RZ 3815 (# Z1202-001) 02/28/2012
Computer Science 21 pages

Research Report

Optimal Workflow-Aware Authorizations

D. Basin‡, S.J. Burri‡*, G. Karjoth*

‡Swiss Federal Institute of Technology (ETH),
Zurich, Switzerland

*IBM Research – Zurich
8803 Rüschlikon
Switzerland

Research
Almaden • Austin • Brazil • Cambridge • China • Haifa • India • Tokyo • Watson • Zurich

Optimal Workflow-aware Authorizations

David Basin, ETH Zurich
Samuel J. Burri, ETH Zurich and IBM Research – Zurich
Günter Karjoth, IBM Research – Zurich

Abstract

Balancing protection and empowerment is a central problem when specifying authorizations. The principle of *least privilege*, the classical approach to balancing these two conflicting objectives, says that users shall only be authorized to execute the tasks necessary to complete their job. However, when there are multiple authorization configurations satisfying least privilege, which one should be chosen?

In this paper, we model the tasks that users must execute as workflows, and the risk and cost associated with authorization configurations and their administration. We then formulate the balancing of empowerment and protection as an optimization problem: finding a cost-minimizing authorization configuration that enables a successful workflow execution. We show that finding an optimal solution for a role-based cost function is **NP**-complete. We support our results with a series of examples, which we also use to measure the performance of our prototype implementation.

1 Introduction

Authorizations, which govern users' access to resources, have a dual nature: they express what actions may and must not occur. In this way, they empower users to execute job-relevant tasks while protecting the integrity and confidentiality of resources. The question naturally arises as to how to best balance protection and empowerment.

The classical answer to this question is the principle of *least privilege* [18], which says that users shall only be authorized to execute the tasks necessary to complete their job. However, business processes that require the execution of multiple tasks by different users can have multiple authorization configurations (representing different authorization policies) that satisfy least privilege. Furthermore, the choice of an authorization configuration may be influenced by the cost associated with the respective administrative change. Thus, although least privilege is a guiding principle, it does not provide the final answer to the question of how to balance protection and empowerment.

In this paper, we present a new approach to answering this question by mapping authorization administration to an optimization problem. Specifically, we model business activities as tasks, structured as workflows. Authorizations then specify which users may execute which tasks. We distinguish between *history-dependent* and *history-independent* authorizations. History-dependent authorizations constrain task executions based on past task executions. Examples are *Separation of Duty (SoD)* and *Binding of Duty (BoD)* constraints. SoD, also known as Four-Eyes-Principle, aims at preventing fraud and errors by requiring a set of critical tasks to be executed by multiple users, whereas BoD requires a set of tasks to be executed by the same

user to limit the exposure of sensitive data and to reuse knowledge. In contrast, the evaluation of history-independent authorizations is not influenced by an execution history. Examples of history-independent authorization constraints are access control lists (ACLs), the Bell-LaPadula Model (BLP) [6], and Role-based Access Control (RBAC) configurations [10] without sessions.

We assume that history-independent authorizations may change during workflow execution. Reasons for this include organizational changes such employees joining or leaving the company or being promoted. In contrast, history-dependent authorization constraints do not change during workflow execution. However, due to their dependence on the expanding execution history, their evaluation may change during workflow execution.

We model the cost of changing from one history-independent authorization configuration to another one by a binary function. This function may account for the cost of the administrative activity associated with the change, the cost of maintaining the new configuration, and the risk associated with the new configuration. We consider minimizing risk to be equivalent to maximizing protection.

Let W be a workflow, H an execution history corresponding to an instance of W, ϕ some history-dependent authorizations, c a history-independent authorization configuration, and cost a function as described above. We investigate the problem

$$\min_{c'} \{\mathsf{cost}(c, c') \mid c' \text{ allows a successful completionof } W \text{ after } H \text{ that satisfies } \phi\} ,$$

where c' ranges over all feasible history-independent authorization configurations. The requirement of "getting the job done" becomes the feasibility condition and cost serves as the the objective function of the optimization problem. Hence, we reduce the question of how to balance empowerment and protection to the problem of finding a feasible configuration that maximizes protection, minimizes the cost associated with the administrative change, and empowers users to do their job while satisfying the authorizations.

We proceed by formalizing workflows, their execution history, and authorization constraints. Workflows, also known as business processes, provide a realistic abstraction for capturing what authorizations users need to get their work done, *i.e.* empowerment. As this paper's focus is not authorization-constrained workflows *per se*, we borrow the constraint model from [5]. In the interest of keeping our formalization concise and not letting the complexity of workflow constraints overshadow the optimization problem's complexity, we abstract from [5]'s process algebraic models and build directly on its graph-based approximations. Based on our formalization and the first generic definition of a cost function, we formally define the optimization problem sketched above.

To demonstrate the applicability of our general approach to a realistic business scenario, in a second step we refine the cost function using roles. The additional structure enables us to map the problem of balancing empowerment and protection to the well-established Integer Linear Programming Problem (**ILP**). A proof of our mapping's soundness and completeness enables us to use off-the-shelf software for **ILP** to compute the optimal authorization configuration that enables a successful execution of a given workflow. We use a running example to illustrate our results and to measure the performance of our mapping's implementation.

Our main contribution is to refine the decision problem of whether a given authorization configuration enables a successful workflow execution to the notion of an *optimal* authorization configuration that satisfies this property. Our approach provides considerable modeling freedom in terms of the notion of optimality used. For example, we may aim to minimize the

cost associated with a configuration change or maximize the protection resulting from the new configuration. We thereby facilitate a fine-grained balancing of empowerment and protection with respect to various criteria. Moreover, we prove that finding a role-based optimal authorization configuration that enables a workflow execution is **NP**-complete. Finally, our work shows how well-established results from optimization theory can be applied to information security, in particular access control.

The remainder of this paper is structured as follows. In Section 2, we provide background on **ILP** and graph coloring. In Section 3, we formalize workflows and authorizations that constrain their execution. In Section 4, we first present the general problem of finding an optimal authorization configuration that enables a workflow execution. Afterwards we refine the general problem, assuming a role-based cost function. We present related work in Section 5 and conclude in Section 6. The appendix provides proofs.

2 Background

We denote by $\mathbb{N}$ the set of natural numbers, by $\mathbb{Z}$ the set of integers, and by $\mathbb{R}$ the set of real numbers. Assume two sets Z_1 and Z_2 and let $z_1 \in Z_1$ and $z_2 \in Z_2$. We will sometimes identify a function $\pi : Z_1 \to Z_2$ with its relation (graph) $\pi \subseteq Z_1 \times Z_2$. For example if $\pi(z_1) = z_2$ we may equivalently write $(z_1, z_2) \in \pi$. Given a relation π we refer to π's *domain* as $\mathsf{dom}(\pi)$, to its *range* as $\mathsf{ran}(\pi)$, and to its *inverse* as π^{-1}.

2.1 Integer Linear Programming

Let $m, n \in \mathbb{N}$. We specify by $\mathbf{A} \in \mathbb{R}^{m \times n}$ an m by n *matrix* $\mathbf{A}$ of real numbers. Furthermore, $\mathbf{b} \in \mathbb{R}^m$ is a *(column) vector* composed of m real numbers. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$ and $\mathbf{x} \in \mathbb{Z}^n$. For $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$, we refer to $\mathbf{A}$'s ith row vector as $\mathbf{a}_i$ and as a_{ij} is the jth element in $\mathbf{a}_i$, *i.e.* the element in $\mathbf{A}$'s ith row and jth column. Correspondingly, b_i is $\mathbf{b}$'s ith element. Moreover, $\mathbf{Ax}$ denotes the standard matrix-vector multiplication resulting in a vector $\mathbf{d} \in \mathbb{R}^m$ and $\mathbf{c}'\mathbf{x}$ denotes the standard vector multiplication $\sum_{j=1}^{n} c_j x_j$ where $\mathbf{c}'$ is $\mathbf{c}$'s *transposed*. For $\mathbf{b}, \mathbf{d} \in \mathbb{R}^m$, we write $\mathbf{d} \leq \mathbf{b}$ if for all $i \in \{1, \ldots, m\}$, $d_i \leq b_i$. This linear algebra review covers all definitions required for the Integer Linear Programming Problem.

Definition 1 (Integer Linear Programming Problem **ILP**)

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, *and* $\mathbf{c} \in \mathbb{R}^n$, *for* $m, n \in \mathbb{N}$.

Output: $\min_{\mathbf{x} \in \mathbb{Z}^n} \{\mathbf{c}'\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\}$ *or* NO *if the above set is empty.*

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$ be an **ILP**-instance, and let $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. We may refer to the output corresponding to the input $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ as $\mathbf{ILP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$. A variable x_j is called a *decision variable* and $\mathbf{c}'\mathbf{x}$ is called the *objective function*. Note that $\mathbf{Ax} \leq \mathbf{b}$ can be decomposed into m inequalities of the form $\mathbf{a}_i\mathbf{x} = \sum_{j=1}^{n} a_{ij} x_j \leq b_i$, each called a *constraint*. If $\mathbf{x}$ satisfies $\mathbf{Ax} \leq \mathbf{b}$, *i.e.* all m constraints, it is called a *feasible solution*. If there exists no feasible solution for a given **ILP**-instance, then the instance is *infeasible*. A feasible solution that minimizes the objective function with respect to all feasible solutions is an *optimal (feasible) solution*.

It is common practice to use shorthand notation for constraints. For example, the equality $\mathbf{a}_i\mathbf{x} = b_i$ is equivalent to the two constraints $\mathbf{a}_i\mathbf{x} \leq b_i$ and $-\mathbf{a}_i\mathbf{x} \leq -b_i$. If variables are not defined, they are implicitly assumed to be zero. For example, the constraint $a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 \leq b_i$ is equivalent to $\mathbf{a}_i\mathbf{x} \leq b_i$ where $a_{i4} = \ldots = a_{in} = 0$.

Integer linear programming is a specialization of linear programming in that decision variables assume only values from $\mathbb{Z}$ and not from $\mathbb{R}$. This is necessary for modeling situations where only a discrete set of states is possible. However, it has substantial algorithmic implications compared to standard linear programming that are outside the scope of this paper. We simply note that **ILP** is **NP**-complete [19].

2.2 Graph Coloring

We use the standard k-**COLORING** problem later in Section 3.4 and briefly define it here. A *graph* G is a tuple (V,E), for a set of *vertices* V and a set of *(undirected) edges* $E \subseteq V \times V$.

Definition 2 (k-**COLORING** Problem)

Input: *A graph* $G = (V,E)$ *and a* $k \in \mathbb{N}$.

Output: YES *if there exists a function* $\mathsf{col} : V \to \{1,\ldots,k\}$ *such that for all* $(v_1,v_2) \in E$, $\mathsf{col}(v_1) \neq \mathsf{col}(v_2)$, *or* NO *otherwise.*

If an algorithm for this problem returns YES for a graph G and an integer k, then the respective function col is called a *k-coloring* of G. The k-**COLORING** problem is **NP**-complete [7].

3 Authorization-constrained Workflows

Our workflow terminology and formalization is based on [5] but adapted to suit our transformation to an optimization problem.

A *task* is a basic unit of work and may be executed multiple times. A task execution is performed by a user and we call it a *task instance*. A *workflow* models the causal and temporal dependencies between a set of tasks, whose execution constitutes a business objective. We call the execution of a workflow a *workflow instance*.

At *design time*, a business expert designs a workflow using a modeling language such as the Business Process Modeling Notation (BPMN) [17] (see Figure 1 for an example). He may additionally specify history-dependent authorizations, such as SoD and BoD constraints, which are workflow-specific. Orthogonal to this, a security expert defines a history-independent access control configuration. At *run time*, the workflow specification is deployed to a *workflow engine*, which schedules and instantiates tasks according to the workflow's control-flow. For each task instance, the workflow engine determines the set of users who are authorized to execute it with respect to both the history-dependent and the history-independent authorizations. We assume that history-dependent authorizations do not change at run time, whereas history-independent authorizations may change as motivated in the introduction.

In this paper, we overapproximate a workflow's control-flow and assume that a workflow engine may eventually instantiate every task. This approximation imposes no constraints on the workflow design and is compatible with all standard workflow patterns [21]. We further comment on this design decision in Section 3.4.

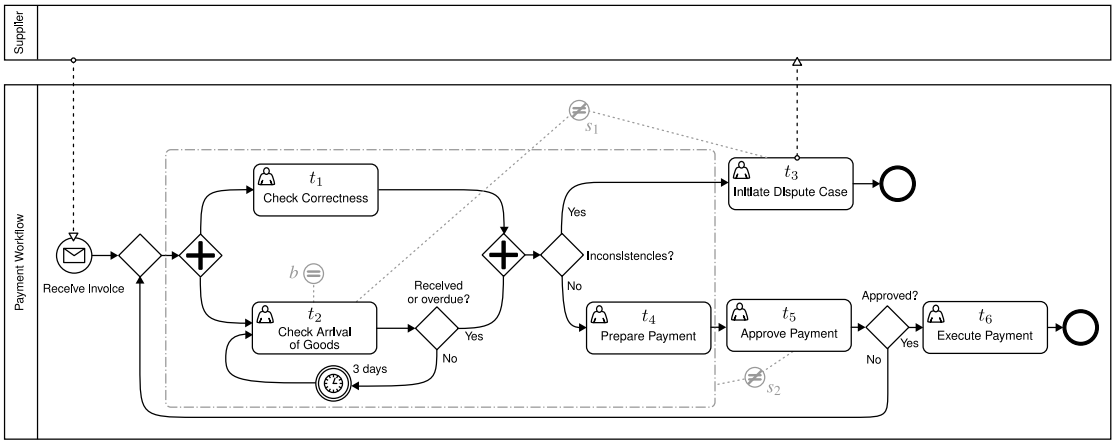

Figure 1: Payment workflow modeled in BPMN

3.1 Workflows

For the remainder of this paper, let $\mathscr{T}$ be a set of tasks and $\mathscr{U}$ a set of users. We model a workflow as a set of tasks $T \subseteq \mathscr{T}$, called a *workflow task set*. Furthermore, we model the execution of a task t by a user u, *i.e.* a task instance involving t and u, as a tuple (t,u) and call it an *execution event*. Let $\mathscr{X} = \mathscr{T} \times \mathscr{U}$ be the set of all execution events. Let T be the workflow task set modeling a workflow W. We model an instance of W as a set of execution events $H \subseteq T \times \mathscr{U}$, called a *workflow (execution) history*. Note that a workflow history does not store how many times a user u has executed a task t but only whether u has executed t. However, this is sufficient to decide whether the constraints that we introduce below are satisfied.

Example 1 As a running example, consider the BPMN [17] model of a payment workflow that is shown in Figure 1. This workflow is based on a report by an EU expert group on e-invoicing [11]. Ignoring the gray modeling elements for the moment, the workflow describes the tasks that a customer (organization) executes to process an invoice received by a supplier. Upon receipt of an invoice, a user checks whether the invoice is correct (t_1). In parallel, a user checks whether the goods corresponding to the invoice have arrived (t_2). If they have not arrived yet and their arrival is not overdue, the user waits for three days and checks again. Otherwise, the workflow proceeds. If inconsistencies have occurred, *i.e.* if the invoice is incorrect or the arrival is overdue, a user sends a dispute case (t_3) to the supplier and the workflow terminates. If no inconsistencies have occurred, a user prepares the payment (t_4). Afterwards, the payment is either approved (t_5), issued (t_6) and the workflow terminates, or the payment is not approved (t_5) and the workflow loops back to the start.

The payment workflow corresponds to the workflow task set $\{t_1,\ldots,t_6\}$ and we assume the set of users $\mathscr{U} = \{\mathsf{Alice},\ \mathsf{Bob}, \mathsf{Claire}, \mathsf{Dave}, \mathsf{Emma}, \mathsf{Fritz}\}$. Consider the workflow histories $H_1 = \{(t_1,\mathsf{Alice}),(t_2,\mathsf{Bob}),(t_2,\mathsf{Dave}),(t_4,\mathsf{Claire}),(t_5,\mathsf{Claire})\}$ and $H_2 = \{(t_1,\mathsf{Alice}),(t_2,\mathsf{Bob}),(t_4,\mathsf{Dave}),(t_5,\mathsf{Claire})\}$. The workflow history H_1 says that Alice executed t_1, Bob executed t_2, *etc.* We return to H_1 and H_2 below. ◇

3.2 History-dependent Authorizations

We consider two kinds of history-dependent authorizations: SoD and BoD constraints. Both of them are commonplace in regulated environments, such as the financial industry, and also recommended by best-practice frameworks, *e.g.* [13], that provide organizations guidance in complying with regulatory requirements.

Definition 3 *An* SoD constraint s *is a tuple* (T_1, T_2)*, for two disjoint sets of tasks* T_1 *and* T_2*. A workflow history* H *satisfies* s*, written* $H \models s$*, if* $\neg\exists u \in \mathscr{U}, t_1 \in T_1, t_2 \in T_2 . \{(t_1,u),(t_2,u)\} \subseteq H$.

In other words, H satisfies s if there is no user in H who executes tasks from both T_1 and T_2. Thereby, s separates the duties associated with the tasks in T_1 from those in T_2.

Definition 4 *A* BoD constraint b *is a set of tasks* T*. A workflow history* H *satisfies* b*, written* $H \models b$*, if* $|\{u \mid \exists t \in T . (t,u) \in H\}| \leq 1$.

Informally, H satisfies b if there is not more than one user in H who executes the tasks in T. Thereby, b binds the duties associated with the tasks in T. Note that according to Definition 4, H satisfies b even if H contains no instance of a task in T. We aggregate SoD and BoD constraints in an authorization policy, which we assume to be static, *i.e.* not changing at run time.

Definition 5 *A* (history-dependent) authorization policy ϕ *is a tuple* (S,B)*, for a set of SoD constraints* S *and a set of BoD constraints* B*. A workflow history* H *satisfies* ϕ*, written* $H \models \phi$*, if* H *satisfies every* $s \in S$ *and every* $b \in B$.

Example 2 We return to our running example. Consider again Figure 1, in particular the gray modeling elements. We visualize an SoD constraint (T_1, T_2) by identifying T_1 and T_2 with two dash-dotted boxes[1] and linking them with a dotted line and a node labeled with the symbol "$\neq$". Similarly, we visualize a BoD constraint b by identifying the respective set of tasks T with a dash-dotted box, linked to a node labeled with the "$=$" symbol. If a set contains only one task, we omit the dash-dotted box and link the task directly to the respective node.

Figure 1 shows the SoD constraints $s_1 = (\{t_2\}, \{t_3\})$ and $s_2 = (\{t_1, t_2, t_4\}, \{t_5\})$ and the BoD constraint $b = \{t_2\}$. Our example authorization policy is thus $\phi = (\{s_1, s_2\}, \{b\})$. The SoD constraint s_1 ensures that a user cannot embezzle the received goods and later initiate a dispute case. Similarly, the constraint s_2 ensures that any user, who approves a payment, did not execute one of the preceding tasks. Consequently, the approval of a fraudulent payment requires the collusion of at least two users. The BoD constraint b requires that only one user checks whether the goods have arrived. This facilitates a reuse of knowledge and thereby increases efficiency if multiple checks are required.

Consider again the workflow histories H_1 and H_2 from Example 1. The history H_1 does not satisfy ϕ because the execution events (t_2, Bob) and (t_2, Dave) violate b (t_2 is executed by two different users) and the events (t_4, Claire) and (t_5, Claire) violate s_2 (t_4 and t_5 are executed by the same user). However, H_2 satisfies ϕ because it satisfies s_1, s_2, and b. ◇

3.3 History-independent Authorizations

In the interest of keeping the forthcoming definitions agnostic with respect to different access control models, we first describe workflow-independent authorizations abstractly by a relation

[1] A dash-dotted box is called a *group artifact* in BPMN [17].

$UT \subseteq \mathscr{U} \times \mathscr{T}$, called a *user-task assignment*. Then, we refine UT using roles and use this additional structure when modeling the cost of changing UT. For the remainder of this paper, let $\mathscr{R}$ be a set of *roles*. We use the core idea of *Role-based Access Control (RBAC)* [10], namely the decomposition of UT into two relations.

Definition 6 *An* RBAC configuration *is a tuple* (UR, RT)*, where* $UR \subseteq \mathscr{U} \times \mathscr{R}$ *is a* user-role assignment *and* $RT \subseteq \mathscr{R} \times \mathscr{T}$ *a* role-task assignment .

Given an RBAC configuration (UR, RT), we can derive a user-task assignment UT by composing RT and UR with the composition operator "$\circ$". Formally, $UT = RT \circ UR = \{(u,t) \mid \exists r \in \mathscr{R}.(u,r) \in UR, (r,t) \in RT\}$.

We use UT not only to define the workflow-independent assignment of users to tasks. Its domain $\mathsf{dom}(UT)$ also represents the set of *available* users and, conversely, $\mathscr{U} \setminus \mathsf{dom}(UT)$ is the set of *unavailable* users, *e.g.* those users who are not ready to work or not part of the organization. We leave it up to an implementation to give these terms a concrete meaning.

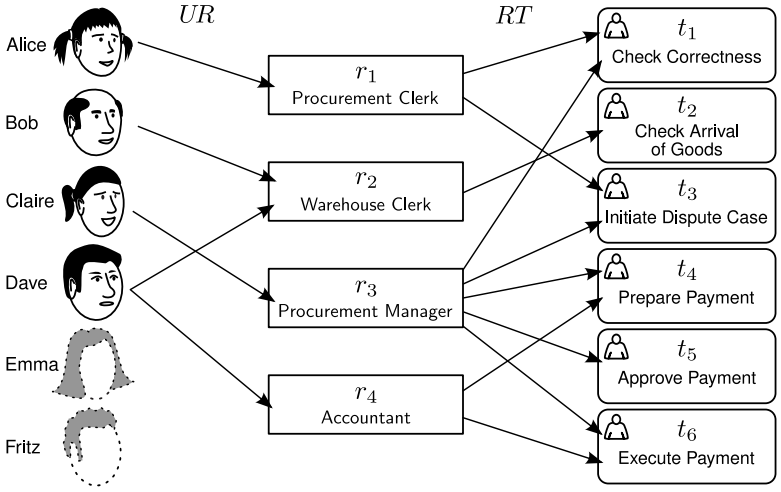

Figure 2: Role-based Access Control configuration

Example 3 Figure 2 shows an RBAC configuration (UR, RT) for the payment workflow. We refer to the role Procurement Clerk as r_1, Warehouse Clerk as r_2, Procurement Manager as r_3, and Accountant as r_4. The set of roles is thus $\mathscr{R} = \{r_1, r_2, r_3, r_4\}$. The user-task assignment $UT = RT \circ UR$ contains, for example, the tuple (Alice, t_1). The set of available users is $\mathsf{dom}(UT) = \{\mathsf{Alice}, \mathsf{Bob}, \mathsf{Claire}, \mathsf{Dave}\}$, whereas Emma and Fritz are unavailable. ◇

3.4 Allocation

Having introduced workflows and authorization constraints, we now formalize the existence of an allocation of users to tasks that satisfies a given set of constraints and every possible workflow execution.

Definition 7 *Let T be a workflow task set, H a workflow history, ϕ an authorization policy, and UT a user-task assignment. An* allocation *for T, H, ϕ, and UT is a (total) function* $\mathsf{alloc} : T \to \mathscr{U}$ *that satisfies:*

(1) $\mathsf{alloc}^{-1} \subseteq UT$ *and*

(2) $H \cup \mathsf{alloc} \models \phi$.

We write $\mathsf{alloc} \models (T,H,\phi,UT)$ if alloc is an allocation for T, H, ϕ, and UT and $\models^{\exists} (T,H,\phi,UT)$ if there exists an allocation alloc such that $\mathsf{alloc} \models (T,H,\phi,UT)$.

A workflow history is a record of past task instances and the users who executed them. An allocation defines for every future task instance the user who will be assigned to execute the respective task. Condition (1) requires that a user u is only allocated to a task t if u is authorized to execute t with respect to UT. Condition (2) requires that future task executions satisfy the history-dependent authorizations in ϕ, also accounting for past task instances. A consequence of Condition (2) is that there exists no allocation for T, H, ϕ, and UT if $H \not\models \phi$. This is consistent with our notion that it is not possible to find an extension of a workflow history H that satisfies an authorization policy ϕ, if H does not satisfy ϕ.

The two conditions illustrate the fundamental difference between history-dependent and history-independent authorizations. Deciding whether a task execution is authorized with respect to a history-dependent authorization depends on past task instances. In contrast, deciding whether a task execution is authorized with respect to a history-independent authorization can be decided without knowing the workflow, in particular its execution history. Hence, the two names.

An allocation instructs a workflow engine which users to assign to newly instantiated tasks. Condition (2) ensures that no matter which tasks are instantiated in the future, there is always a user who is authorized to execute them. Thereby, the existence of an allocation guarantees that the workflow engine can execute the respective workflow instance to completion.

Example 4 Consider again our example with the workflow task set T and the workflow history H_2 from Example 1, the authorization policy ϕ from Example 2, and the user-task assignment UT from Example 3. The function $\mathsf{alloc} = \{(t_1,\mathsf{Alice}),(t_2,\mathsf{Bob}),(t_3,\mathsf{Alice}),(t_4,\mathsf{Dave}),(t_5,\mathsf{Claire}),(t_6,\mathsf{Dave})\}$ is an allocation for T, ϕ, UT, and H_2. ◇

This example also illustrates that our overapproximation of a workflow's control-flow is reasonable, in particular when the workflow contains loops. Even though almost all tasks of the payment workflow have been executed in the workflow instance corresponding to H_2, a workflow engine may eventually schedule an instance of every task if the payment is not approved.

We now cast the existence of an allocation as a decision problem and analyze its complexity.

Definition 8 (Allocation Existence Problem **AEP**)

Input: *A workflow task set T, a workflow history H, an authorization policy ϕ, and a user-task assignment UT.*

Output: YES *if* $\models^{\exists} (T,H,\phi,UT)$ *or* NO *otherwise.*

Lemma 1 **AEP** *is* **NP**-*complete.*

Proof. Assume an instance of the **NP**-complete k-**COLORING** problem, introduced in Section 2.2, consisting of a graph (V,E) and an integer k. In the following, we present a polynomial

reduction to **AEP**. Let $T = V$, $H = \varnothing$, $\mathscr{U} = \{1,\ldots,k\}$, and $UT = \mathscr{U} \times T$. For every $(v_1, v_2) \in E$ add an SoD constraint $(\{v_1\},\{v_2\})$ to the set of SoD constraints S and let $\phi = (S, \varnothing)$.

Suppose an algorithm for **AEP** finds an allocation alloc such that $\mathsf{alloc} \models (T, H, \phi, UT)$. We show that alloc is a k-coloring for (V,E). By our construction and Definition 7, $\mathsf{alloc} : V \to \{1,\ldots,k\}$, *i.e.* alloc has the domain and range of a k-coloring for (V,E). Let $H' = H \cup \{(v,n) \mid \mathsf{alloc}(v) = n\}$. Consider an edge $(v_1,v_2) \in E$ and let s be the corresponding SoD constraint $(\{v_1\},\{v_2\})$ in S. By condition (2) of Definition 7, $H' \models \phi$ and therefore $H' \models s$. It follows by Definition 3 that $\{u \mid \exists v \in \{v_1\}.(v,u) \in H'\} \cap \{u \mid \exists v \in \{v_2\}.(v,u) \in H'\} = \varnothing$. Because $(v_1, \mathsf{alloc}(v_1)) \in H'$ and $(v_2, \mathsf{alloc}(v_2)) \in H'$ by the definition of H' it follows that $\mathsf{alloc}(v_1) \neq \mathsf{alloc}(v_2)$. Hence, alloc is a k-coloring for (V,E).

Let $\mathsf{col} : V \to \{1,\ldots,k\}$ be a k-coloring for (V,E). Because $UT = \{1,\ldots,k\} \times V$, col satisfies Condition (1) of Definition 7. By our construction, $\mathsf{col} \models s$ for every $s \in S$. Because $B = \varnothing$ and $H = \varnothing$ it therefore follows that $H \cup \mathsf{col} \models \phi$ by Definition 5, *i.e.* col satisfies Condition (2) of Definition 7. Hence, col is an allocation for (T,H,ϕ,UT) and **AEP** is **NP**-hard.

Given an instance (T,H,ϕ,UT) of **AEP** and a function $\mathsf{alloc} : T \to \mathscr{U}$, one can check in polynomial time whether $\mathsf{alloc} \models (T,H,\phi,UT)$ by verifying that alloc satisfies the two conditions of Definition 7. Hence, **AEP** is in **NP** and thereby **NP**-complete. ■

We do not provide an algorithm for **AEP** here. Instead, we show in Section 4.2 how to use algorithms for problems that build on **AEP** to solve instances of **AEP**.

4 Optimal Administrative Changes

Our formal model for authorization-constrained workflows, in particular the existence of an allocation, gives us a notion of empowerment, required for achieving a particular business objective. We now investigate the counterpart of empowerment, namely protection and the question of how to balance the two. Consider the following motivational example.

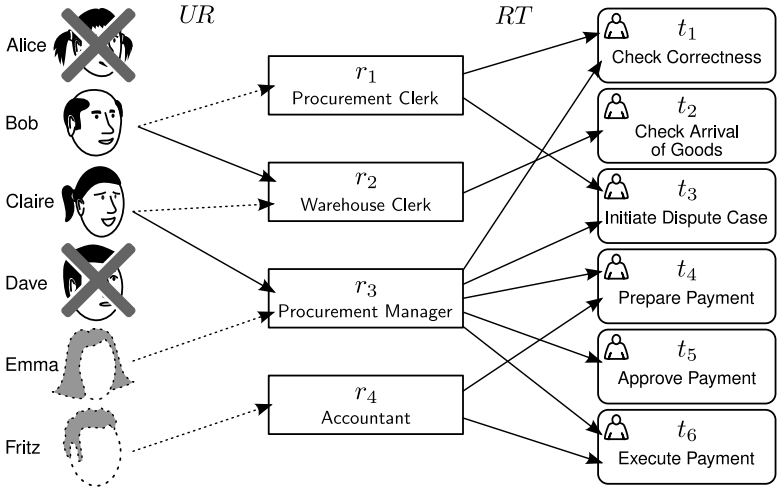

Figure 3: Changed RBAC configuration

Example 5 Let UR_0 be the user-role assignment and RT the role-task assignment illustrated in Figure 2. Furthermore, let $UT_0 = RT \circ UR_0$. We concluded in Example 4 that there exists

an allocation for the workflow task set T, the workflow history H_2, the authorization policy ϕ, and UT_0. Suppose now that Alice and Dave become unavailable, say they went on holiday. The new RBAC configuration (UR, RT) is illustrated in Figure 3, ignoring the dotted arrows for the moment. Note that RT did not change whereas $UR = UR_0 \setminus \{(\mathsf{Alice}, r_1), (\mathsf{Dave}, r_2), (\mathsf{Dave}, r_4)\}$. As a result, we get the new user-task assignment $UT = RT \circ UR$.

It is easy to see that there exists no allocation for T, H_2, ϕ, and UT. Only Claire is authorized to execute t_1 and t_4 with respect to UT. However, the SoD constraint s_2 in ϕ does not authorize Claire to execute t_1 and t_4 because according to H_2 she executed t_5 already. ◇

To overcome the situation illustrated in this example, we must change UT by assigning more roles to available users or making previously unavailable users available. However, this change should incur minimal cost.

In this section, we introduce a cost function that models the administrative cost of changing UT to UT' and the associated risks. We use this function to evaluate potential new user-task assignments and ultimately to find the optimal assignment UT' such that $\models^{\exists} (T, H_2, \phi, UT')$.

4.1 The General Problem

In the interest of keeping the general definition of the problem of balancing empowerment and protection agnostic with respect to access control models, we start with a generic definition of the cost function.

Definition 9 *A* cost function *is a partial function* $\mathsf{cost} : 2^{\mathscr{U}\times\mathscr{T}} \times 2^{\mathscr{U}\times\mathscr{T}} \to C$*, for a totally ordered set* C*.*

We use a cost function for two purposes. For two user-task assignments UT and UT'

1. $\mathsf{cost}(UT, UT')$ defines the cost of changing UT to UT' and
2. $\mathsf{dom}(\mathsf{cost})$ defines the feasible changes, *i.e.* it is possible to change from UT to UT' if $(UT, UT') \in \mathsf{dom}(\mathsf{cost})$.

In this general setting, the cost of changing from a user-task assignment UT to a user-task assignment UT' can have many meanings and cost may satisfy different properties accordingly. We give a few examples of potential costs that may be modeled using cost. A concrete example for a role-based cost function follows in the next section.

Risk: By empowering users to execute tasks, a user-task assignment exposes the underlying resources to risks, such as fraud, errors, and data leakage. There exist various methodologies for performing a risk analysis [4,14]. We consider them outside the scope of this paper and simply point out that the expected value in a quantitative risk analysis corresponds to a cost [14]. If the cost function encodes only risks, the return value of $\mathsf{cost}(UT, UT')$ is independent of UT. Additionally, if the risk quantifies only the misuse of authorizations, it is reasonable to assume that $\mathsf{cost}(UT, \varnothing) \leq \mathsf{cost}(UT, UT')$ for all user-task assignments UT and UT'. In other words, empowering no user to execute a task entails the least risk. We consider minimizing risk equivalent to maximizing protection.

Administrative cost: The activities associated with changing an access control configuration are typically not for free. For example, recruiting a new employee, assigning her initial authorizations, and training her to use them appropriately may be costly [15]. Consequently, if cost

encodes only administrative costs it is reasonable to assume that $\mathsf{cost}(UT, UT) \leq \mathsf{cost}(UT, UT')$ for all user-task assignments UT and UT'. In other words, it costs the least to make no changes at all.

Maintenance cost: Maintaining an access control configuration may involve costs such as salaries and license fees required for the execution of tasks. Abstractly, a cost function only encoding maintenance costs behaves the same way as a cost function only encoding risk: it is cheapest to maintain an empty user-task assignment.

Using the existence of an allocation as empowerment condition and a cost function as a measure of protection, we now reduce the question of how to balance empowerment and protection to an optimization problem.

Definition 10 (Optimal Workflow-aware Authorization Administration Problem **OWA**)

Input: *A cost function* cost*, a workflow task set T, a workflow history H, an authorization policy ϕ, and a user-task assignment UT.*

Output: $\min_{(UT,UT') \in \mathsf{dom}(\mathsf{cost})} \{\mathsf{cost}(UT, UT') \mid \models^{\exists} (T, H, \phi, UT')\}$ *or* NO *if the above set is empty.*

The Optimal Workflow-aware Authorization Administration Problem **OWA** asks for a user-task assignment that enables the successful completion of the given workflow instance and incurs minimal cost.

Note that instead of using the domain of the cost function as a predicate for feasible access control configurations, we could alternatively require cost to be a total function and define the cost of infeasible configurations to be infinite. However, this would lead to two case distinctions in **OWA**: one for the case that there exists no feasible configuration and one for the case that there exists no allocation.

Without any assumptions about the structure of the cost function it is impossible to make statements about **OWA**'s runtime or space complexity.

4.2 A Role-based Cost Function

To demonstrate the applicability of **OWA** to a realistic example, we refine **OWA** by decomposing user-task assignments into RBAC configurations and assume the cost function to be role-based. For simplicity, we also assume that the totally ordered set C is $\mathbb{R}$. More specifically, we define the cost function in terms of the following auxiliary functions. For a role $r \in \mathscr{R}$:

- $\mathsf{risk}(r) \in \mathbb{R}$ models the risk associated with the assignment of a user to r,
- $\mathsf{add}(r) \in \mathbb{R}$ models the administrative cost of assigning a user to r,
- $\mathsf{rm}(r) \in \mathbb{R}$ models the administrative cost of removing a user's assignment from r, and
- $\mathsf{ma}(r) \in \mathbb{R}$ models the maintenance cost of having a user assigned to r.

Using these functions, we define the cost of changing a user-role assignment.

Definition 11 *Given the auxiliary functions* $\mathsf{risk}, \mathsf{add}, \mathsf{rm}, \mathsf{ma} : \mathscr{R} \to \mathbb{R}$*, a* role cost function *is a function* $\mathsf{costR} : 2^{\mathscr{U} \times \mathscr{R}} \times 2^{\mathscr{U} \times \mathscr{R}} \to \mathbb{R}$*, such that for two user-role assignments UR and UR',*

$$\mathsf{costR}(UR, UR') = \sum_{(u,r) \in UR'} (\mathsf{risk}(r) + \mathsf{ma}(r)) + \sum_{(u,r) \in UR' \setminus UR} \mathsf{add}(r) + \sum_{(u,r) \in UR \setminus UR'} \mathsf{rm}(r)$$

A role cost function defines the cost of changing from UR to UR' simply as the sum of all the risk and maintenance costs associated with UR' and the administrative cost of adding and removing assignments when changing from UR to UR'. We assume that the auxiliary functions risk, add, rm, and ma are total and hence costR is total too. Instead of using costR's domain to determine feasible user-role assignment changes, we define a *maximal user-role assignment* $UR^{\max} \subseteq \mathscr{U} \times \mathscr{R}$ and assume that every user-role assignment $UR \subseteq UR^{\max}$ is feasible.

Example 6 Table 1 lists the risk, maintenance, and administrative costs associated with the four roles of the payment workflow. We adopt the elementary approach that roles assigned to a large number of tasks represent more responsibility and are therefore more costly [12]. Let costR be the corresponding role cost function.

	risk	ma	add	rm
Procurement Clerk (r_1)	5	3	2	1
Warehouse Clerk (r_2)	3	3	2	1
Procurement Manager (r_3)	12	5	3	2
Accountant (r_4)	7	4	2	1

Table 1: Decomposition of role cost function

Recall the RBAC configuration (UR, RT) shown in Figure 3 and let the solid and dotted arrows between users and roles in Figure 3 be the maximal user-role assignment $UR^{\max}$ for the payment workflow. For example, Emma is an unavailable user with respect to UT. Because $(\mathsf{Emma}, r_3) \in UR^{\max}$, we may change Emma's availability by assigning her to r_3, resulting in the user-role assignment $UR' = UR \cup \{(\mathsf{Emma}, r_3)\}$. The administrative activity of assigning Emma to r_3 costs 3 and the overall risk and maintenance cost rises by $12+5$. Thus, $\mathsf{costR}(UR, UR') - \mathsf{costR}(UR, UR) = 3+12+5 = 20$. ◇

Note that a role cost function costR and a maximal user-role assignment $UR^{\max}$ induce a cost function as follows. Let UT and UT' be two user-task assignments where UT is composed from an RBAC configuration (UR, RT). The cost of changing from UT to UT' is then $\mathsf{cost}(UT, UT') = \min_{UR' \subseteq UR^{\max}} \{\mathsf{costR}(UR, UR') \mid UT' = RT \circ UR'\}$ and is undefined if the set is empty. Hence, we may use costR, $UR^{\max}$, and (UR, UT) instead of cost and UT in following refinement of **ROWA**.

Definition 12 (Role-based Optimal Workflow-aware Authorization Administration Problem **ROWA**)

Input: *A role cost function* costR*, a maximal user-role assignment* UR^{max}*, a workflow task set* T*, a workflow history* H*, an authorization policy* ϕ*, and an RBAC configuration* (UR, RT)*, such that* $H \models \phi$*.*

Output: $\min_{UR' \subseteq UR^{max}} \{\mathsf{costR}(UR, UR') \mid \models^{\exists} (T, H, \phi, RT \circ UR')\}$ *or* NO *if the above set is empty.*

We may refer to the output corresponding to the **ROWA**-instance *rowa* as **ROWA**$(rowa)$. In the following, we define a function ROWAtoILP that transforms a **ROWA**-instance to an **ILP**-instance. We specify the matrix $\mathbf{A}$ and the vectors $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{x}$ indirectly by defining the respective (**ILP**) constraints and the cost function in terms of sums. Furthermore, we index decision variables with a superscript; they are not to be confused with an exponent. We thereby simplify the forthcoming proofs. Transforming the constraints and variables to a matrix-vector form is straightforward and therefore not shown in detail.

Definition 13 *Let* $(\mathsf{costR}, UR^{max}, T, H, \phi, (UR, RT))$ *be a* **ROWA***-instance, let* costR *be composed of the auxiliary functions* risk, add, rm, *and* ma, *and let* $U = \mathsf{dom}(UR^{max})$ *and* $R = \mathsf{ran}(UR^{max})$. *The function* ROWAtoILP *transforms* $(\mathsf{costR}, UR^{max}, T, H, \phi, (UR, RT))$ *to an instance of* **ILP** *as follows:*

Decision variables:

$$\forall u \in U, r \in R, t \in T \,.\, x^{u,r}, x^{u,t} \in \mathbb{Z}$$

Objective function:

$$\sum_{(u,r) \in U \times R} x^{u,r}(\mathsf{risk}(r) + \mathsf{ma}(r)) + \sum_{(u,r) \in (U \times R) \setminus UR} x^{u,r}\mathsf{add}(r) + \sum_{(u,r) \in UR} (1 - x^{u,r})\mathsf{rm}(r)$$

Constraints:

$$(1)\ \forall t \in T,\, u \in U \,.\, \textstyle\sum_{\{r \mid (r,t) \in RT\}} x^{u,r} \geq x^{u,t}$$

$$(2)\ \forall t \in T. \textstyle\sum_{u \in U} x^{u,t} = 1$$

$$(3)\ \forall t \in T. \textstyle\sum_{\{u \in U \mid H \cup \{(u,t)\} \not\models \phi\}} x^{u,t} = 0$$

$$(4)\ \forall (T_1, T_2) \in S, t_1 \in T_1, t_2 \in T_2, u \in U \,.\, x^{u,t_1} + x^{u,t_2} \leq 1$$

$$(5)\ \forall T' \in B, t_1, t_2 \in T', u \in U \,.\, x^{u,t_1} - x^{u,t_2} = 0$$

$$(6)\ \textstyle\sum_{(u,r) \in (U \times R) \setminus UR^{max}} x^{u,r} = 0$$

$$(7)\ \forall u \in U,\, r \in R \,.\, x^{u,r} \geq 0 \text{ and } x^{u,r} \leq 1$$

$$(8)\ \forall u \in U,\, t \in T \,.\, x^{u,t} \geq 0 \text{ and } x^{u,t} \leq 1$$

Consider a **ROWA**-instance composed of costR, $UR^{\max}$, T, H, ϕ, and (UR, RT), and let $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ be the corresponding **ILP**-instance returned by ROWAtoILP. We refer to a constraint or a set of constraints i in Definition 13 as Ci.

Next, we define a relation between feasible solutions of **ILP**-instances generated by ROWAtoILP, and user-role assignments and allocations for their corresponding **ROWA**-instances. Afterwards, we use this relation to explain the constraints C1–C8 and finally to prove soundness and completeness of ROWAtoILP.

Note the following. A feasible solution $\mathbf{x}$ for $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ is composed of the decision variables $x^{u,r}$ and $x^{u,t}$ where u ranges over $\mathsf{dom}(UR^{\max})$, r over $\mathsf{ran}(UR^{\max})$, and t over T. Because $\mathbf{x}$ is a feasible solution, the decision variables satisfy all constraints listed in Definition 13, in particular C7 and C8. Therefore, the decision variables assume either the value 0 or 1.

Definition 14 *Let* $(\mathsf{costR}, UR^{max}, T, H, \phi, (UR, RT))$ *be a* **ROWA***-instance and* $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ *the corresponding* **ILP***-instance returned by* ROWAtoILP. *Furthermore, let* $\mathbf{x}$ *be a feasible solution for* $(\mathbf{A}, \mathbf{b}, \mathbf{c})$, $U = \mathsf{dom}(UR^{max})$, *and* $R = \mathsf{ran}(UR^{max})$. *For a user-role assignment* UR' *and an allocation* alloc, *we say that* $\mathbf{x}$ corresponds *to* (UR', alloc), *written* $\mathbf{x} \sim (UR', \mathsf{alloc})$, *if*

(1) $UR' = \{(u,r) \in U \times R \mid x^{u,r} = 1\}$ *and*

(2) $\mathsf{alloc} = \{(t,u) \in T \times U \mid x^{u,t} = 1\}$.

$$
\begin{aligned}
&\mathsf{costR}(UR, UR') \\
&\quad = \textstyle\sum_{(u,r)\in UR'}(\mathsf{risk}(r) + \mathsf{ma}(r)) + \sum_{(u,r)\in UR'\setminus UR}\mathsf{add}(r) + \sum_{(u,r)\in UR\setminus UR'}\mathsf{rm}(r) \\
&\quad = \textstyle\sum_{(u,r)\in UR'} 1\,(\mathsf{risk}(r) + \mathsf{ma}(r)) + \sum_{(u,r)\in (U\times R)\setminus UR'} 0\,(\mathsf{risk}(r) + \mathsf{ma}(r)) \\
&\qquad + \textstyle\sum_{(u,r)\in UR'\setminus UR} 1\,\mathsf{add}(r) + \sum_{(u,r)\in ((U\times R)\setminus UR')\setminus UR} 0\,\mathsf{add}(r) \\
&\qquad + \textstyle\sum_{(u,r)\in UR\setminus UR'} 1\,\mathsf{rm}(r) + \sum_{(u,r)\in UR\cap UR'} 0\,\mathsf{rm}(r) \\
&\quad = \textstyle\sum_{(u,r)\in U\times R} x^{u,r}(\mathsf{risk}(r)+\mathsf{ma}(r)) + \sum_{(u,r)\in (U\times R)\setminus UR} x^{u,r}\mathsf{add}(r) + \sum_{(u,r)\in UR}(1-x^{u,r})\mathsf{rm}(r) \\
&\quad = \mathbf{cx}
\end{aligned}
$$

Figure 4: Equality of role cost function and objective function

In other words, the decision variables of the form $x^{u,r}$ determine UR' and those of the form $x^{u,t}$ determine alloc. More specifically, if, for a user u and a role r, $x^{u,r} = 1$ then u is assigned to r in UR'. Moreover, for a user u and a task t, $x^{u,t} = 1$ implies that alloc maps t to u. Note that the correspondence relation $\sim$ uniquely determines a tuple (UR', alloc) given a vector $\mathbf{x}$ and *vice versa*.

We now give an informal description of the (**ILP**) constraints created by ROWAtoILP. A more thorough elaboration follows in the proof of Lemma 2. C1 makes sure that an allocation assigns a user u only to a task t if u is assigned to a role r that is assigned to t. C2 enforces that an allocation maps every task to exactly one user. C3 ensures that an allocation's assignments do not violate the given execution history. C4 and C5 enforce that an allocation satisfies the given SoD and BoD constraints, respectively. Finally, C6 restricts user-role assignments to subsets of the given maximal user-role assignment. The use of C7 and C8 was explained above already.

The following lemma, which we prove in Appendix A, establishes that ROWAtoILP is both sound and complete.

Lemma 2 *Let* $(\mathsf{costR}, UR^{max}, T, H, \phi, (UR, RT))$ *be a* **ROWA***-instance and* $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ *the corresponding* **ILP***-instance returned by* ROWAtoILP. *Let* $\mathbf{x}$ *be a vector,* UR' *a user-role assignment, and* alloc *an allocation, such that* $\mathbf{x} \sim (UR', \mathsf{alloc})$.

- Soundness: *If* $\mathbf{x}$ *is a feasible solution for* $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ *then* $UR' \subseteq UR^{max}$ *and* $\mathsf{alloc} \models (T, H, \phi, RT \circ UR')$.
- Completeness: *If* $UR' \subseteq UR^{max}$ *and* $\mathsf{alloc} \models (T, H, \phi, RT \circ UR')$ *then* $\mathbf{x}$ *is a feasible solution for* $(\mathbf{A}, \mathbf{b}, \mathbf{c})$.

Given the soundness and completeness of ROWAtoILP, we now show with Theorem 1 that ROWAtoILP and algorithms for **ILP** can be employed to solve **ROWA**-instances.

Theorem 1 *For every* **ROWA***-instance rowa,* $\mathbf{ROWA}(rowa) = \mathbf{ILP}(\mathsf{ROWAtoILP}(rowa))$.

Proof. Let $(\mathsf{costR}, UR^{\max}, T, H, \phi, (UR, RT))$ be a **ROWA**-instance and $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ the corresponding **ILP**-instance returned by ROWAtoILP. Let $U = \mathsf{dom}(UR^{\max})$, $R = \mathsf{ran}(UR^{\max})$, $\phi = (S, B)$, and let costR be defined by the auxiliary functions risk, add, rm, and ma. Furthermore, let

UR' be a user-role assignment, alloc an allocation, and $\mathbf{x}$ a vector such that $UR' \subseteq UR^{\max}$, $\mathsf{alloc} \models (T,H,\phi,RT \circ UR')$ and $\mathbf{x} \sim (UR',\mathsf{alloc})$. From Lemma 2 we have that $\mathbf{x}$ is a feasible solution for $(\mathbf{A},\mathbf{b},\mathbf{c})$.

As derived in Figure 4, it follows from Definitions 11, 13, and 14 that $\mathsf{costR}(UR,UR') = \mathbf{cx}$. Assume that UR' minimizes costR with respect to all user-role assignments $UR'' \subseteq UR^{\max}$ such that $\models^{\exists} (T,H,\phi,RT \circ UR'')$, *i.e.* $\mathsf{costR}(UR,UR') = \mathbf{ROWA}(\mathsf{costR},UR^{\max},T,H,\phi,(UR,RT))$. To derive a contradiction, assume that $\mathbf{ILP}(\mathbf{A},\mathbf{b},\mathbf{c}) \neq \mathsf{costR}(UR,UR')$. Because $\mathbf{x}$ is a feasible solution for $(\mathbf{A},\mathbf{b},\mathbf{c})$ and $\mathsf{costR}(UR,UR') = \mathbf{cx}$, there must exist a feasible solution $\mathbf{y}$ for $(\mathbf{A},\mathbf{b},\mathbf{c})$ such that $\mathsf{costR}(UR,UR') > \mathbf{cy}$. Let UR'' be a user-role assignment and alloc' an allocation such that $\mathbf{y} \sim (UR'',\mathsf{alloc}')$. It follows by Lemma 2 that $UR'' \subseteq UR^{\max}$ and $\mathsf{alloc}' \models (T,H,\phi,RT \circ UR'')$. As reasoned before, we have $\mathsf{costR}(UR,UR'') = \mathbf{cy}$ and therefore $\mathsf{costR}(UR,UR') > \mathsf{costR}(UR,UR'')$. However, this violates the minimality assumption we made about $\mathsf{costR}(UR,UR')$. Hence, $\mathbf{x}$ is an optimal solution for $(\mathbf{A},\mathbf{b},\mathbf{c})$ and the two outputs are equal. ■

We now establish the space and runtime complexity of ROWAtoILP. To this end, let again $(\mathsf{costR},UR^{\max},T,H,\phi,(UR,RT))$ be a **ROWA**-instance and $(\mathbf{A},\mathbf{b},\mathbf{c})$ the corresponding **ILP**-instance returned by ROWAtoILP. Furthermore, let $U = \mathsf{dom}(UR^{\max})$, $R = \mathsf{ran}(UR^{\max})$, $\phi = (S,B)$. The **ILP**-instance $(\mathbf{A},\mathbf{b},\mathbf{c})$ ranges over $|U||R| + |U||T|$ decision variables, which corresponds to the same number of columns of the matrix $\mathbf{A}$. There are $|T||U|$ constraints of kind (1), $|T|$ constraints of kind (2) and (3), $O(|S||T|^2|U|)$ constraints of kind (4), $O(|B||T|^2|U|)$ constraints of kind (5), there is one constraint of kind (6), $|U||R|$ constraints of kind (7), and $|U||T|$ constraints of kind (8). Thus, the total number of constraints is in $O(|U|(|T|^2(|S|+|B|)+|R|+|T|))$, corresponding to the same number of rows of $\mathbf{A}$. For the generation of constraints of kind (3) $H \cup \{(u,t)\} \not\models \phi$ must be computed for every task $t \in T$ and user $u \in U$. However, by Definitions 3, 4, and 5, this computation has a polynomial runtime complexity in the size of the **ROWA**-instance. Hence, ROWAtoILP is a polynomial reduction from **ROWA** to **ILP**.

Solving **ROWA** requires solving **AEP**, which is **NP**-complete by Lemma 1. Therefore, the following corollary is a direct consequence of Theorem 1 and the observation that ROWAtoILP is a polynomial reduction from **ROWA** to the **NP**-complete **ILP**.

Corollary 1 **ROWA** *is* **NP**-*complete.*

We have thereby shown that finding an optimal RBAC configuration that enables a successful completion of a given workflow instance is in the same complexity class as deciding whether the workflow instance can be successfully completed for a given RBAC configuration. Furthermore, the polynomial reduction from **ROWA** to **ILP** enables us to solve **ROWA**-instances using well-established algorithms for **ILP**. An example follows in the next section.

Note that ROWAtoILP and an algorithm for **ILP** can also be used to solve **AEP**. Let (T,H,ϕ,UT) be an **AEP**-instance. Using a set of roles R, we decompose UT into an RBAC configuration (UR,RT) such that $RT \circ UR = UT$. Furthermore, let $UR^{\max} = UR$, and costR be the role cost function composed of the auxiliary functions $\mathsf{risk}(r) = \mathsf{ma}(r) = 0$ and $\mathsf{add}(r) = \mathsf{rm}(r) = 1$, for all $r \in R$. $\mathbf{ROWA}(\mathsf{costR},UR^{\max},T,H,\phi,(UR,RT)) = 0$ if and only if $\models^{\exists} (T,H,\phi,UT)$. This follows from the observation that the minimal return value of costR is 0, which is only possible for $\mathsf{costR}(UR,UR) = 0$, implying that $\models^{\exists} (T,H,\phi,RT \circ UR)$.

4.3 Experimental Results

We return to our running example and demonstrate how off-the-shelf software can be used to solve **ROWA**-instances using our reduction to **ILP**. We implemented ROWAtoILP using the numerical computing software MATLAB [20].

Example 7 Recall the RBAC configuration (UR, RT) shown in Figure 3 and our observation in Example 5 that there exists no allocation for T, H_2, ϕ, and $UT = UR \circ RT$. Furthermore, recall the role cost function costR and the maximal user-role assignment $UR^{\max}$ presented in Example 6.

Using our ROWAtoILP-implementation, we transformed the **ROWA**-instance $(\mathsf{costR}, UR^{\max}, T, H, \phi, (UR, RT))$ to an **ILP**-instance $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ and computed an optimal solution $\mathbf{x}$. Following Definition 14, $\mathbf{x}$ corresponds to the user-role assignment $UR' = \{(\mathsf{Bob}, r_2), (\mathsf{Claire}, r_3), (\mathsf{Emma}, r_3)\}$ and the allocation $\mathsf{alloc} = \{(t_1, \mathsf{Emma}), (t_2, \mathsf{Bob}), (t_3, \mathsf{Claire}), (t_4, \mathsf{Emma}), (t_5, \mathsf{Claire}), (t_6, \mathsf{Claire})\}$. The cost of changing from UR to UR' is $\mathsf{costR}(UR, UR') = 43$. Hence the optimal administrative change with respect to costR that empowers the users to complete the payment workflow, without violating ϕ and respecting the execution history H_2, is to extend UR by assigning Emma to the role Procurement Manager (r_3).

Suppose now that the risk exposure changes in that the risk associated with an assignment to role r_3 increases by 3 to 15. The other numbers in Table 1 remain unchanged. By running our program again, we see that this small change of cost results in a different optimal solution. The optimal user-role assignment is now $UR'' = \{(\mathsf{Bob}, r_2), (\mathsf{Bob}, r_2), (\mathsf{Claire}, r_3), (\mathsf{Fritz}, r_4)\}$, the respective allocation is $\mathsf{alloc}' = \{(t_1, \mathsf{Bob}), (t_2, \mathsf{Bob}), (t_3, \mathsf{Claire}), (t_4, \mathsf{Fritz}), (t_5, \mathsf{Claire}), (t_6, \mathsf{Claire})\}$, and $\mathsf{costR}(UR, UR'') = 46$. Because the risk associated with r_3 increased, it is now cheaper, *i.e.* less risky, to assign Bob additionally to the role Procurement Clerk (r_1) and Fritz to Accountant (r_4) instead of assigning Emma to the role Procurement Manager (r_3). ◇

Computing optimal solutions for **ILP**-instances, such as the ones presented in the example above, takes about 100 milliseconds on a standard PC configuration[2]. We also experimented with larger, randomly generated maximal user-role assignments. On our test system, we could observe an exponential increase of the running time in the size of the input, which is consistent with our complexity approximation of ROWAtoILP and Corollary 1. However, we did not investigate optimizations of our prototypical implementation.

5 Related Work

When users are unavailable, delegation of roles and tasks may enable the continuation of a workflow's execution. Atluri *et al.* provide support for delegation in workflows ensuring that delegation operations, enabled to grant or to transfer rights to a particular user, still satisfy the authorization constraints [1]. In later work, users may perform concrete or abstract task delegations, and delegation may be conditioned on time, workload, and task attributes [2,22].

Crampton and Khambhammettu [9] were the first to check if permitting a delegation request prevents the completion of workflow instances or renders the workflow authorization schema unsatisfiable. In [23], Wang and Li show that the workflow satisfiability problem is in general **NP**-hard. They also introduced the workflow resiliency problem, asking whether a workflow

[2] Mac OS X on 2.5 GHz Intel Core 2 Duo processor with 2 GB RAM.

can be completed even if a number of users may be unavailable. They show that the complexity of deciding whether an authorization configuration is resilient for a workflow with up to t unavailable users at one time is **PSPACE**-complete. Our constraint model builds on the work of Basin *et al.* [5] and **AEP** is an adaption of their graph-based approximation of the enforcement process existence problem. However, none of the above works consider optimal authorization configurations. They simply provide algorithms to determine whether an authorization configuration satisfies a workflow and its authorization constraints, that is the question formalized by **AEP** in this paper.

The notion of risk has been introduced into access control models to adapt authorizations to changing conditions. Methods to measure and quantify risks are given in [8,14,16]. Aziz *et al.* use a risk semantics to transform policies with respect to operational, combinatorial, and conflict of interest risks with the goal to minimize the risk associated with a configuration [3]. In contrast to our work, they reassign permissions to roles leaving the user-role assignment, where changes occur in practice more frequently, untouched.

To quantify risk in role delegation, Han *et al.* consider the position of the role within the role hierarchy, the number of permissions gained, and also associate workflow instances with a risk based on the data that is processed [12]. For example, the value of a reimbursement workflow may depend on the amount of money involved. However, risk is not linked with successful workflow termination. Associating risk and benefit vectors with every read and update transactions, Zhang *et al.* study the optimization of an allowed transaction graph with respect to a given accessibility graph that defines the underlying communication system [24].

The work of Casassa Mont *et al.* [15] provides further metrics that are useful for defining role cost functions. In their economic interpretation of identity and access management, they identify potential cost drivers such as the approval of users' accounts and authorizations by managers and security teams. Moreover, they observe that privileged users, such as IT administrators with root access to sensitive systems, information, and shared accounts, expose higher risks.

6 Conclusion and Future Work

We have presented the concept of a cost-minimizing authorization configuration that empowers users to execute a given workflow. Our approach comes with considerable modeling freedom. For example, cost can model the risk associated with an authorization configuration and hence the optimal configuration maximizes protection. By first introducing the generic **OWA**-problem and later refining it to **ROWA**, we showed that our approach is both general and also applicable to concrete business scenarios. Furthermore, we presented a mapping from **ROWA** to the well-established optimization problem **ILP**. Proving our mapping sound and complete enabled us to use of off-the-shelf software to solve **ROWA**.

The generality of our approach gives rise to many design decisions and consequently to various directions for future work. For example, we based our authorization constraints on the model proposed in [5]. Other models provide different features, *e.g.* support for delegation [9]. Similarly, user-task assignments can be refined based on different access control models. In particular, our role cost function could be further refined by incorporating role hierarchies. Furthermore, the predicate whether an authorization change is feasible could account for additional properties such as time. With each such modeling change, the computational complexity of finding an optimal configuration must be reexamined.

Meaningful risk metrics for authorization configurations are a precondition for the effective use of our approach. We pointed to various methods for quantifying the risk associated with authorization configurations. However, finding such metrics is challenging. This does not, of course, reduce the importance of such metrics and we see our results as providing additional evidence for their usefulness.

Acknowledgments. This work was partially funded by the European Commission under the Seventh Framework Project "PoSecCo" (IST 257129).

References

[1] V. Atluri, E. Bertino, E. Ferrari, and P. Mazzoleni. Supporting delegation in secure workflow management systems. In *Proc. of the Annual Working Conference on Data and Application Security*, pp. 190–202, 2003.

[2] V. Atluri and J. Warner. Supporting conditional delegation in secure workflow management systems. In *Proc. of the ACM Symposium on Access Control Models and Technologies (SACMAT '05)*, pp. 49–58, 2005.

[3] B. Aziz, S. N. Foley, J. Herbert, and G. Swart. Reconfiguring role based access control policies using risk semantics. *J. High Speed Networks*, 15(3):261–273, 2006.

[4] D. Basin, P. Schaller, and M. Schläpfer. *Applied Information Security*. Springer, 2011.

[5] D. Basin, S. J. Burri, and G. Karjoth. Obstruction-free authorization enforcement: Aligning security with business objectives. In *Proc. of the IEEE Computer Security Foundations Symposium (CSF '11)*, pp. 99–113 2011.

[6] D. E. Bell and L. J. LaPadula. *Secure computer systems: Mathematical foundations*, MTR-2547, The Mitre Corporation, 1973.

[7] G. Chartrand and P. Zhang. *Chromatic Graph Theory*. Chapman & Hall, 2008.

[8] P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger. Fuzzy multilevel security: An experiment on quantified risk-adaptive access control. In *Proc. of the IEEE Symposium on Security and Privacy (S&P '07)*, pp. 222–230, 2007.

[9] J. Crampton and H. Khambhammettu. Delegation and satisfiability in workflow systems. In *Proc. of the ACM Symposium on Access Control Models and Technologies (SACMAT '08)*, pp. 31–40, 2008.

[10] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for Role-Based Access Control. *TISSEC*, 4(3):224–274, 2001.

[11] European Union. *Final report of the expert group on e-invoicing*. `http://bit.ly/yvWtfQ`, 2009.

[12] W. Han, Q. Ni, and H. Chen. Apply measurable risk to strengthen security of a role-based delegation supporting workflow system. In *Proc. of the IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY '09)*, pp. 45–52, 2009.

[13] IT Governance Institute. *Control objectives for information and related technology (COBIT) 4.1*, 2005.

[14] I. Molloy, P.-C. Cheng, and P. Rohatgi. Trading in risk: using markets to improve access control. In *Proc. of the Workshop on New Security Paradigms (NSPW '08)*, pp. 107–125, 2008.

[15] M. C. Mont, Y. Beresnevichiene, D. Pym, and S. Shiu. Economics of identity and access management: Providing decision support for investments. In *Network Operations and Mgnt. Symposium Workshops*, pp. 134 –141, 2010.

[16] Q. Ni, E. Bertino, and J. Lobo. Risk-based access control systems built on fuzzy inferences. In *Proc. of the ACM Symposium on Information, Computer and Communi- cations Security (ASIACCS '10)*, pp. 250–260, 2010.

[17] Object Management Group (OMG). Business Process Model and Notation (BPMN), v 2.0. 2011.

[18] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proc. of the IEEE*, pp. 1278–1308, 1975.

[19] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.

[20] The MathWorks. Matlab r2011b. `www.mathworks.com/products/matlab`, 2012.

[21] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[22] J. Wainer, A. Kumar, and P. Barthelmess. DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Inf. Syst.*, 32(3):365–384, 2007.

[23] Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *TISSEC*, 13(4):40, 2010.

[24] L. Zhang, A. Brodsky, and S. Jajodia. Toward information sharing: Benefit and risk access control (BARAC). In *Proc. of the IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '06)*, pp. 45–53, 2006.

A Proof of Lemma 2

Proof. Let $(\mathsf{costR}, UR^{\max}, T, H, \phi, (UR, RT))$ be a **ROWA**-instance and $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ the corresponding **ILP**-instance returned by ROWAtoILP. Furthermore, let $U = \mathsf{dom}(UR^{\max})$, $R = \mathsf{ran}(UR^{\max})$, and $\phi = (S, B)$.

Soundness: Let $\mathbf{x}$ be a feasible solution for $(\mathbf{A}, \mathbf{b}, \mathbf{c})$, UR' a user-role assignment, and alloc an allocation, such that $\mathbf{x} \sim (UR', \mathsf{alloc})$. We first show that $UR' \subseteq UR^{\max}$. To derive a contradiction, assume $(u, r) \in UR' \setminus UR^{\max}$, for a user u and a role r. By Definition 14, it follows that $x^{u,r} = 1$. However, this contradicts C6, which forces $x^{u,r}$ to be 0 (Remember, decision variables of a feasible solution only assume the values 0 and 1). Hence, $UR' \setminus UR^{\max} = \varnothing$ and therefore $UR' \subseteq UR^{\max}$.

By C2, alloc maps every task to exactly one user and is therefore a total function. To show that $\mathsf{alloc} \models (T, H, \phi, RT \circ UR')$, we first prove that Condition (1) of Definition 7 holds and afterwards that Condition (2) holds. Let $(t, u) \in \mathsf{alloc}$, for a task t and a user u. By Definition 14, $x^{u,t} = 1$. It follows by C1 that there exists an r such that $x^{u,r} = 1$ and $(r, t) \in RT$. By Definition 14, $(u, r) \in UR'$ and therefore $(u, t) \in RT \circ UR'$. Hence, Condition (1) of Definition 7 holds.

To show that Condition (2) of Definition 7 holds, we make use of the following observation. Given an SoD constraint s, it follows by Definition 3 that $(H \cup \mathsf{alloc}) \models s$ is equivalent to $\{x_1, x_2\} \models s$ for all $x_1, x_2 \in (H \cup \mathsf{alloc})$. Similarly, given a BoD constraint b, it follows by Definition 4 that $(H \cup \mathsf{alloc}) \models b$ is equivalent to $\{x_1, x_2\} \models b$ for all $x_1, x_2 \in (H \cup \mathsf{alloc})$.

In the following we show that for every two execution events $x_1, x_2 \in (H \cup \mathsf{alloc})$, every SoD constraint $s \in S$, and every BoD constraint $b \in B$, $\{x_1, x_2\} \models s$ and $\{x_1, x_2\} \models b$. Using the previous observation and Definition 5 we then have $(H \cup \mathsf{alloc}) \models \phi$.

Case $\{x_1, x_2\} \subseteq H$: By Definition 12, $H \models \phi$. By Definition 5 and our previous observation, it follows that, for every $s \in S$ and $b \in B$, $\{x_1, x_2\} \models s$ and $\{x_1, x_2\} \models b$.

Case $x_1 \in H$ and $x_2 \in \mathsf{alloc}$: To derive a contradiction, assume $\{x_1, x_2\} \not\models \phi$. Let $x_2 = (t_2, u_2)$. By Definitions 3, 4, and 5 every superset of $\{x_1, x_2\}$ does not satisfy ϕ, in particular $H \cup \{x_2\}$. From C3 it follows that $x^{u_2,t_2} = 0$. However, by Definition 14 this contradicts $(t_2, u_2) \in \mathsf{alloc}$. Hence, $\{x_1, x_2\} \models \phi$ and by Definition 5 $\{x_1, x_2\} \models s$ and $\{x_1, x_2\} \models b$, for every $s \in S$ and $b \in B$. The case where $x_1 \in \mathsf{alloc}$ and $x_2 \in H$ is analogous.

Case $\{x_1, x_2\} \subseteq \mathsf{alloc}$: Let $x_1 = (t_1, u_1)$ and $x_2 = (t_2, u_2)$. Let $s = (T_1, T_2) \in S$ be an SoD constraint. By Definition 3, $\{x_1, x_2\} \models s$ unless $t_1 \in T_1$, $t_2 \in T_2$, and $u_1 = u_2$ (or analogously $t_1 \in T_2$, $t_2 \in T_2$, and $u_1 = u_2$). To derive a contradiction, assume $t_1 \in T_1$, $t_2 \in T_2$, and $u_1 = u_2$. Because $\{x_1, x_2\} \subseteq \mathsf{alloc}$ it follows by Definition 14 that $x^{u_1,t_1} = x^{u_2,t_1} = 1$. However, because $u_1 = u_2$, this contradicts C4. Hence, $\{x_1, x_2\} \models s$. Let $b = T' \in B$ be a BoD constraint. By Definition 4, $\{x_1, x_2\} \models b$ unless $t_1, t_2 \in T'$ and $u_1 \neq u_2$. To derive a contradiction, assume $t_1, t_2 \in T'$ and $t_1 \neq t_2$. Because $\{x_1, x_2\} \subseteq \mathsf{alloc}$ it follows by Definition 14 that $x^{u_1,t_1} = x^{u_2,t_1} = 1$. By C2 it follows that $x^{u_2,t_1} = 0$. This, however, contradicts C5, which requires $x^{u_1,t_1} = x^{u_2,t_1}$. Hence, $\{x_1, x_2\} \models b$.

Completeness: Let UR' be a user-role assignment, alloc an allocation, and $\mathbf{x}$ a vector such that $UR' \subseteq UR^{\max}$, $\mathsf{alloc} \models (T, H, \phi, RT \circ UR')$, and $\mathbf{x} \sim (UR', \mathsf{alloc})$. In the following, we show that $\mathbf{x}$ is a feasible solution for $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ by showing that every constraint of $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ is satisfied.

C1: Let t be a task, u a user, and consider the constraint $\sum_{\{r \mid (r,t) \in RT\}} x^{u,r} \geq x^{u,t}$. If $\mathsf{alloc}(t) \neq u$, then $x^{u,t} = 0$ by Definition 14 and the constraint is trivially satisfied. If $\mathsf{alloc}(t) = u$, it follows by Condition (1) of Definition 7 that $(u, t) \in UT \circ UR'$. Therefore, there exists an $r \in R$ such that

$(u,r) \in UR'$ and $(r,t) \in RT$. By Definition 14, $x^{u,r} = 1$ and the constraint is therefore satisfied.

C2: Let t be a task. By Definition 7, alloc is a total function and therefore maps t to one user $u \in U$. By Definition 14 we have $x^{u,t} = 1$. Hence, $\sum_{u \in U} x^{u,t} = 1$.

C3: Let t be a task. To derive a contradiction, let u be a user and assume that $H \cup \{(u,t)\} \not\models \phi$ and $x^{u,t} = 1$. By Definition 14 it follows that $(t,u) \in \mathsf{alloc}$. Because $\mathsf{alloc} \models (T,H,\phi,RT \circ UR')$ then by Definition 7 $H \cup \mathsf{alloc} \models \phi$. However, this contradicts the previous assumption that $H \cup \{(u,t)\} \not\models \phi$. Hence, $\sum_{\{u \in U \mid H \cup \{(u,t)\} \not\models \phi} x^{u,t} = 0$.

C4: Let $s = (T_1,T_2) \in S$ be an SoD constraint, $t_1 \in T_1$, $t_2 \in T_2$, and $u \in U$. Consider the constraint $x^{u,t_1} + x^{u,t_2} \leq 1$. If $\mathsf{alloc}(t_1) \neq u$, then $x^{u,t_1} = 0$ by Definition 14 and the constraint is trivially satisfied. If $\mathsf{alloc}(t_1) = u$, then $x^{u,t_1} = 1$ by Definition 14. To derive a contradiction, assume that $x^{u,t_2} = 1$. It follows by Definition 14 that $(t_2,u) \in \mathsf{alloc}(t_2)$. Because $\mathsf{alloc} \models (T,H,\phi,RT \circ UR')$, $H \cup \mathsf{alloc} \models \phi$ by Definition 7. Furthermore, $\{(t_1,u),(t_2,u)\} \models s$ because of Definition 5 and our previous observation. However, this contradicts $\{(t_1,u),(t_2,u)\} \not\models s$, which follows by Definition 3. Hence $x^{u,t_2} = 0$ and the constraint is satisfied.

C5: Let $b = T' \in B$ be a BoD constraint, $t_1,t_2 \in T'$, and $u \in U$. Consider the constraint $x^{u,t_1} - x^{u,t_2} = 0$ and let $x^{u,t_1} = 1$. It follows from Definition 14 that $(t_1,u) \in \mathsf{alloc}$. To derive a contradiction, assume $x^{u,t_2} = 0$. By C2, there exists an $u_2 \in U$ such that $u \neq u_2$ and $x^{u_2,t_2} = 1$. By Definition 14, then $(t_2,u_2) \in \mathsf{alloc}$. Because $\mathsf{alloc} \models (T,H,\phi,RT \circ UR')$, $H \cup \mathsf{alloc} \models \phi$ by Definition 7. By Definition 5 and our previous observation we have $\{(t_1,u),(t_2,u_2)\} \models b$. However, this contradicts $\{(t_1,u),(t_2,u_2)\} \not\models s$, which follows by Definition 4 because $u \neq u_2$. Hence $x^{u,t_2} = 1$ and the constraint is satisfied. The case where $x^{u,t_1} = 0$ and we derive a contradiction for $x^{u,t_2} = 1$ is analogous.

C6: $\sum_{(u,r) \in (U \times R) \setminus UR^{\max}} x^{u,r} = 0$ follows directly from $UR' \subseteq UR^{\max}$ and Definition 14.

C7 and C8: The satisfaction of these constraints follows directly from Definition 14. ■