \cdots, x_m$. Intuitively, assigning user u_i (or v_i) to a_i represents setting x_i to "true" (or "false"). Steps in B correspond to the k clauses in ϕ.

First of all, it is clear that Player 2 should remove one user in each round. However, since there are two copies of u_i and v_i for odd number $i \in [1, m]$, and two copies of p_j and q_j for $j \in [1, k]$, Player 2's action only affects the user-step assignment of a_i for even number $i \in [1, m]$. Therefore, Player 1 and Player 2 has control over the user-step assignment of odd number steps in A and even number steps in A, respectively. A user-step assignment for steps in A represents a truth assignment for variables $x_1, \cdots, x_m$.

Secondly, according to relation ρ, p_i and q_i correspond to u_i and v_i respectively. According to the construction of SA and UR, p_i (or q_i) is authorized to perform b_j if and only if setting x_i to true (or false) satisfies clause ϕ_j. Due to the constraint $\langle \rho(b_j, \exists A) \rangle$, Player 1 can assign a user to b_j if and only if the truth assignment determined by user-step assignment in A satisfies ϕ_j. Therefore, Player 1 can complete all steps in B if and only if the truth assignment satisfies ϕ.

In general, Player 1 can always win the game if and only if the answer to the QSAT instance is "yes". □

Input:

$\exists_{x_1}\forall_{x_2}\exists_{x_3}\cdots\exists_{x_m}\phi$, where $\phi = \phi_1 \wedge \cdots \wedge \phi_k$

Output:

A workflow $W = \langle S, \preceq, SA, C\rangle$, an integer $t = (m-1)/2$, a configuration $\Gamma = \langle U, UR, \{\rho_0, \rho_1, \rho_2\}\rangle$

Construction of W and Γ:

- **Steps and Step-Authorization:**

  $S = A \cup B \cup D$

  We have $A = \{a_1, \cdots, a_m\}$, $B = \{b_1, \cdots, b_k\}$, $D = \{d_1, d_2\}$, and $a_1 \preceq \cdots \preceq a_m \preceq d_1 \preceq d_2 \preceq b_1 \preceq \cdots \preceq b_k$

  $SA = \{(r_{a_i}, a_i) \mid a_i \in A\} \cup \{(r_{b_i}, b_i) \mid b_i \in B\} \cup \{(r_{d_1}, d_1), (r_{d_2}, d_2)\}$

- **Configuration:**

  $U = \{u_i, v_i, p_i, q_i \mid i \in [1, m]\} \cup \{u', v'\}$

  For every odd number i in $[1, m]$, there are $t+1$ copies of u_i and v_i. For every $j \in [1, m]$, there are $t+1$ copies of p_j and q_j. There are $t+1$ copies of u' and v' as well.

$$\begin{aligned} UR = &\{(u_i, r_{a_i}), (v_i, r_{a_i}) \mid i \in [1, m] \wedge i \text{ is odd}\} \\ &\cup \{(u_i, r_{a_i}), (v_i, r_{a_i}), (u', r_{a_i}) \mid i \in [1, m] \wedge i \text{ is even}\} \\ &\cup \{(u', r_{b_i}) \mid i \in [1, m]\} \cup \Upsilon_1 \cup \cdots \cup \Upsilon_k \\ &\cup \{(v', r_{d_1})\} \cup \{(u_i, r_{d_1}), (v_i, r_{d_1}) \mid i \in [1, m] \wedge i \text{ is even}\} \\ &\cup \{(u', r_{d_2})\} \cup \{(u_i, r_{d_2}), (v_i, r_{d_2}) \mid i \in [1, m] \wedge i \text{ is even}\} \end{aligned}$$

  Construction of Υ_i: Let L_i be the set of literals in clause ϕ_i. $(p_j, r_{b_i}) \in \Upsilon_i$ if and only if there exists a literal $l \in L_i$ such that $l = x_j$; and $(q_j, r_{b_i}) \in \Upsilon_i$ if and only if there exists a literal $l \in L_i$ such that $l = \neg x_j$.

- **Constraints:**

  $C = \{\langle \rho_0(b_i, \exists A)\rangle \mid i \in [1, k]\} \cup \{\langle \overline{\rho_1}(d_1, \forall A_{even})\rangle, \langle \rho_2(d_2, d_1)\rangle\}$

  where $A_{even} = \{a_i \mid i \text{ is even}\}$. We have
  - $\rho_0 = \{(p_i, u_i), (q_i, v_i) \mid i \in [1, m]\} \cup \{(u', u')\}$
  - $\rho_1 = \{(v', u')\}$
  - $\rho_2 = \{(u_i, v_i), (v_i, u_i) \mid i \text{ is even}\} \cup \{(u', v')\}$.

Figure 2. Generating a CRCP instance for a QSAT instance.

Input:

$\exists_{x_1}\forall_{x_2}\exists_{x_3}\cdots\exists_{x_m}\phi$, where $\phi = \phi_1 \wedge \cdots \wedge \phi_k$

Output:

A workflow $W = \langle S, \preceq, SA, C\rangle$, an integer $t = 1$, a configuration $\Gamma = \langle U, UR, \{\rho\}\rangle$

Construction of W and Γ:

- **Steps and Step-Authorization:**

  $S = A \cup B$

  We have $A = \{a_1, \cdots, a_m\}$, $B = \{b_1, \cdots, b_k\}$, and $a_1 \preceq \cdots \preceq a_m \preceq b_1 \preceq \cdots \preceq b_k$

  $SA = \{(r_{a_i}, a_i) \mid a_i \in A\} \cup \{(r_{b_i}, b_i) \mid b_i \in B\}$

- **Configuration:**

  $U = \{u_i, v_i, p_i, q_i \mid i \in [1, m]\}$

  For every odd number i in $[1, m]$, there are 2 copies of u_i and v_i. For every $j \in [1, m]$, there are 2 copies of p_j and q_j.

  $$UR = \{(u_i, r_{a_i}), (v_i, r_{a_i}) \mid i \in [1, m]\} \cup \Upsilon_1 \cup \cdots \cup \Upsilon_k$$

  Construction of Υ_i: Let L_i be the set of literals in clause ϕ_i. $(p_j, r_{b_i}) \in \Upsilon_i$ if and only if there exists a literal $l \in L_i$ such that $l = x_j$; and $(q_j, r_{b_i}) \in \Upsilon_i$ if and only if there exists a literal $l \in L_i$ such that $l = \neg x_j$.

- **Constraints:**

  $C = \{\langle \rho(b_i, \exists A)\rangle \mid i \in [1, k]\}$
  where $\rho = \{(p_i, u_i), (q_i, v_i) \mid i \in [1, m]\}$

Figure 3. Generating a DRCP instance for a QSAT instance.