Satisfiability and Feasibility in a Relationship-Based Workflow Authorization Model

Arif Akram Khan and Philip W.L. Fong

Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
{arikhan,pwlfong}@ucalgary.ca

Abstract. A workflow authorization model is defined in the framework of Relationship-Based Access Control (ReBAC), in which the protection state is a social network. Armed with this model, we study a new decision problem called workflow feasibility. The goal is to ensure that the space of protection states contains at least one member in which the workflow specification can be executed to completion. We identify a sufficient condition under which feasibility can be decided by a refutation procedure that is both sound and complete. A formal specification language, based on a monotonic fragment of the Propositional Dynamic Logic (PDL), is proposed for specifying protection state spaces. The adoption of this language renders workflow feasibility NP-complete in the general case but polynomial-time decidable for an important family of workflows.

Keywords: Relationship-based access control, workflow authorization model, workflow satisfiability, workflow feasibility, graph homomorphism, refutation procedure, propositional dynamic logic, model checking.

1 Introduction

In a workflow authorization system [1, 2, 3, 4, 5, 6], permissions are encapsulated in tasks, such that users acquire permissions by executing a task in a workflow. This design achieves permission abstraction [7] in the context of business processes in enterprise-level systems. Access control policies are usually specified in the form of constraints over who can execute which tasks in the workflow. An instantiation of the workflow (i.e., an assignment of tasks to users) must honour the constraints in order to be considered valid.

Research on workflow authorization models focus on the issue of ***availability***: i.e., are the permissions encapsulated in a workflow ***available*** when needed? Early studies explored the problem of ***workflow satisfiability*** [3, 5]: i.e., can a workflow be instantiated such that all constraints are satisfied in the current protection state? In a recent study pursued by Wang and Li [8, 6], a higher degree of availability is analyzed. Specifically, they [8, 6] studied the problem of

S. Foresti, M. Yung, and F. Martinelli (Eds.): ESORICS 2012, LNCS 7459, pp. 109–126, 2012.

workflow resiliency[1]. Suppose an emergency makes some users unavailable, can the workflow still be completed? A (statically) k-resilient workflow remains satisfiable even after any k users are removed from the current state.

In this work, we study the problem of ***workflow feasibility*** where we ask whether the permissions can be made available in any "*reasonable*" protection state or not. Although the permissions may not be available in the current protection state (i.e., the workflow is not currently satisfiable), there should be at least one protection state in a known state space where the workflow is satisfiable. This notion can been seen as the dual of resiliency, as explained in Sect. 2.

We envision that the policy developer has *a priori* knowledge of what the protection state space looks like. When she is to author a new workflow (or revise an existing one), she wants to evaluate the workflow, to determine whether the specification may be overly restrictive with respect to her understanding of what protection states are known to be in the state space. If the workflow is not satisfiable in any of the known protection states (i.e., infeasible), then constraints in the workflow specification must be relaxed.

While previously proposed workflow authorization models are built on top of a Role-Based Access Control framework [11, 3, 12, 13, 4], recent authors did recognize that many constraints in a workflow specification are binary relations over executors of tasks [5, 6]. To facilitate the modelling of interpersonal relationships, we ground our workflow authorization model in the recently proposed framework of ***Relationship-Based Access Control (ReBAC)*** [14, 15, 16]. In ReBAC, authorization decisions are based on the interpersonal relationships among users, and the protection state is a social network of users. As we will be studying satisfiability in the midst of evolving interpersonal relationships, we find ReBAC to be a natural theoretical basis.

Our specific contributions are the following:

1. We formulate a ReBAC workflow authorization model (Sect. 3). With this model, we offer a novel characterization of the workflow satisfiability problem based on graph homomorphism [17] (Sect. 4). This new perspective forms the basis for the rest of our results.
2. We propose a new decision problem, workflow feasibility, for assessing the availability of permissions encapsulated in a workflow against a known space of protection states (Sect. 5).
3. We propose to decide workflow feasibility using a refutation procedure, and we identify the exact conditions under which refutation is both sound and complete (Sect. 6).
4. We propose to specify a ReBAC protection state space using a fragment of ***Propositional Dynamic Logic*** [18]. When state spaces are specified in this way, workflow feasibility is NP-complete in the general case, but polynomial-time decidable for an important class of workflow specifications in which constraints are conjunctive (Sect. 7).

[1] Resiliency was also studied in a context not related to workflow authorization systems [9, 10].

2 Related Works

Workflow Authorization Models. Workflow authorization has received a lot of attention from the security research community. Early works on workflow authorization focus on synchronizing authorization decisions with the progression of workflow [2, 1]. These works do not concentrate on workflow constraints [12, 19, 20, 21, 22]. Role Based Access Control (RBAC) is widely used to model such constraints [11, 3, 12, 13, 4]. Our work instead focuses on relationship-based constraints.

Consistency. Bertino *et al.* developed a sophisticated constraint specification language [11, 3], in which constraints are clauses in logic programming. A desired characteristic of a set of constraints is that there should be at least one way to complete the workflow without breaching any of them. Bertino *et al.* called this ***consistency***. They proposed a planning algorithm for assigning users and roles to tasks in such a way that the workflow constraints are satisfied (i.e., deciding consistency). The complexity of this algorithm is exponential. Tan *et al.* defined consistency in a stricter sense by requiring one complete plan for each authorized user and role [4]. Their algorithm for deciding consistency is again exponential.

Satisfiability. Although the concept was implicit in the work of Bertino *et al.* [11, 3], Crampton first coined the term ***workflow satisfiability***, and gave a precise, solution-independent definition for it [5]. Specifically, satisfiability refers to the existence of an instantiation of a workflow specification such that all the constraints are satisfied. Satisfiability is defined with respect to a fixed protection state (e.g., for an RBAC system, the protection state consists of a role hierarchy and a user-role assignment). Wang and Li showed that workflow satisfiability is NP-complete [8, 6], and NP-completeness remains even by considering only the simplest types of constraints (i.e., user-step authorization and user-inequality constraints). They further proved that the problem is fixed-parameter tractable, meaning that there is a decision procedure for satisfiability which is exponential only to the number of tasks (but not the size of the protection state). Assuming the number of tasks is bounded by a small value, the problem is tractable. Workflow satisfiability is the building block of the feasibility problem. We offer a novel characterization of satisfiability in terms of graph homomorphism, which is instrumental in establishing our results.

Resiliency. Wang and Li pointed out availability to be the essence behind the pursuit of workflow satisfiability, and took it into a higher degree [8, 6]. While workflow satisfiability examines availability in the current protection state, they extended the notion to examine availability in any bleak future state when some users might be absent. They call the new problem ***workflow resiliency***. In other words, resiliency deals with availability in a state space that includes all possible protection states reachable from the current one by removing users.

This paper is related to their work in three ways. First, we follow the lead of Wang and Li in considering availability in the midst of a changing protection

state. Second, the problem of workflow feasibility can be seen as the dual of the resiliency problem. While resiliency is concerned with availability in *every* state of a state space induced by *adversarial transitions*, feasibility is concerned with availability in *at least one* state of a state space induced by *normal transitions*. Third, we consider not only the change of personnel, but also the evolution of their interpersonal relationships.

Relationship Based Access Control. We base our workflow authorization model on Relationship-Based Access Control (ReBAC) [14, 15, 16], which uses interpersonal relationships among users as the basis of authorization decisions. These relationships induce a social network that is explicitly tracked by a ReBAC system. A ReBAC access control policy specifies how a resource requester shall be topologically related to the resource owner in the social network in order for access to be granted. Sect. 3 shows that our ReBAC workflow authorization model can capture all constraint types previously discussed in [4, 5, 8, 6].

3 A ReBAC Workflow Authorization Model

Notation. We write $dom(f)$ and $ran(f)$ respectively for the domain and the range of a function f. We write $f : A \rightharpoonup B$ whenever $dom(f) \subseteq A$ and $ran(f) \subseteq B$. If $X \subseteq dom(f)$ then we write $f(X)$ for $\{f(x) \mid x \in X\}$. We write 2^X for the power set of X (i.e., the set of all subsets of X).

Protection System. A ***ReBAC protection system*** (or simply a ***system***) is parameterized by two sets, $\mathcal{U}$ and $\mathcal{L}$. The set $\mathcal{U}$ is a *countably infinite* set of ***user identifiers*** of the system. At run time, only a finite subset of $\mathcal{U}$ is active (see below). We write u and v for typical members of $\mathcal{U}$. The set $\mathcal{L}$ is a *finite* set of ***relation identifiers***. Each member of $\mathcal{L}$ identifies a type of interpersonal relationship tracked by the system: e.g., "parent", "doctor", etc. It is assumed that these relations are binary. Note that a relation and its inverse may be named differently: e.g.,"patient" and "doctor". We write l for a typical member of $\mathcal{L}$. In the following, we fix the sets $\mathcal{U}$ and $\mathcal{L}$.

Protection State. The ***state*** of a ReBAC protection system is characterized by a ***social network***, which is essentially an edge-labelled directed graph. The vertices of a social network represent the *active* users of the system. The directed edges are labelled with relation identifiers from $\mathcal{L}$. There is a directed edge from u to v with label l whenever the two participate in the binary relation named by l. We make these notions formal in the following.

A ***relational structure*** (or simply a ***graph*** for brevity) is a pair $\langle V, \{R_l\}_{l \in \mathcal{L}} \rangle$, where $V \subseteq \mathcal{U}$ is a non-empty, finite set of vertices, and $\{R_l\}_{l \in \mathcal{L}}$ is a family of binary relations, indexed by the set $\mathcal{L}$. Each binary relation $R_l \subseteq V \times V$ specifies the vertex pairs that are related in a type-l relationship. Given a relational structure G, $V(G)$ is the vertex set of G, and $R_l(G)$ is the binary relation with index l. Given a *finite or countably infinite* set X, $\mathcal{G}(X)$ denotes the set of all relational structures for which the vertex set is a *nonempty*, *finite* subset of X.

Workflow Specification. Fixing a ReBAC system with a set of relation identifiers $\mathcal{L}$, a ***workflow specification*** $\mathcal{W}$ is a 3-tuple $\langle \mathcal{T}, \leq, \mathcal{C} \rangle$, where:

- $\mathcal{T}$ is a *finite* set of ***task identifiers***. A typical member of $\mathcal{T}$ is denoted by t. When a workflow is instantiated, a user $u \in \mathcal{U}$ will be assigned to execute each task $t \in \mathcal{T}$. User u is said to be the ***executor*** of task t.
- $\leq$ is a partial order over $\mathcal{T}$. It is required that the partial order has both a least element and a greatest element. Intuitively, if $t_1 \leq t_2$ then t_2 must not be executed before t_1. If t_1 and t_2 are incomparable, then their relative ordering is not restricted.
- $\mathcal{C}$, the ***constraint expression***, is a *positive* boolean combination of ***primitive constraints***. A primitive constraint has the form $l(t_i, t_j)$, where $l \in \mathcal{L}$ and $t_i, t_j \in \mathcal{T}$. The constraint requires that the executor of t_i must be related to the executor of t_j in a type-l relationship.

An example of a constraint expression is the following: $\big(l_1(t_1, t_2) \vee l_2(t_1, t_2)\big) \wedge l_3(t_1, t_2)$. Intuitively, the constraint expression is satisfied iff $l_3(t_1, t_2)$ as well as one of $l_1(t_1, t_2)$ or $l_2(t_1, t_2)$ are satisfied. Formally, each primitive constraint is interpreted as a proposition symbol in the propositional formula $\mathcal{C}$. A truth assignment to the propositional symbols can be represented as a family of binary relations $\{C_l\}_{l \in \mathcal{L}}$ (where each $C_l \subseteq \mathcal{T} \times \mathcal{T}$), such that $l(t_i, t_j)$ receives an assignment of true iff $(t_i, t_j) \in C_l$. Note that, as $\mathcal{C}$ does not contain negation, there is always at least one truth assignment that satisfies $\mathcal{C}$.

A constraint expression is ***conjunctive*** if it does not contain disjunctions.

Workflow Instances. Given a workflow specification $\mathcal{W} = \langle \mathcal{T}, \leq, \mathcal{C} \rangle$, a function $\pi : \mathcal{T} \rightharpoonup \mathcal{U}$ is called a ***workflow instance*** (or simply a ***plan***). Intuitively, π is an assignment of executors to tasks. If $dom(\pi) = \mathcal{T}$ (i.e., there is an user assignment for every task), then π is said to be ***complete***. Otherwise, π is ***partial***. If $dom(\pi) = \emptyset$ then π is ***empty***. Given a protection state $G = \langle V, \{R_l\}_{l \in \mathcal{L}} \rangle$, a workflow instance π is ***valid for*** $\mathcal{W}$ ***in*** G (or simply ***valid***) if the partial ordering and the constraint expression of $\mathcal{W}$ are both satisfied. Formally, π is valid if the following three conditions are all satisfied: (i) $ran(\pi) \subseteq V$; (ii) if $t_1 \leq t_2$ and $t_2 \in dom(\pi)$ then $t_1 \in dom(\pi)$ (such that no preceding task is left out); (iii) there exists a *satisfying* truth assignment $\{C_l\}_{l \in \mathcal{L}}$ for the constraint expression $\mathcal{C}$, such that for every $(t_1, t_2) \in dom(\pi) \times dom(\pi)$, if $(t_1, t_2) \in C_l$ then $(\pi(t_1), \pi(t_2)) \in R_l$.

Example 1. A hypothetical *Assignment Evaluation Workflow* needs to be run against a social network (Fig. 1) of an academic institution. Bob is teaching a class in which both Alice and Elham are enrolled, and for which both Charlene and Daniel are Teaching Assistants (TAs). In the social network, two vertices are related by a Not-Equal relationship iff they are distinct. Such edges are not displayed in Fig. 1 to preserve cleanliness.

Consider the workflow specification $\mathcal{W}$ with tasks and constraints depicted in Fig. 2. The tasks are ordered as in the following:

$$\text{Submission} < \text{Marking} < \text{Reviewing} < \text{Grading}$$

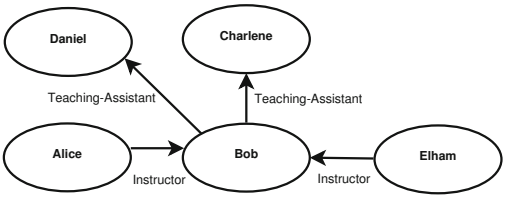

Fig. 1. Social Network without $\neq$ edges

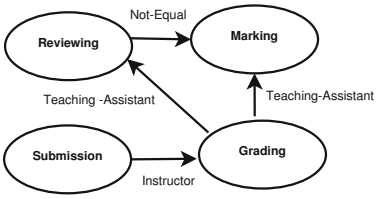

Fig. 2. Tasks and Constraints

The constraint expression is the conjunction of the following four primitive constraints:

1. The submitted assignment is graded by the instructor of the submitter:

$$\text{Instructor}(\text{Submission}, \text{Grading})$$

2. The assignment is marked by a TA assisting the instructor:

$$\text{Teaching-Assistant}(\text{Grading}, \text{Marking})$$

3. Reviewing is performed by a TA of the instructor:

$$\text{Teaching-Assistant}(\text{Grading}, \text{Reviewing})$$

4. The TA who reviews the marking is different from the TA who performs the marking:

$$\text{Not-Equal}(\text{Reviewing}, \text{Marking})$$

The four edges in Fig. 2 denote these four primitive constraints. The following workflow instance is complete and valid for $\mathcal{W}$ in G:

$$[\,\text{Submission} \mapsto \text{Alice}, \text{Grading} \mapsto \text{Bob}, \text{Marking} \mapsto \text{Charlene}, \text{Reviewing} \mapsto \text{Daniel}\,]$$

Expressiveness. We compare our workflow authorization model with those of Crampton *et al.* [4, 5] and Wang and Li [8, 6]. The goal is to point out that, the present formulation of ReBAC workflow authorization is sufficient for expressing all constraints that appeared in the literature.

The first kind of constraints we discuss about is ***role-step authorization constraints*** (what Wang and Li later called ***user-step authorization constraints***). The idea is to associate with a task a unary predicate specifying the set of users who can carry out that task. This can be captured readily in our model, as every unary user predicate P can be represented by a binary relation R_P, such that $P(u)$ holds whenever $(u, u) \in R_P$.

A second kind of constraints in [4, 5] (called ***entailment constraints***) has the form (D, t_1, t_2, ρ), where $D \subseteq V(G)$, $(t_1, t_2) \in \mathcal{T} \times \mathcal{T}$, and ρ is a binary relation. Such a constraint is simply a binary relation between the executors of t_1 and t_2 which can be encoded by a dedicated binary relation in ReBAC.

A third kind of constraints in [4] is ***cardinality constraints***, which come in two variants: local and global. A ***local cardinality constraint*** can be encoded as a number of entailment constraints [23]. A global cardinality constraint demands that at least k distinct executors must be involved in the execution of some n tasks. Such a constraint can be expressed as a disjunction of $\binom{n}{k}$ conjunctive constraints. Each of the conjunctive constraints demands that k of the n tasks are assigned distinct executors.

Wang and Li [8, 6] have ***simple binary constraints*** $\rho(t_1, t_2)$, which is identical to our primitive constraints. Their ***universal constraints*** $\rho(\forall X, t)$ can be expressed as a conjunction of primitive constraints (one for each task in X). The ***existential constraints*** $\rho(\exists X, t)$ can be encoded as a disjunction of primitive constraints (one for each task in X).

4 Workflow Satisfiability as Graph Homomorphism

Workflow satisfiability is the building block of workflow feasibility. In this section, we point out a novel connection between workflow satisfiability and graph homomorphism that will be used extensively in the sequel.

A workflow specification $\mathcal{W}$ is ***satisfiable*** in a protection state G if there is a complete workflow instance π that is valid for $\mathcal{W}$ in G.

Given a workflow specification $\mathcal{W} = \langle \mathcal{T}, \leq, \mathcal{C} \rangle$, a relational structure $\langle \mathcal{T}, \{C_l\}_{l \in \mathcal{L}} \rangle$ is a ***task network*** whenever $\{C_l\}_{l \in \mathcal{L}}$ is a *satisfying* truth assignment for the constraint expression $\mathcal{C}$. A task network is ***minimal*** if no proper subgraph corresponds to a satisfying truth assignment for $\mathcal{C}$. Let $TN(\mathcal{W})$ be the set of all *minimal* task networks for $\mathcal{W}$. Note that if $\mathcal{C}$ is conjunctive, then $TN(\mathcal{W})$ contains exactly one minimal task network. Fig. 2 shows the minimal task network for the workflow in Example 1.

A ***homomorphism*** from a relational structure $G = \langle V, \{R_l\}_{l \in \mathcal{L}} \rangle$ to another relational structure $G' = \langle V', \{R'_l\}_{l \in \mathcal{L}} \rangle$ is a function $h : V \to V'$ such that $(u, v) \in R_l$ implies $(h(u), h(v)) \in R'_l$. If such a function h exists, then G is ***homomorphic to*** G', and we write $G \to G'$. If, in addition, the function h is a bijection, and h^{-1} is a homomorphism from G' to G, then h is called an ***isomorphism***, and G is ***isomorphic to*** G'. In this case we write $G \cong G'$. We also write $h(G)$ for the ***homomorphic image*** of G, which is defined as the relational structure $\langle h(V), \{R''_l\}_{l \in \mathcal{L}} \rangle$ such that $R''_l = \{(h(u), h(v)) \mid (u, v) \in R_l\}$. Note that $h(G)$ is a subgraph of G'. Specifically, the homomorphic image $h(G)$ consists of only those vertices and edges in G' to which vertices and edges of G are mapped. Obviously, $G \to h(G)$ via h. The following proposition relates workflow satisfiability and homomorphism.

Proposition 1. *A workflow specification $\mathcal{W}$ is satisfiable in protection state G iff there exists a task network in $TN(\mathcal{W})$ that is homomorphic to G.*

The proposition is simply a corollary of (a) the definition of valid complete workflow instances as well as (b) the observation that a workflow is satisfiable iff it is satisfiable via a *minimal* task network (i.e., a member of $TN(\mathcal{W})$).

Deciding if one graph (or relational structure in general) is homomorphic to another is known to be NP-complete [17]. Thus workflow satisfiability for ReBAC is NP-hard even if the constraint expression is conjunctive. Moreover, the general problem is NP-complete since a nondeterministic Turing Machine can decide a problem instance in two steps: (i) guess a satisfying truth assignment for the constraint expression and a homomorphic mapping from the corresponding task network to the social network; (ii) verify the guesses (which takes polynomial time). This lower bound is not a new result [6]. The purpose here is to establish the connection between workflow satisfiability and graph homomorphism.

5 Workflow Feasibility

Workflow satisfiability assesses availability against the current protection state. We examine an alternative way of evaluation in this section by determining availability in the presence of state changes. We ask, is the workflow specification reasonably formulated, such that it can be instantiated in *some* protection state (among all possible states in the state space)? If yes, we say that the workflow specification is ***feasible***. As relationships change over time, a workflow that is unsatisfiable in one protection state can become satisfiable after a period of time. So the lack of availability in the current state does not necessarily mean that the permissions encapsulated in the workflow can not be made *available* ever. That is why we need to assess feasibility.

Not every possible social network is in the state space. Generally there are some topological restrictions governing the articulation of relationships in the social networks of a given business domain. These restrictions render some states (i.e., social networks) illegitimate. Social networks containing such an inconsistency do not belong to the state space. So, the definition of feasibility considers satisfiability with respect to a well-defined family of legitimate social networks. As we shall see, such a family is specified in terms of what we call a graph predicate.

Example 2. The following are examples of topological restrictions.

1. ***Antitransitivity***. Consider a relational structure describing the relationships among manufacturers, distributors and retailers. Suppose that there is a single relation supply, which relates a supplier to a consumer. A manufacturer shall not compete with its distributors. So we demand supply to be an ***antitransitive*** relation. That is, there shall not be x, y and z for which x supplies goods to y, y supplies goods to z, but x also supplies goods directly to z.
2. ***Acyclicity***. Suppose the supervise relation relates a supervisor to a person under her supervision. We expect this relation to be ***acyclic***. That is, the social network induced by this relation shall not contain a directed cycle.
3. ***Bipartiteness***. Say the vertices of a social network are firms. To prevent conflicts of interests, the firms that any firm deals with shall not deal with one another. To ensure this, the deals-with relation shall form a bipartite

graph (assuming deals-with is symmetric). A bipartite graph is a graph in which vertices can be divided into two disjoint partitions, such that no two vertices in the same partition are adjacent.

5.1 Graph Predicates

A ***graph predicate*** is a boolean function P with type $\mathcal{G}(\mathbb{N}) \to \{0, 1\}$, where $\mathbb{N}$ is the set of natural numbers. That is, given a relational structure G with vertices labelled by natural numbers, the predicate $P(G)$ returns a boolean value. In the following, we consider only graph predicates that are ***topology based*** [24]: i.e. $G \cong G'$ implies $P(G) = P(G')$. A topology-based graph predicate ignores vertex labelling. As vertex labelling is not important, we overload our notation such that graph predicates can be applied to either social networks or task networks. The ***negation*** of a graph predicate P, denoted by $\neg P$, is the graph predicate defined such that $(\neg P)(G) = 1$ iff $P(G) = 0$.

5.2 Families of Social Networks

The family of relational structures induced by a graph predicate P is defined as follows: $\mathcal{G}_P(X) = \{G \in \mathcal{G}(X) \mid P(G) = 1\}$. Intuitively, $\mathcal{G}_P(X)$ is the set of all relational structures from $\mathcal{G}(X)$ that satisfy the graph predicate P. When the state space is defined in this way, the graph predicate is called a ***characteristic predicate*** of the state space. Similarly, a family of relational structures can also be induced by a ***violation predicate*** (which is the negation of the characteristic predicate) as $\mathcal{G}^-_P(X) = \{G \in \mathcal{G}(X) \mid P(G) = 0\}$. Intuitively, a relational structure is excluded from the family iff it satisfies the violation predicate. We will make extensive use of violation predicates in the next section.

5.3 Workflow Feasibility Defined

A workflow specification $\mathcal{W}$ is said to be ***feasible for*** the family of social networks induced by a characteristic predicate P, iff there is a social network $G \in \mathcal{G}_P(\mathcal{U})$ such that $\mathcal{W}$ is satisfiable in G. We require P to be decidable in the following definition of workflow feasibility as a decision problem.

WORKFLOW FEASIBILITY
Instance: Workflow specification $\mathcal{W}$ and decidable graph predicate P.
Question: Is $\mathcal{W}$ feasible for graph family $\mathcal{G}_P(\mathcal{U})$?

A problem instance can also be specified with a decidable *violation predicate*. The reason is that the negation of a decidable graph predicate is also decidable.

Formulated in its full generality, workflow feasibility is recursively enumerable[2]. By placing further constraints on the choice of characteristic predicates, workflow feasibility can be rendered decidable, a topic to which we now turn.

[2] Feasible workflows can be enumerated as follows. Enumerate all pairs $(\mathcal{W}, G)$, where $\mathcal{W}$ is a workflow specification and $G \in \mathcal{G}(\mathcal{U})$. For each pair, check that (i) $P(G)$ and (ii) $\mathcal{W}$ is satisfiable in G. Output $\mathcal{W}$ if both checks succeed.

6 Feasibility via Refutation

While we have argued that feasibility is an important criterion of availability, the feasibility of a given workflow is not known to be decidable in the general case. In this section, we propose a ***refutation procedure*** to decide feasibility. We motivate our approach as follows. Under some special conditions to be stated below, if a task network satisfies a violation predicate P, then every social network that it is homomorphic to will also satisfy P. Therefore, if every task network in $TN(\mathcal{W})$ satisfies the violation predicate, then we can safely conclude that the workflow is infeasible. The refutation procedure is summarized below:

> To determine if $\mathcal{W}$ is feasible for $\mathcal{G}_P^-(\mathcal{U})$, evaluate $P(G)$ for every task network $G \in TN(\mathcal{W})$. If the test is positive in every case, then declare $\mathcal{W}$ "*infeasible*"; otherwise declare $\mathcal{W}$ "*maybe feasible*".

The procedure always terminates because there is only a finite number of truth assignments and P is decidable. It is a refutation procedure because it employs a *violation predicate* to detect if $\mathcal{W}$ is *unsatisfiable* for any member of $\mathcal{G}_P^-(\mathcal{U})$.

The examples below show that the refutation procedure is not always sound.

Example 3. Here are two violation conditions that can cause refutation to make unsound judgement.

1. ***Connectedness***. Consider the (irreflexive, symmetric) colleague relation in an organizational setting where it is an integrity requirement that the graph induced by the colleague relation must be connected. It would be an unsound decision by the refutation procedure to declare a task network infeasible when two tasks are found to be disconnected, as the two tasks could be assigned to users that are connected in the social network by other ways.
2. ***Quasi-reflexivity***. Suppose clerk is a unary predicate over users, marking off those users who are in the role of an administrative assistant. Suppose further that we would like to impose clerk as a role-step authorization constraint. Sect. 3 suggests that we can do so by encoding clerk as a binary relation. Such a relation must be ***quasi-reflexive*** (i.e., xRy implies $x = y$). It would be an unsound decision by the refutation procedure to declare a task network infeasible when two distinct tasks are found to be related by the clerk relation, as the two tasks could be assigned to the same user.

Therefore, some special conditions must be met in order for refutation to be sound. Sect. 6.1 specifies these conditions. Sect. 6.2 then considers the issue of ***completeness***: when the refutation procedure fails to declare that a workflow specification $\mathcal{W}$ is infeasible, can we then conclude that $\mathcal{W}$ is feasible?

6.1 Soundness

A graph predicate P is ***invariant over homomorphism*** if, for every relational structures G and H, $P(G)$ and $G \to H$ jointly imply $P(H)$. Now we can express the condition under which refutation is sound.

Theorem 1 (Soundness). *The refutation procedure is sound if the violation predicate P is invariant over homomorphism.*

The process of establishing that a violation predicate is invariant over homomorphism can be tedious. We therefore streamline this process by decomposing invariance over homomorphism into two easily checkable conditions: (a) ***monotonicity*** and (b) ***invariance over vertex contraction***. We define these two conditions in the sequel, and give examples of how they can be used for demonstrating that a violation predicate is invariant over homomorphism.

Monotonicity. A graph predicate P is monotonic if, when P is evaluated to true for a relational structure G, adding vertices and edges to G will not cause P to be evaluated to false. Formally, a graph predicate P is ***monotonic*** iff, for every relational structure G and H, $G \subseteq H$ implies $P(G) \Rightarrow P(H)$. In Example 3.1, the violation predicate is *not* monotonic, causing refutation to become unsound.

Invariance over Vertex Contraction. Intuitively, vertex contraction is an operation that merges two vertex in a relational structure. Formally, given a relational structure $G \in \mathcal{G}(X)$ and two vertices $u, v \in V$, we denote by $VC(G, u, v)$ the graph $G' = \langle V', \{R'_l\}_{l \in \mathcal{L}} \rangle$, where: $V' = V(G) \setminus \{ v \}$ and $R'_l = \{(h_{u,v}(x), h_{u,v}(y)) \mid (x, y) \in R_l(G)\}$. Here, the function $h_{u,v} : V \to V'$ is defined as follows: $h_{u,v}(x) = u$ if $x = v$, otherwise $h_{u,v}(x) = x$. In summary, $VC(G, u, v)$ is obtained from G by "folding" v into u. All the edges previously joining v now join u instead.

A graph predicate P is ***invariant over vertex contraction*** iff $P(G) \Rightarrow P(VC(G, u, v))$. Example 3.2 shows a violation predicate that is *not* invariant over vertex contraction, thereby causing refutation to become unsound.

Invariance over Homomorphism. To show that a graph predicate P is invariant over homomorphism, one could instead demonstrate that P is both monotonic and invariant over vertex contraction.

Theorem 2. *A graph predicate P is invariant over homomorphism iff it is both monotonic and invariant over vertex contraction.*

Proving invariance over homomorphism is usually tedious. Monotonicity, however, is usually quite trivial to establish. Similarly, invariance over vertex contraction is also quite manageable, as one only focuses on the effect of "merging" a pair of vertices.

Example 4. Consider the violating predicates for the restrictions in Example 2.

1. ***Antitransitivity***. The violation predicate detects if there exist vertices x, y and z (not necessarily distinct), for which supply holds between x and y, y and z, as well as x and z.
2. ***Acyclicity***. The violation predicate returns true whenever the graph contains a directed cycle.

3. ***Bipartiteness***. The violation predicate returns true whenever the graph contains an odd-length cycle [25, Theorem 1.2].

These violation predicates are monotonic and invariant over vertex contraction. In each case, violation corresponds to the presence of a specific graph structure, and thus monotonicity follows immediately. The violation of antitransitivity is invariant over vertex contraction: since the vertices x, y and z are not necessarily distinct, merging two of them preserves violation. A similar argument applies to the violation of acyclicity (resp. bipartiteness): contracting two vertices in a cycle (resp. odd cycle) produces a shorter cycle (resp. a shorter cycle of odd length).

6.2 Completeness

When the refutation procedure declares a workflow to be "*infeasible*", soundness guarantees that the declaration must be correct. Another possible declaration is "*maybe feasible*". If the declaration of "*maybe feasible*" always implies that the workflow is feasible, then we say that the refutation procedure is ***complete***. In a technical sense, the refutation procedure is complete.

Theorem 3 (Completeness). *The refutation procedure is complete if the violation predicate P is invariant over homomorphism.*

In practice, a violation predicate that precisely characterizes the state space may be highly complex, and thus it needs not be invariant over homomorphism. A pragmatic policy developer will be wise to use another predicate P' to approximate P, in such a way that (a) $P'(G) \Rightarrow P(G)$ for every $G \in \mathcal{G}(\mathcal{U})$, and (b) P' is invariant over homomorphism. In this case, the refutation procedure will be sound but not complete, as $\neg P'(\mathcal{W})$ does not guarantee $\neg P(\mathcal{W})$.

7 Using PDL for Violation Specification

This section examines the specification of violation predicates using a formal specification language. We have two design objectives for this language. The first objective concerns the appropriateness of the specified predicates. In order for the refutation procedure to be sound, violation predicates shall be invariant over homomorphism. We therefore demand the specification language to express only graph predicates that are invariant over homomorphism. The second objective concerns the efficiency of evaluating the specified predicates. We demand that the specification language captures only polynomial-time checkable graph predicates.

To meet both objectives, we adopt a monotonic fragment of the Propositional Dynamic Logic (PDL) [18] for specifying violation predicates. As we shall see, our PDL-based specification language is invariant over homomorphism, and has a polynomial-time model checking algorithm [26].

We have also introduced minor adaptations to the PDL for our purposes. Specifically, we observe in Example 4 that many violation predicates detect the presence of cyclic substructures. Our adaptation of PDL can be used for specifying the existence of such substructures.

7.1 Syntax and Semantics of PDL

We provide here a brief introduction to the syntax and semantics of the PDL fragment on which our specification language is based.

Suppose there is a finite or countably infinite set *Prop* of propositional symbols. The monotonic PDL fragment involves two types of constructs: ***formulas*** and ***relations***. The syntax of formulas (ϕ, ψ) and relations (α, β) are defined inductively in the following:

$$\alpha, \beta ::= l \mid -\alpha \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$
$$\phi, \psi ::= \top \mid p \mid \phi \vee \psi \mid \phi \wedge \psi \mid \langle \alpha \rangle \phi$$

where $l \in \mathcal{L}$ is a relation identifier, and $p \in \mathit{Prop}$ is a propositional symbol. Informally, relations are regular expressions with the relation identifiers as the alphabet. Specifically, our regular expression language offers the following relation combinators: converse ($-$), alternation ($\cup$), concatenation ($;$) and Kleene star (*). Lastly, formulas are modal logic formulas with relations as modalities. Note that we only consider monotonic constructs in the language. Note also that we do not include the "$\phi?$" relation in PDL (originally for constructing conditional relations). As we shall see, this last design decision has a significant impact on the complexity of model checking.

The formulas and relations are interpreted over a relational structure $G = \langle V, \{R_l\}_{l \in \mathcal{L}} \rangle$ and a labelling function $L : \mathit{Prop} \to 2^V$. The labelling function specifies for each propositional symbol the set of vertices for which the propositional symbol is true. Two interpretation functions μ_G and $\rho_{G,L}$ respectively provide the interpretations of relations and formulas (for the specific G and L). Specifically, $\mu_G(\alpha) \subseteq V \times V$ is a binary relation over the vertex set of G. Similarly, $\rho_{G,L}(\phi) \subseteq V$ identifies the set of vertices in G for which the formula ϕ is satisfied. When the choice of G and L is clear, we simply write $\mu(\alpha)$ and $\rho(\phi)$ for brevity. The semantics of the language is defined as follows.

$$\begin{aligned}
\mu(l) &= R_l & \rho(\top) &= V \\
\mu(-\alpha) &= \mu(\alpha)^{-1} & \rho(p) &= L(p) \\
\mu(\alpha \cup \beta) &= \mu(\alpha) \cup \mu(\beta) & \rho(\phi \vee \psi) &= \rho(\phi) \cup \rho(\psi) \\
\mu(\alpha; \beta) &= \mu(\alpha) \circ \mu(\beta) & \rho(\phi \wedge \psi) &= \rho(\phi) \cap \rho(\psi) \\
\mu(\alpha^*) &= \mu(\alpha)^* & \rho(\langle \alpha \rangle \phi) &= \{u \in V \mid \exists v \in \rho(\phi) \,.\, (u, v) \in \mu(\alpha)\}
\end{aligned}$$

In words, l is interpreted simply as the binary relation R_l in G. The converse $-\alpha$ is interpreted as the inverse of the relation represented by α. Alternation $\alpha \cup \beta$ is interpreted as the union of the relations represented by α and β. The concatenation $\alpha; \beta$ is the relational composition of the relations represented by α and β. The Kleene star α^* is interpreted as the reflexive transitive closure of the relation represented by α. The constant formula $\top$ is satisfied by all vertices in G. The propositional symbol p is satisfied by the vertices in $L(p)$. Disjunction and conjunction are interpreted as set union and intersection. The formula $\langle \alpha \rangle \phi$ is satisfied by those vertices with an α-neighbor that in turn satisfies ϕ.

We write $G, L, u \models \phi$ whenever $u \in \rho_{G,L}(\phi)$. When this is the case, the graph pattern specified by ϕ is satisfied in G at vertex u with labelling function L.

7.2 Specifying Violation Predicates with PDL

Without using propositional symbols (i.e., $Prop = \emptyset$), the above language allows us to write formulas that represent path patterns (with $\top$ as the pattern for the empty path). Recall we observe from Example 4 that many practical violation predicates detect cyclic structures in the social network. We therefore customize the monotonic PDL fragment above for representing cyclic structures. Specifically, we adopt exactly one propositional symbol org, called the ***anchor***. That is, we set $Prop = \{\mathsf{org}\}$. The anchor proposition org names the vertex from which the search for cyclic pattern begins[3]. In this way, we can write formulas that detect if the initial vertex is revisited, and thus a cycle is found. Formally, let $[\mathsf{org} \mapsto u]$ denote the labelling function $L : \{\mathsf{org}\} \to \mathcal{U}$ for which $L(\mathsf{org}) = u$. Then the following checks if the cyclic structure specified by ϕ exists in G by initiating the search from the vertex u.

$$G, [\mathsf{org} \mapsto u], u \models \phi$$

We write $[\![\phi]\!]$ to denote the graph predicate that takes a graph G as argument and returns true iff:

$$\text{there exists } u \in V(G) \text{ such that } G, [\mathsf{org} \mapsto u], u \models \phi.$$

That is, $[\![\phi]\!]$ searches for a vertex $u \in V(G)$ at which ϕ is satisfied. The space of protection states induced by ϕ is therefore $\mathcal{G}^{-}_{[\![\phi]\!]}(\mathcal{U})$.

Example 5. The formulas below specify the violation predicates in Example 4.

1. ***Antitransitivity***. Note the use of converse $(-)$ in this example.

$$\langle \mathsf{supply}; \mathsf{supply}; -\mathsf{supply} \rangle \mathsf{org}$$

2. ***Acyclicity***. Note the use of Kleene star $(^*)$ in this example.

$$\langle \mathsf{supervise}; \mathsf{supervise}^* \rangle \mathsf{org}$$

3. ***Bipartiteness***. Let l be a shorthand for $\mathsf{deals\text{-}with}$. Note the use of alternation $(\cup)$ in this example.

$$\langle (l \cup -l); ((l \cup -l); (l \cup -l))^* \rangle \mathsf{org}$$

The following is a corollary of a well-known result [28, Chapter 2].

Theorem 4. *For every formula ϕ in the above monotonic fragment of PDL, $[\![\phi]\!]$ is invariant over homomorphism.*

Invariance is due to our choice of a monotonic fragment of PDL. Therefore, specifying violation predicates with our PDL fragment preserves the soundness of refutation.

[3] A propositional symbol that names a specific vertex is called a ***nominal*** in hybrid logic [27, 16].

7.3 Model Checking

We show that evaluating a graph predicate $[\![\phi]\!]$ takes polynomial time. This involves testing the $\models$ relation at every vertex of the graph argument. To this end, we describe an adaptation of Lange's PDL model checking algorithm [26].

Suppose $G = \langle V, \{R_l\}_{l \in \mathcal{L}} \rangle$. Let $n = |V|$. We represent a binary relation over V as a boolean square matrix (i.e., an $n \times n$ matrix with boolean entries). Specifically, $\mu(\alpha)$ returns such a matrix. We also represent a subset of V by a boolean column vector of size n. Specifically, $\rho(\phi)$ returns such a column vector.

With this data representation, μ can be evaluated inductively using matrix operations. As the base case, $\mu(l)$ is simply the boolean matrix representation of R_l. Now, $\mu(-\alpha)$ involves transposing a matrix, and $\mu(\alpha \cup \beta)$ involves bitwise-or. Both operations take $O(n^2)$ time. Next, $\mu(\alpha; \beta)$ involves (boolean) matrix multiplication, while $\mu(\alpha^*)$ involves reflexive transitive closure. Both operations are $O(n^3)$. In summary, $O(n^3)$ time is needed for each relation subexpression.

In a similar fashion, ρ can be evaluated inductively. The picture, however, is sightly more complex. The evaluation of $\rho_{G,[\mathsf{org} \mapsto u]}$ must be performed for every $u \in V$. The first base case $\rho_{G,[\mathsf{org} \mapsto u]}(\top)$ is a column vector with all entries set to 1. The second base case $\rho_{G,[\mathsf{org} \mapsto u]}(\mathsf{org})$ is a column vector with a 1 in u's row, and 0 everywhere else. The evaluation of $\rho_{G,[\mathsf{org} \mapsto u]}(\phi \vee \psi)$ and $\rho_{G,[\mathsf{org} \mapsto u]}(\phi \wedge \psi)$ involve bitwise-or and bitwise-and respectively, both are $O(n)$ operations. The evaluation of $\rho_{G,[\mathsf{org} \mapsto u]}(\langle \alpha \rangle \phi)$ involves the multiplication of the $n \times n$ matrix $\mu_G(\alpha)$ with the column vector $\rho_{G,[\mathsf{org} \mapsto u]}(\phi)$, which is an $O(n^2)$ operation. In summary, evaluating $\rho_{G,[\mathsf{org} \mapsto u]}(\phi)$ for a single $u \in V$ is an $O(n^2)$ operation. Therefore, evaluating $\rho_{G,[\mathsf{org} \mapsto u]}(\phi)$ for every $u \in V$ takes $O(n^3)$ time.

Since the processing of each subexpression or subformula takes $O(n^3)$ time, and there are all together $|\phi|$ subexpressions and subformulas, the evaluation of $[\![\phi]\!]$ takes $O(|\phi| \times n^3)$ time. Although a third-degree polynomial is involved, the complexity is actually quite reasonable, as the refutation procedure evaluates the violation predicate against task networks rather than social networks. While the latter typically have an intimidating size, the former have a much more manageable size. This same assumption regarding the small number of tasks in a typical workflow has been used by other authors also [6].

Our design choice of not including the "$\phi?$" construct from PDL pays dividend in the time complexity of model checking. Had we included that construct, the interpretation function μ_G would be parameterized not only by G, but also the labelling function (as in $\mu_{G,L}$). As a result, we would have to perform an $O(n^3)$ operation for every relation subexpression *and* for every labelling function $[\mathsf{org} \mapsto u]$, where $u \in V$. The final tally for the time complexity of model checking would then be $O(|\phi| \times n^4)$ instead of our current $O(|\phi| \times n^3)$.

7.4 Complexity of Refutation

When the constraint expression is conjunctive, *the* minimal task network can be readily constructed from the constraint expression, and thus refutation can be conducted in polynomial time by applying the above model checking algorithm

over the minimal constraint network. But with this restriction of expressiveness, certain constraints such as global cardinality constraints [4] and existential constraints [8, 6] can not be expressed.

The following theorem asserts that refutation using PDL formulas is computationally hard for general constraint expressions.

Theorem 5. *Given a workflow $\mathcal{W}$ and a PDL formula ϕ, deciding if $[\![\phi]\!](G)$ is true for every task network $G \in TN(\mathcal{W})$ is coNP-complete.*

This result implies that workflow feasibility, of which refutation is a co-problem, is NP-complete when the violation predicate is specified in PDL. Intuitively, feasibility is hard in the general setting because it involves a search for a non-violating task network among all task networks. In contrast, there is only one minimal task network when the constraint expression is conjunctive, and it takes only polynomial time to check if this minimal task network satisfies a graph predicate that is specified in PDL.

8 Conclusion and Future Work

We introduced a workflow authorization model in the framework of ReBAC. The model offers a fresh characterization of the workflow satisfiability problem in terms of graph homomorphism. Armed with this new understanding, we studied a new decision problem called workflow feasibility, which is the dual of workflow resiliency. A refutation procedure was proposed for deciding feasibility. The refutation procedure was shown to be sound and complete if the space of protection states can be characterized by a violation predicate that is invariant over graph homomorphism. To facilitate verification, we proposed two verification conditions that jointly imply invariance over homomorphism. We also proposed the adoption of a monotonic fragment of PDL as a language for specifying the violation predicate. We showed that with this specification language the time complexity of refutation is polynomial for conjunctive constraints, and coNP-complete for general constraint expressions.

We highlight some possible extensions of this work. The monotonic fragment of PDL used in this paper employs the anchor nominal org for detecting revisited vertices. A possible extension of this feature involves the hybridization of the monotonic fragment of PDL [27, 16]. This extension will allow us to capture complex graph patterns while maintaining invariance over homomorphism. It is unknown how this extension will impact the time complexity of model checking.

We pointed out that feasibility can be seen as the dual of resiliency. This perspective generates a new form of resiliency problem, in which deformative transitions do not remove users (as in [8, 6]), but instead remove relationships (i.e., the social network is evolving). This problem has not been studied in existing literature. As user removal can be simulated via relationship removal (i.e., presence of users can be captured by a unary predicate present, which in turn can be encoded using a binary relation), such a problem is a generalization of Wang and Li's notion of resiliency in [8, 6].

Acknowledgments. This work is supported in part by an NSERC Discovery Grant and a Canada Research Chair.

References

1. Thomas, R.K., Sandhu, R.S.: Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Autorization Management. In: Proceedings of the 11th IFIP WG11.3 Working Conference on Database and Application Security (DAS 1998), Lake Tahoe, California, USA, pp. 166–181 (1998)
2. Atluri, V., Huang, W.K.: An Authorization Model for Workflows. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 44–64. Springer, Heidelberg (1996)
3. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Transactions on Information and System Security 18(1), 65–104 (1999)
4. Tan, K., Crampton, J., Gunter, C.A.: The consistency of task-based authorization constraints in workflow systems. In: Proceedings of the 17th IEEE Workshop on Computer Security Foundations (CSFW 2004), pp. 155–169. IEEE Computer Society, Washington, DC (2004)
5. Crampton, J.: 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), Stockholm, Sweden, pp. 38–47 (2005)
6. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM Transactions on Information and System Security 13(4), 40:1–40:35 (2010)
7. Baldwin, R.W.: Naming and grouping privileges to simplify security management in large databases. In: Proceedings of the 1990 IEEE Symposium on Security and Privacy (S&P 1990), Oakland, CA, USA, pp. 116–132 (May 1990)
8. Wang, Q., Li, N.: Satisfiability and Resiliency in Workflow Systems. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 90–105. Springer, Heidelberg (2007)
9. Li, N., Tripunitara, M.V., Wang, Q.: Resiliency policies in access control. In: Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006), Alexandria, VA, USA, pp. 113–123 (October 2006)
10. Li, N., Wang, Q., Tripunitara, M.: Resiliency policies in access control. ACM Transactions on Information and System Security 12(4) (April 2009)
11. Bertino, E., Ferrari, E., Atluri, V.: A flexible model supporting the specification and enforcement of role-based authorization in workflow management systems. In: Proceedings of the Second ACM Workshop on Role-based Access Control (RBAC 1997), Fairfax, Virginia, United States, pp. 1–12 (1997)
12. Ahn, G.J., Sandhu, R., Kang, M., Park, J.: Injecting RBAC to secure a web-based workflow system. In: Proceedings of the Fifth ACM Workshop on Role-Based Access Control (RBAC 2000), Berlin, Germany, pp. 1–10 (2000)
13. Kandala, S., Sandhu, R.: Secure role-based workflow models. In: Proceedings of IFIP WG11.3 Working Conference on Database and Application Security (DAS 2001), Niagara, Ontario, Canada, pp. 45–58 (2001)
14. Fong, P.W.L.: Relationship-based access control: protection model and policy language. In: Proceedings of the First ACM Conference on Data and Application Security and Privacy (CODASPY 2011), San Antonio, TX, USA, pp. 191–202 (February 2011)

15. Fong, P.W.L., Siahaan, I.: Relationship-based access control policies and their policy languages. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT 2011), Innsbruck, Austria, pp. 51–60 (June 2011)
16. Bruns, G., Fong, P.W.L., Siahaan, I., Huth, M.: Relationship-based access control: Its expression and enforcement through hybrid logic. In: Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY 2012), San Antonio, TX, USA (February 2012)
17. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. Oxford University Press (2004)
18. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences 18(2), 194–211 (1979)
19. Casati, F., Castano, S., Fugini, M.: Managing workflow authorization constraints through active database technology. Information Systems Frontiers 3(3), 319–338 (2001)
20. Crampton, J., Huth, M.: Synthesizing and verifying plans for constrained workflows: Transferring tools from formal methods. In: Proceedings of the 2011 Workshop on Verification and Validation of Planning and Scheduling Systems (June 2011)
21. Huang, W.K., Atluri, V.: Secureflow: a secure Web-enabled workflow management system. In: Proceedings of the Fourth ACM Workshop on Role-based Access Control (RBAC 1999), Fairfax, Virginia, United States, pp. 83–94 (1999)
22. Warner, J., Atluri, V.: Inter-instance authorization constraints for secure workflow management. In: Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies (SACMAT 2006), Lake Tahoe, California, USA, pp. 190–199 (2006)
23. Crampton, J.: On the satisfiability of constraints in workflow systems. Technical Report RHUL-MA-2004-1, Department of Mathematics, Royal Holloway, University of London (2004)
24. Fong, P.W.L., Anwar, M., Zhao, Z.: A Privacy Preservation Model for Facebook-Style Social Network Systems. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 303–320. Springer, Heidelberg (2009)
25. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. North-Holland (1976)
26. Lange, M.: Model checking propositional dynamic logic with all extras. Journal of Applied Logic 4(1), 39–49 (2006)
27. Areces, C., ten Cate, B.: Hybrid logics. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic. Elsevier (2007)
28. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic, Cambridge (2002)