The Consistency of Task-Based Authorization Constraints in Workflow Systems[*]

Kaijun Tan
Department of Computer Science
University of Pennsylvania

Jason Crampton
Information Security Group
Royal Holloway, University of London

Carl A. Gunter
Department of Computer Science
University of Pennsylvania

Abstract

Workflow management systems (WFMSs) have attracted a lot of interest both in academia and the business community. A workflow consists of a collection of tasks that are organized to facilitate some business process specification. To simplify the complexity of security administration, it is common to use role-based access control (RBAC) to grant authorization to roles and users. Typically, security policies are expressed as constraints on users, roles, tasks and the workflow itself. A workflow system can become very complex and involve several organizations or different units of an organization, thus the number of security policies may be very large and their interactions very complex. It is clearly important to know whether the existence of such constraints will prevent certain instances of the workflow from completing. Unfortunately, no existing constraint models have considered this problem satisfactorily.

In this paper we define a model for constrained workflow systems that includes local and global cardinality constraints, separation of duty constraints and binding of duty constraints. We define the notion of a workflow specification and of a constrained workflow authorization schema. Our main result is to establish necessary and sufficient conditions for the set of constraints that ensure a sound constrained workflow authorization schema, that is, for any user or any role who are authorized to a task, there is at least one complete workflow instance when this user or this role executes this task.

1 Introduction

In recent years, workflow management systems have attracted significant interest from the research community. Typical examples of workflows include purchase order processing, the handling and refereeing of papers in electronic journals, and the processing of tax refunds.

Informally, a workflow is a collection of tasks which are organized to facilitate some business process specification, re-engineering and automation [11]. Workflow management systems (WFMSs) make it possible to deploy business processes within a computerized system, thereby gaining significant improvements in efficiency. However, there remain significant challenges to be resolved before we see widespread use of sophisticated commercial WFMSs. Of particular significance to the security community is the administration of security-relevant information and the specification of authorization policies and constraints.

The study of authorization constraints has a long history dating back to the Chinese Wall model of Brewer and Nash [7]. The specification and enforcement of authorization constraints has recently received a lot of attention in the role-based community [1, 10, 12, 17].

In the past decade, a considerable amount of work has also been done on the use of role-based access control (RBAC) to support access control in workflow systems [2, 3, 5, 6, 14]. RBAC is a natural choice for workflow systems because tasks are synonymous with permissions and can be assigned to roles.

However, a role-based model alone is not sufficient to meet all the requirements of security policies of an organization. There exist several schemes and models in the literature for specifying different kinds of constraints in workflow systems, such as separation of duty constraints [4, 5, 6, 8, 13, 15, 18], binding of duties constraints [8, 18] and cardinality constraints [5]. Separation of duty requirements exist to prevent conflicts of interest and to make fraudulent acts more difficult to commit. Binding of duty constraints require that if a certain user executed a particular task then that user must also execute a second task in the workflow.

A workflow system may span multiple organization

[*]Appearing in Computer Security Foundations Workshop (CSFW) 2004

units, or even multiple organizations, each of them having its own security policies. Thus the number of constraints may grow very large and the interactions between constraints will be complex and difficult to predict. The purpose of constraints is to enforce a secure workflow, but they should not prevent a workflow being completed. Given a set of constraints and sets of authorized users and roles for each task, there should exist a workflow execution that satisfies all the constraints. Therefore, the specification of constraints is a difficult task and the ability to determine whether a set of constraints is consistent with the security requirements of the organization and permits every instance of a workflow to complete successfully would be useful to security administrators.

Unfortunately, solutions to this problem have largely been overlooked in the literature. Bertino *et al.* [5] consider the completion of workflow instances and solve the problem by preventing an authorized user from performing a task if it would mean subsequent tasks would have no authorized users that satisfied the workflow constraints. Wainer *et al.* [18] tackled this problem from another direction: namely, if a workflow can not complete because of the constraints, then some constraints may be overridden. Our argument is that an unsuccessful workflow execution may be caused by the inconsistency within the specification of the constraints.

In this paper, we focus on solving the consistency problem of the constraints. We provide an approach to help the workflow designers to define a sound constrained workflow authorization schema. So the first contribution of this paper is to define consistency rules for constraint-task pairs that guarantee there is no inconsistency, ambiguities and redundancy contained in the set of constraints. When the constraint-task pairs conform to these rules, then for each user and each role authorized in a task in the workflow, there is at least one successful workflow instance that satisfies all the constraints.

In the context of workflow systems, we believe that existing approaches do not make a sufficiently clear distinction between the specification of a workflow and a workflow instance. A workflow can be viewed as an abstraction that describes the ordering of the tasks within the workflow and the number of executions for each task, upon which authorization and constraints can be specified. A workflow instance or execution is an instantiation of the corresponding workflow specification and consists of a set of task instances upon which the authorization and constraints must be enforced. The second contribution of this paper is to explicitly define a workflow instance and a task instance. This leads to a natural definition of a *workflow authorization schema*, which associates roles with tasks in the workflow specification. We then extend this definition to define a *constrained workflow authorization schema*, which specifies the constraints that should be enforced on each instance of the workflow.

We also believe that existing approaches to the specification of authorization constraints are unnecessarily complicated. There exist similarly complicated approaches to separation of duty constraints in the RBAC literature. The third contribution of this paper is to describe a simple tuple-based method for constraint specification, making the expression and verification of authorization constraints easy for both system administrators and application writers. The scheme is based on a model for separation of duty constraints in role-based systems [10]. We identify two constraint types: *cardinality constraints* and *entailment constraints*. Entailment constraints can be used to model separation of duty and binding of duty constraints.

The specification language associates constraints with tasks rather than a workflow, thereby facilitating distributed authorization and enforcement of constraints. (Usually, a centralized WFMS system is not desirable [3] because the reference monitor can become a performance bottleneck.)

In the next section we introduce some preliminary concepts and notation from RBAC literature. In the following section, we describe our model for workflows. In section 4, we discuss the specification of three different kinds of constraints using tuples and explicitly define the meaning of these constraints. In section 5, we define a set of consistency rules that can guarantee a sound constrained workflow authorization schema. The last section suggests some future research directions.

2 Role-based access control

It is natural to use role-based access control (RBAC) in order to determine who is permitted to perform tasks within a workflow: the permission to execute a task can be associated with one or more roles and users can be associated with those roles; and it provides natural support for implementing any separation of duty requirements on task execution. We will base our discussion on the RBAC96 family of models [16]. In particular, we use the RBAC$_1$ model, which supports the use of a role hierarchy. With this model, if a role is authorized to play a task, the roles dominating this role will inherit the execution authorization.

We assume the existence of a partially ordered set of roles $\langle R, \leqslant \rangle$ that can be visualized as a role hierarchy. Users are associated with roles using the user-role assignment relation $UA \subseteq U \times R$. Similarly, permissions are associated with roles using the permission-role assignment relation $PA \subseteq P \times R$, where P denotes the set of permissions. We will assume that a task is an atomic action within the context of a workflow instance. Hence we will assume that a task is synonymous with a permission.

If $(p, r) \in PA$ and $(u, r) \in UA$, then we say that p is *explicitly assigned* to r and u is *explicitly assigned* to r. We assume that the set of roles explicitly assigned to a permission is an antichain in R.[1] However, unlike the RBAC96 model, we also assume that all user-role assignments have to be made explicit. In particular, if u is explicitly assigned to r, then u is not implicitly assigned to any role $r' < r$; such assignments would have to be explicitly entered in the UA relation.

3 A formal model for workflow systems

A *workflow specification* is a triple $(\mathsf{T}, \leqslant, \rho)$ where $\langle \mathsf{T}, \leqslant \rangle$ is a finite partially ordered set of tasks with a least element t_s and a greatest element t_f and $\rho : \mathsf{T} \to \mathbb{N}$ is a function indicating how many times a task must be executed in each instance of the workflow. We assume $\mathsf{t}_s \neq \mathsf{t}_f$.

A *workflow authorization schema* over a partially ordered set of roles $\langle R, \leqslant \rangle$ is a pair $(\mathsf{S}, PA_\mathsf{T})$, where $\mathsf{S} = (\mathsf{T}, \leqslant, \rho)$ is a workflow specification and $PA_\mathsf{T} \subseteq \mathsf{T} \times R$ is the *task-role assignment relation* for T. Given a task $\mathsf{t} \in \mathsf{T}$, we define

$$R(\mathsf{t}) = \{r' \in R : \exists(\mathsf{t}, r) \in PA_\mathsf{T}, r' \geqslant r\}$$
$$U(\mathsf{t}) = \{u \in U : (u, r) \in UA, r \in R(\mathsf{t})\}.$$

In other words, $R(\mathsf{t})$ is the set of roles authorized to perform t; $U(\mathsf{t})$ is the set of users authorized to perform t. Analogously, given $r \in R$ and $R' \subseteq R$, we will write

$$U(r) = \{u \in U : (u, r) \in UA\}$$
$$U(R') = \bigcup_{r \in R'} U(r).$$

Definition 1 *Given a workflow specification* $((\mathsf{T}, \leqslant, \rho), PA_\mathsf{T})$, *a* workflow instance I *is a list of task instances* $[(\mathsf{t}_1, u_1, r_1), \ldots, (\mathsf{t}_n, u_n, r_n)]$, *where* (t_i, u_i, r_i) *is an instance of task* $\mathsf{t}_i \in \mathsf{T}$ *performed by user* $u_i \in U(\mathsf{t}_i)$ *acting in role* $r_i \in R(\mathsf{t}_i)$ *and if* $\mathsf{t}_i < \mathsf{t}_j$ *then* t_i *precedes* t_j *in the list. A workflow instance* $[(\mathsf{t}_1, u_1, r_1), \ldots, (\mathsf{t}_n, u_n, r_n)]$ *is* complete *if for all* $\mathsf{t} \in \mathsf{T}$, *the number of occurrences of* t *equals* $\rho(\mathsf{t})$.

Definition 2 *A workflow instance* $[(\mathsf{t}_1, u_1, r_1), \ldots, (\mathsf{t}_n, u_n, r_n)]$ *is*

- before $\mathsf{t} \in \mathsf{T}$ *if it is a complete instance of the workflow specification* $((\mathsf{T}', \leqslant, \rho'), PA'_\mathsf{T})$, *where* $\mathsf{T}' = \{\mathsf{t}' \in \mathsf{T} : \mathsf{t}' < \mathsf{t}\}$, $PA'_\mathsf{T} = \{(\mathsf{t}, r) \in PA_\mathsf{T} : \mathsf{t} \in \mathsf{T}'\}$ *and* $\rho' : \mathsf{T}' \to \mathbb{N}$ *such that* $\rho'(\mathsf{t}') = \rho(\mathsf{t}')$ *for all* $\mathsf{t}' \in \mathsf{T}'$*;*

- during $\mathsf{t} \in \mathsf{T}$ *if it is a complete instance of the workflow specification* $((\mathsf{T}', \leqslant, \rho'), PA'_\mathsf{T})$, *where* $\mathsf{T}' = \{\mathsf{t}' \in \mathsf{T} : \mathsf{t}' \leqslant \mathsf{t}\}$, $PA'_\mathsf{T} = \{(\mathsf{t}, r) \in PA_\mathsf{T} : \mathsf{t} \in \mathsf{T}'\}$ *and* $\rho' : \mathsf{T}' \to \mathbb{N}$ *such that* $\rho'(\mathsf{t}') = \rho(\mathsf{t}')$ *for all* $\mathsf{t}' < \mathsf{t}$ *and* $0 < \rho'(\mathsf{t}) < \rho(\mathsf{t})$*;*[2]

- after $\mathsf{t} \in \mathsf{T}$ *if it is a complete instance of the workflow specification* $((\mathsf{T}', \leqslant, \rho'), PA'_\mathsf{T})$, *where* $\{\mathsf{t}' \in \mathsf{T} : \mathsf{t}' \leqslant \mathsf{t}\} \subseteq \mathsf{T}'$, $PA'_\mathsf{T} = \{(\mathsf{t}, r) \in PA_\mathsf{T} : \mathsf{t} \in \mathsf{T}'\}$ *and* $\rho' : \mathsf{T}' \to \mathbb{N}$ *such that* $\rho'(\mathsf{t}') = \rho(\mathsf{t}')$ *if* $\mathsf{t}' \leqslant \mathsf{t}$.

A workflow instance $[(\mathsf{t}_1, u_1, r_1), \ldots, (\mathsf{t}_n, u_n, r_n)]$ *is a* partial instance *of the workflow specification* $((\mathsf{T}, \leqslant, \rho), PA_\mathsf{T})$ *if it is a complete execution of the workflow specification* $((\mathsf{T}', \leqslant, \rho'), PA'_\mathsf{T})$, *where* T' *is an order ideal in* T *and* $\rho' : \mathsf{T}' \to \mathbb{N}$ *such that* $\rho'(\mathsf{t}) \leqslant \rho(\mathsf{t})$.[3]

We denote the set of instances before t by $Pre(\mathsf{t})$, the set of instances during t by $In(\mathsf{t})$ and the set of instances after t by $Post(\mathsf{t})$. If $I \in Post(\mathsf{t})$, then we say that task t has *completed* in I. For any instance of t in I, if it is performed by user u acting in role r, then we say t completes with u and r; t may complete with several users and roles in I if it has several different instances in I.

Definition 3 *Let* $I = [(\mathsf{t}_1, u_1, r_1), \ldots, (\mathsf{t}_n, u_n, r_n)]$ *be a partial instance of a workflow. We say* $I' = [(\mathsf{t}_1, u_1, r_1), \ldots, (\mathsf{t}_n, u_n, r_n), (\mathsf{t}, u, r)]$ *is an* evolution *of* I, *which we will denote* $[I, (\mathsf{t}, u, r)]$.

Let $I \in Pre(\mathsf{t})$ and let the set of tasks executed in I be T', then I may evolve into $[I, (\mathsf{t}, u, r)]$ for any t such that $\mathsf{T}' \cup \{\mathsf{t}\}$ is an order ideal in T. Clearly, $[I, (\mathsf{t}, u, r)] \in In(\mathsf{t})$ if $\rho(\mathsf{t}) > 1$ and $[I, (\mathsf{t}, u, r)] \in Post(\mathsf{t})$ otherwise. Similarly, if $I \in In(\mathsf{t})$, then $[I, (\mathsf{t}, u, r)] \in In(\mathsf{t})$ or $[I, (\mathsf{t}, u, r)] \in Post(\mathsf{t})$.

As we observed in the introduction, a role-based model alone is generally not sufficient to meet all the requirements of the security policies of an organization. There exist several schemes and models in the literature for specifying different kinds of constraints in workflow systems. Bertino *et al.* [5] identified three different types of constraints: static, dynamic and hybrid constraints. These types differ in when they are evaluated: static constraints can be evaluated without executing a workflow instance; dynamic constraints can be evaluated only during the execution of a workflow instance because they express restrictions based on the execution history of the workflow; hybrid constraints can be partially verified without executing the workflow. In our paper, we are only interested in hybrid constraints because we want to statically check the consistency of such constraints defined in a workflow.

[1]In other words, if $(p, r), (p, r') \in PA$ with $r \neq r'$, then $r \not\leqslant r'$ and $r' \not\leqslant r$.

[2]If $\rho(\mathsf{t}) = 1$, then it is not possible for a workflow instance to be during t (since there is a single execution of the task).

[3]Given a poset X, $Y \subseteq X$ is an *order ideal* if for all $y \in Y$ and $x \in X$, $x \leqslant y$ implies that $x \in Y$.

The example workflow specification we will use throughout this paper is a combination of examples used by Bertino *et al.* [5] and Knorr and Stormer [15]. It is based on a workflow that is used to produce payments for tax refunds and contains the following tasks:

($\mathtt{t}_1$) A clerk prepares a cheque for a tax refund;

($\mathtt{t}_2$) The cheque is approved or denied by two different managers;

($\mathtt{t}_3$) A third manager makes a final decision (based on the decisions made in task 2);

($\mathtt{t}_4$) A clerk issues or voids the cheque (based on the decision made in task 3).

Figure 1a represents the workflow specification. Figure 1b represents a role hierarchy, where GM denotes General Manager, RM denotes Refund Manager, TM denotes Technical Manager and RC denotes Refund Clerk. We denote the two instances of $\mathtt{t}_2$ by $\mathtt{t}_{2,1}$ and $\mathtt{t}_{2,2}$. Figure 1 also includes the UA and PA relations.

In addition, we articulate the following constraints:

(c_1) The two instances of $\mathtt{t}_2$ should be performed by different users [5, 15];

(c_2) $\mathtt{t}_2$ and $\mathtt{t}_3$ should be performed by different users [5];

(c_3) $\mathtt{t}_1$ and $\mathtt{t}_4$ should be performed by different users [5];

(c_4) $\mathtt{t}_2$ must be performed by a role that is more senior than the role that performed $\mathtt{t}_1$ unless $\mathtt{t}_1$ and $\mathtt{t}_2$ are performed by GM [5];

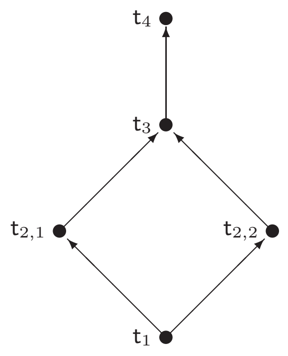
(c_5) $\mathtt{t}_1$ and $\mathtt{t}_2$ must be performed by different users [15];

(c_6) `Bob` should not be able to perform task $\mathtt{t}_4$ if his sister `Alice` performed task $\mathtt{t}_1$ [15];

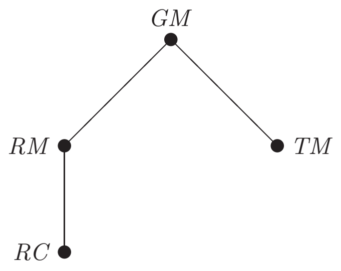
(c_7) At least three roles should perform the workflow [5].[4]

Note that a constraint may apply to one task (*e.g.*, c_1), two tasks (*e.g.*, c_2) or a set of tasks (typically this will be T, as in c_7). Constraints are *enforced* in a workflow instance. A constraint that applies to a single task will be enforced at that task. A constraint involving two tasks $\mathtt{t}'$ and $\mathtt{t}$, where $\mathtt{t}' < \mathtt{t}$, is enforced at $\mathtt{t}$ since it is impossible to enforce it unless we know which user or role has performed $\mathtt{t}'$. A constraint that applies to a set of tasks $\mathsf{T}' \subseteq \mathsf{T}$ will be enforced at each of the tasks in T'. We will formally define what it means for a workflow instance satisfy a constraint in Section 5.

[4]The original constraint in [5] is "At least three roles must be associated with the workflow", which is taken as static constraints in their terminology. This constraint just checks if $|R| > 3$. We change it "at least three roles should perform the workflow" which is a hybrid constraint since we are only interested in such constraints.



(a) Workflow specification



(b) Role hierarchy

Task	Role
$\mathtt{t}_1$	RC
$\mathtt{t}_2$	RM
$\mathtt{t}_3$	RM
$\mathtt{t}_4$	RC

(c) Task-role assignment relation PA

User	Role
`Alice`	RC
`Bob`	RM
`Carol`	RM
`Dave`	RC
`Eve`	GM
`Fred`	TM

(d) User-role assignment relation UA

Figure 1. A workflow authorization schema

4 Constraint specification

We will refer to constraints that apply to two tasks as *entailment constraints* in this paper. Informally, an entailment constraint requires that the execution of $\mathtt{t}$ is constrained in some way by the execution of $\mathtt{t}'$ (where $\mathtt{t}' < \mathtt{t}$): separation of duties and binding of duties are examples of entailment constraints.

The constraints relevant to one task or a set of tasks are actually *cardinality constraints* which impose restrictions on the number of users or roles required to execute a task or a set of tasks. We define two types of cardinality constraints that differ in their scope: a *local cardinality constraint* specifies the restrictions on different task instances of a given task in the same workflow instance; a *global cardinality constraint* specifies restriction on the task instances of a set of tasks in a workflow instance. We do not consider inter-case (workflow instance) constraints [8, 13] in this paper; they will be subject of future work.

Cardinality constraints and entailment constraints are of particular interest when considering consistency because these constraints apply to multiple "elements": *e.g.*, multiple instances of the same task in a local cardinality constraint, or multiple tasks in global cardinality constraints and entailment constraints. We note that inconsistency is more likely to arise when a constraint applies to several "elements" rather than a single element (as in time-dependent constraints [8, 13], for example).

4.1 Entailment constraints

If the execution of a task t is constrained by the execution of another task t', where $\mathsf{t}' < \mathsf{t}$, then we say that t is an *entailed* task of t'.

An entailment constraint defined on t has the form $(\mathbf{ur}, \mathsf{t}', E, pred)$, where $\mathbf{ur} \in \{\mathbf{u}, \mathbf{r}\}$, $E \subseteq U_{\mathsf{t}'}$ if $\mathbf{ur} = \mathbf{u}$, $E \subseteq R_{\mathsf{t}'}$ if $\mathbf{ur} = \mathbf{r}$ and $pred$ is a binary predicate such as $\neq$ or $\leqslant$. The semantics of the constraint $(\mathbf{ur}, \mathsf{t}', E, pred)$ are that if an actor $a \in E$ (either a role or a user) performed t' then t must be performed by some b such that $pred(a, b)$ is true.

Let $c = (\mathbf{r}, \mathsf{t}', E, pred)$ be a role-based entailment constraint. We will assume that $pred$ can be evaluated by considering the structure of the role hierarchy. In other words, $pred \in \{=, \neq, <, \leqslant, >, \geqslant\}$. Actually, there are subtleties in the interpretation of role-based entailment constraints. In c_4, for example, which requires that t_2 must be performed by a role that is more senior than the role that performed t_1 [5], it is not clear whether these two tasks can be performed by the same user given this user has two roles and one role is higher than another one. In order to provide strict security guarantee, in the role case, when the constraints require two different roles for two tasks $pred \in \{\neq, <, \leqslant, >, \geqslant\}$, it implicitly means two different users will be required, but if $pred$ is '=' then this restriction is not necessary.

In RBAC96 (and most other RBAC models) it is assumed that if $(p, r) \in PA$ then p is implicitly assigned to all roles in that dominating r. This fact means that the number of permission-role assignments can be significantly reduced – an attractive feature of the RBAC paradigm. However, this is problematic if entailment constraints with $pred \in \{=, \neq, >, \geqslant\}$. For example, if $pred$ is '=', then the constraint requires the two tasks be executed by the same role. If the traditional RBAC model is used in which permission inheritance is upwards, then the constraint will lose its real meaning. One obvious way is to prohibit the permission-role assignment inheritance when the constraints are applied.

4.2 Cardinality constraints

A set of authorized roles is defined for each task in a workflow and each authorized role is thereby authorized to perform that task. Thus it is meaningless to define role-based local cardinality constraints. Therefore we are only interested in the user-based local cardinality constraints. A local cardinality constraint is a pair (k, n_u), where $k \in \mathbb{N}$ indicates the number of executions (instances) in the current workflow instance[5] and n_u indicates the number of different users that must perform these k instances. In this paper, we assume that $n_u = 1$ or $n_u = k$. For example, $(k, 1)$ means that each instance of the task is performed by the same user (*i.e.*, binding of duties) and (k, k) means that each instance of the task is performed by a different user (*i.e.*, separation of duty). If $n_u = 1$, it implicitly means this user should use the same role to perform the task instances. Clearly the use of such constraints may mean that certain workflow instances cannot be completed. The constraint (k, k), for example, will prevent a workflow instance from completing if fewer than k users are authorized to perform t. We will discuss this problem in Section 5.

The need for global cardinality constraints is not so obvious. We note that tasks are assigned to roles and that a maximal role will typically be authorized to perform most, if not all, the tasks in a workflow. However, we may wish that some subset of tasks in the workflow is executed by a number of roles greater than some threshold value in order to provide a form of *relaxed separation of duty*. Like role-based entailment constraints, we assume that different roles implies different users. We will not consider binding of duties in a global cardinality constraint because such a constraint can be specified by a set of entailment constraints.

A global cardinality constraint is specified as a pair (T', n_r), where $\mathsf{T}' \subseteq \mathsf{T}$ and n_r indicates the minimum number of different roles that must execute the tasks in T'. If $\mathsf{t} \in \mathsf{T}'$, then we call t a *restricted task* of the global cardinality constraint. We assume that $|\mathsf{T}'| > 2$ and $n_r > 1$: if $|\mathsf{T}'| = 2$, then the cardinality constraint can be specified as an entailment constraint; if $n_r = 1$, then it can be defined as a set of entailment constraints.

[5]k is redundant since it has the same meaning as $\rho(\mathsf{t})$, but it is included in the constraint for ease of reference.

4.3 Workflow authorization with constraints

In Section 3 we defined a *workflow authorization schema* to be a pair $(\mathsf{S}, PA_{\mathsf{T}})$, where $\mathsf{S} = (\mathsf{T}, \leqslant, \rho)$ and $PA_{\mathsf{T}} \subseteq \mathsf{T} \times R$ is the permission-role assignment relation for the workflow specification. A *constrained workflow authorization schema* is a triple $(\mathsf{S}, PA_{\mathsf{T}}, PC_{\mathsf{T}})$, where $PC_{\mathsf{T}} \subseteq C \times \mathsf{T}$ and C is the set of constraints defined on T. $C = C_l \cup C_g \cup C_e$, where C_l denotes the set of local cardinality constraints, C_g denotes the set of global cardinality constraints and C_e denotes the set of entailment constraints.

We say that t is *assigned* to c if $(c, \mathsf{t}) \in PC_{\mathsf{T}}$ and (c, t) is a *constraint-task pair* if $(c, \mathsf{t}) \in PC_{\mathsf{T}}$. We will write $C(\mathsf{t})$ for the set of constraints to which t is explicitly assigned. We will use $C_l(\mathsf{t})$, $C_g(\mathsf{t})$ and $C_e(\mathsf{t})$ to denote the set of local cardinality, global cardinality and entailment constraints t is assigned to, respectively. We assume that $|C_l(\mathsf{t})| \leqslant 1$ and $|C_g(\mathsf{t})| \leqslant 1$. In other words, a task can be assigned to at most one local cardinality constraint and at most one global cardinality constraint. There is no restriction on the number of entailment constraints that can be assigned to a task.

The table below shows the constraints for our tax refund example and their association with tasks in the workflow specification.

Constraint	**Task**
$c_1 = (2, 2)$	t_2
$c_2 = (\mathbf{u}, \mathsf{t}_2, U_{\mathsf{t}_2}, \neq)$	t_3
$c_3 = (\mathbf{u}, \mathsf{t}_1, U_{\mathsf{t}_1}, \neq)$	t_4
$c_4' = (\mathbf{r}, \mathsf{t}_1, R_{\mathsf{t}_1}, <)$	t_2
$c_4'' = (\mathbf{r}, \mathsf{t}_1, GM, =)$	t_3
$c_5 = (\mathbf{u}, \mathsf{t}_1, U_{\mathsf{t}_1}, \neq)$	t_2
$c_6 = (\mathbf{u}, \mathsf{t}_1, \{Alice\}, notrelated)$	t_4
$c_7 = (\mathsf{T}, 3)$	t_1
$c_7 = (\mathsf{T}, 3)$	t_2
$c_7 = (\mathsf{T}, 3)$	t_3
$c_7 = (\mathsf{T}, 3)$	t_4

Constraints like c_6 may be used to prevent collusion between family members [2, 5]. However, the enforcement of constraint c_6 requires the the existence of some kind of information store and engine for evaluating such predicates. The analysis of such constraints is beyond the scope of this paper. Henceforth we will assume that the predicate in a user-based entailment constraint belongs to the set $\{=, \neq\}$.

5 Consistency

For any $\mathsf{t} \in \mathsf{T}$, we have $R(\mathsf{t})$, $U(\mathsf{t})$, $C_l(\mathsf{t})$, $C_g(\mathsf{t})$ and $C_e(\mathsf{t})$. In order to make the constraints *consistent* – that is, for all $u \in U(\mathsf{t})$ and $r \in R(\mathsf{t})$, there is at least one workflow instance that satisfies all of them – we have to ensure that two definitions are satisfied: constraint-task pair self-consistency and constraint-task pairs inter-play-consistency.

5.1 Consistency analysis

We defined a simple constraint specification language in Section 4, but each element in the specification should be defined accurately in order to avoid incorrectness and ambiguities. In addition, the constraint specification should be consistent with the authorization schema.

Let $c = (\mathbf{r}, \mathsf{t}', E, pred)$, $(c, \mathsf{t}) \in PC_{\mathsf{T}}$ and $r' \in E$. Then the set of (t', r', c)-*constrained roles* authorized to perform t, denoted $R(\mathsf{t}|\mathsf{t}', r', c)$, is given by

$$\{r \in R(\mathsf{t}) : pred(r', r) = \mathtt{true}\}.$$

If the execution of t is not constrained by the execution of t', then we adopt the notation $R(\mathsf{t}|\mathsf{t}', r', \mathtt{null}) = R(\mathsf{t})$ for completeness.

Similarly, let $c = (\mathbf{u}, \mathsf{t}', E, pred)$, $(c, \mathsf{t}) \in PC_{\mathsf{T}}$ and $u' \in E$. Then the set of (t', u', c)-*constrained users* authorized to perform t, denoted $U(\mathsf{t}|\mathsf{t}', u', c)$, is given by

$$\{u \in U(\mathsf{t}) : pred(u', u) = \mathtt{true}\};$$

we write $U(\mathsf{t}|\mathsf{t}', r', \mathtt{null}) = U(\mathsf{t})$ when t is not constrained by t'.

Definition 4 *An entailment constraint $c = (\mathbf{ur}, \mathsf{t}', E, pred)$ such that $(c, \mathsf{t}) \in PC_{\mathsf{T}}$ is* well-formed *if $\mathsf{t}' < \mathsf{t}$ and either*

$$\mathbf{ur} = \mathbf{u},\ pred \in \{=, \neq\} \text{ and } E \subseteq U_{t'},\ \text{or} \quad (1)$$

$$\mathbf{ur} = \mathbf{r},\ pred \in \{=, \neq, <, \leqslant, >, \geqslant\} \text{ and } E \subseteq R_{t'}. \quad (2)$$

A local cardinality constraint $c = (k, n_u)$ such that $(c, \mathsf{t}) \in PC_{\mathsf{T}}$ is well-formed *if*

$$k = \rho(\mathsf{t}); \quad (3)$$

$$n_u = 1 \text{ or } n_u = k. \quad (4)$$

A global cardinality constraint $c = (\mathsf{T}', n_r)$ such that $(c, \mathsf{t}) \in PC_{\mathsf{T}}$ is well-formed *if*

$$|\mathsf{T}'| \geqslant 3; \quad (5)$$

$$\mathsf{t} \in \mathsf{T}'; \quad (6)$$

$$n_r > 1. \quad (7)$$

That all the constraint-task pairs are well-formed cannot guarantee there is a workflow execution that satisfies all constraint-task pairs. We will define a *sound* constrained workflow authorization schema, which guarantees a successful workflow exists and for all $\mathsf{t} \in \mathsf{T}$, there is at least one successful workflow instance for any $u \in U(\mathsf{t})$ and $r \in R(\mathsf{t})$.

The constraints will make a workflow system very complicated, especially if we require this soundness property. In order to simplify the analysis we make a number of assumptions.

Assumption 1 *Let* $c_1 = (\mathbf{ur}_1, \mathsf{t}', E_1, pred_1)$ *and* $c_2 = (\mathbf{ur}_2, \mathsf{t}'', E_2, pred_2)$ *be two well-formed entailment constraints such that* $(c_1, \mathsf{t}), (c_2, \mathsf{t}) \in PC_{\mathsf{T}}$. *Then*

$$\mathsf{t}' = \mathsf{t}'' \tag{8}$$

$$\mathbf{ur}_1 = \mathbf{ur}_2 \tag{9}$$

$$pred_1 \neq pred_2 \tag{10}$$

Firstly, we assume that a task t can be entailed by no more than one task t', which we denote by $\mathbf{e}(\mathsf{t})$. Secondly, if two or more entailment constraints containing the same task t' are associated with the same task t, then they are all either user-based or role-based constraints. Finally, we assume the predicates are different. A consequence of this assumption is that there are at most two user-based entailment constraints and at most six role-based entailment constraints that can be associated with a task.

Assumption 2 *Let* $C_e(\mathsf{t}) = \{c_1, \dots, c_k\}$, *where* $c_i = (\mathbf{r}, \mathsf{t}', E_i, pred_i)$, $1 \leqslant k \leqslant 6$, *is a well-formed entailment constraint. Then either*

$$E_1 = R(\mathsf{t}') \text{ and } E_i \cap E_j = \emptyset \ (1 < i < j \leqslant k) \text{ or} \tag{11}$$

$$E_1 \subset R(\mathsf{t}') \text{ and } E_i \cap E_j = \emptyset \ (1 \leqslant i < j \leqslant k) \tag{12}$$

Let $C_e(\mathsf{t}) = \{c_1, \dots, c_k\}$, *where* $c_i = (\mathbf{u}, \mathsf{t}', E_i, pred_i)$, *(*$1 \leqslant k \leqslant 2$*) is a well-formed entailment constraint. Then either*

$$E_1 = U(\mathsf{t}') \text{ and } E_i \neq U(\mathsf{t}') \ (1 < i \leqslant k) \text{ or} \tag{13}$$

$$E_1 \subset U(\mathsf{t}') \text{ and } E_i \cap E_j = \emptyset \ (1 \leqslant i < j \leqslant k) \tag{14}$$

In other words, we assume that there is at most one entailment constraint whose domain is equal to the set of authorized roles or users and if there is such a constraint, we label it c_1 for convenience. We also assume that all other pairs of constraint domains are pairwise disjoint.

Given $\alpha \in R(\mathsf{t}')$, if $c_1 = (\mathbf{r}, \mathsf{t}', R(\mathsf{t}'), pred_1)$ and $\alpha \in E_i$ for some other constraint c_i, then we must decide which of c_1 and c_i should be applied. Similar considerations apply to user-based entailment constraints.

Definition 5 *Let* $C_e(\mathsf{t}) = \{c_1, \dots, c_k\}$, *where* $c_i = (\mathbf{ur}, E_i, \mathsf{t}', pred_i)$. *If* α *executes* t' *in a workflow instance* I, *the* enforced entailment constraint *for* α, *denoted* $AC(\alpha, \mathsf{t}')$, *is defined as follows:*

$$AC(\alpha, \mathsf{t}') = \begin{cases} c_i & \text{if } \alpha \in E_i \text{ and } i > 1, \\ c_1 & \text{if } \alpha \in E_1, \\ \texttt{null} & \text{otherwise}. \end{cases}$$

In other words, we enforce the constraint that contains the smallest domain to which α belongs.

Assumption 3 *If* $((k, n), \mathsf{t}'), ((\mathbf{ur}, \mathsf{t}', E, pred), \mathsf{t}) \in PC_{\mathsf{T}}$, *then either* $n = 1$ *or* $C_e(\mathsf{t}) = \{(\mathbf{u}, \mathsf{t}', E, \neq)\}$.

The reason for this assumption is that if t' is executed multiple times, then the execution of t is restricted by those of t' and ambiguities may arise. However, if $n_u = 1$, then the multiple instances for t' have the same user acting in the same role, so they can be considered to be a single instance thereby avoiding ambiguity. Similarly, if $n_u > 1$ and there is a single user-based separation of duty constraint in $C_e(\mathsf{t})$ then no ambiguities can arise because t should be performed by a user who is different from all the users that performed instances of t'.

Assumption 4 *For any global cardinality constraint* (T', n_r) *and for all* $\mathsf{t} \in \mathsf{T}'$,

- *if* $((k, n), \mathsf{t}) \in PC_{\mathsf{T}}$, *then* $n = 1$*;*
- *if* $((\mathbf{r}, \mathsf{t}', E, pred), \mathsf{t}) \in PC_{\mathsf{T}}$, *then* $\mathsf{t}' \in \mathsf{T}'$*;*
- *if* $((\mathbf{u}, \mathsf{t}', E, pred), \mathsf{t}) \in PC_{\mathsf{T}}$, *then* $\mathsf{t}' \in \mathsf{T}'$ *and* $C_e(\mathsf{t}) = \{(\mathbf{u}, \mathsf{t}', E, \neq)\}$.

In other words, if a restricted task t is assigned to a local cardinality constraint, then every instance of t has to be performed by the same user. A global cardinality constraint requires that a set of tasks is executed by a number of different roles, thus ambiguities could arise in the interpretation of the global cardinality constraint if $n \neq 1$. This assumption further suggests that if t has role-based entailment constraints, then $\mathbf{e}(\mathsf{t}) = \mathsf{t}'$ should be one of the restricted tasks. The reason is that the role that executes t is restricted by the role executing $\mathbf{e}(\mathsf{t})$, thus the legitimate role path should correspond to the entailment constraints of t. In other words, including $\mathbf{e}(\mathsf{t})$ as a restricted task will simplify the computation of a legitimate role path. Note that if $\mathbf{e}(\mathsf{t})$ is one of the restricted tasks, then t is not required to be a restricted task. Finally, if t has user-based entailment constraints, then $\mathbf{e}(\mathsf{t})$ must also be a restricted task and t must have a single separation of duty constraint associated with it. We do not allow entailment constraints in which two restricted tasks are performed by the same user because such tasks would implicitly be performed by the same role and complicate the analysis of global cardinality constraints.

Note that c_7 in our example in section 2 does not satisfy this assumption since $((2, 2), \mathsf{t}_2) \in PC_{\mathsf{T}}$. However, we can simply replace c_7 with $c'_7 = (\{\mathsf{t}_1, \mathsf{t}_3, \mathsf{t}_4\}, 2)$.

Figure 2 shows an algorithm Θ that will be used to analyze the consistency of a global constraint (T', n_r), where $\mathsf{T}' = \{\mathsf{t}'_0, \dots, \mathsf{t}'_{n-1}\}$. Θ is a similar to the *role planner* algorithm used by Bertino *et al.* [5]. The role planner algorithm computes legitimate role paths for all tasks in a workflow, whereas Θ only computes legitimate role paths corresponding to the restricted tasks in T'.

A *role path* GP is a list of length $l = |\mathsf{T}'|$, where the ith element in the list, denoted $GP[i]$, is a role authorized to perform task t_i' that is consistent with the constraints applied to t_i. A *legitimate role path* LP is a role path that contains at least n_r distinct roles. A role r has a legitimate role path LP at position i (or task t_i) if $r = LP[i]$. A set of roles $\{r_1', \ldots, r_x'\}$ $(1 < x \leqslant l)$ are on a legitimate role path if there exists LP such that $LP[i] = r_j'$ $(j = 1, \ldots, x)$, assume r_j' corresponding to the ith position in the role path. Θ checks for each $r \in R(\mathsf{t}_i')$ that there exists at least one legitimate role path, otherwise this role will never satisfy this global cardinality constraint. Note the complexity of Θ is exponential in the number of tasks in T'. More precisely, the overall worst-case complexity is $\mathcal{O}(N_R^n)$, where N_R is the maximum number of roles associated with any task in T' and $n = |\mathsf{T}'|$. Bertino *et al.* justify why their role planner algorithm is feasible in practice. A similar argument applies to Θ.

Definition 6 *Let* $\mathsf{t}, \mathsf{t}' \in \mathsf{T}$ *with* $\mathsf{t}' \leqslant \mathsf{t}$, *and let* $I = [(\mathsf{t}_1, u_1, r_1), \ldots, (\mathsf{t}_n, u_n, r_n)] \in Post(\mathsf{t})$. *We say* I satisfies *the workflow constraints* $C(\mathsf{t})$ *if all the following conditions hold:*

1. $u_i \in U(\mathsf{t})$ *and* $r_i \in R(\mathsf{t})$ *if* $\mathsf{t}_i = \mathsf{t}$*;*
2. *if* $((k, n), \mathsf{t}) \in PC_{\mathsf{T}}$ *then there are* k *occurrences of* t *in* I *and they are performed by* n *different users;*
3. *if there is more than one occurrence of* $\mathbf{e}(\mathsf{t})$ *in* I *then* $u_i \neq u_j$ *for all* u_i *and* u_j *where* $\mathsf{t}_i = \mathsf{t}$ *and* $\mathsf{t}_j = \mathbf{e}(\mathsf{t})$*;*
4. *if there is only one occurrence of* $\mathbf{e}(\mathsf{t})$ *in* I*, and assume* $\mathsf{t}_i = \mathbf{e}(\mathsf{t})$*:*
   - *if* $C_e(\mathsf{t})$ *is in user case and* $AC(u_i, \mathbf{e}(\mathsf{t})) = (\mathbf{u}, \mathbf{e}(\mathsf{t}), E, pred)$*, then* $pred(u_i, u)$ *evaluates to true for all* $(\mathsf{t}, u, r) \in I$*;*
   - *if* $C_e(\mathsf{t})$ *is in role case and* $AC(r_i, \mathbf{e}(\mathsf{t})) = (\mathbf{r}, \mathbf{e}(\mathsf{t}), E, pred)$*, then* $pred(r_i, r)$ *evaluates to true for all* $(\mathsf{t}, u, r) \in I$*;*
5. *if* $((\mathsf{T}', n), \mathsf{t}) \in PC_{\mathsf{T}}$ *then for any sublist* J *of* $Post(\mathsf{t})$ *containing only tasks in* T'*, the roles in* J *must be on some legitimate role path (determined by* Θ*) and if two different tasks in* J *are executed by different roles then they must be executed by different users.*

In definition 6, item 1 requires the task instance for t should be executed by authorized users and roles. Item 2 requires the task instances of t should satisfy a local cardinality constraint if there is one. Items 3 and 4 check that any entailment constraints are satisfied. Item 5 requires that if t has a global cardinality constraint, then t and other restricted tasks that have completed in this workflow instance, they should be in some legitimate role path. With the constraints defined in the workflow specification, some definitions in section 2 should be considered a little bit differently to take constraints into account. In other words, each task instance of t should satisfy $C(\mathsf{t})$ in any workflow instance.

Definition 7 *A constrained workflow authorization schema* $(\mathsf{T}, PA_{\mathsf{T}}, PC_{\mathsf{T}})$ *is* sound *if*

- *all* $(c, \mathsf{t}) \in PC_{\mathsf{T}}$ *is well-formed;*
- *for each* $u \in U(\mathsf{t})$*, there exists a complete workflow instance in which* t *completes with* u*.*
- *for each* $r \in R(\mathsf{t})$*, there exists a complete workflow instance in which* t *completes with* r*.*

Definition 8 *Two well-formed constraint-task pairs* (c_1, t) *and* (c_2, t') inter-play *if one of the following conditions hold:*

- $c_1, c_2 \in C_e$ *and* $\mathsf{t} = \mathsf{t}'$*;*
- $c_1, c_2 \in C_e$, $\mathsf{t} \neq \mathsf{t}'$ *and there exists* $(\mathsf{T}', n_r) \in C_g$ *such that* $\mathsf{t}, \mathsf{t}' \in \mathsf{T}'$*;*
- $c_1 \in C_e$, $c_2 = (k, n)$, $n > 1$ *and* $\mathsf{t} = \mathsf{t}'$*;*
- $c_1 \in C_e$, $c_2 = (k, n)$, $n > 1$ *and* $\mathsf{t}' = \mathbf{e}(\mathsf{t})$*;*
- $c_1 = (k_1, n_1)$, $c_2 = (k_2, n_2)$, $\mathsf{t}' = \mathbf{e}(\mathsf{t})$, $n_1 > 1$ *and* $n_2 > 1$*;*
- $c_1 \in C_e$, $c_2 \in C_g$, $\mathsf{t} = \mathsf{t}'$ *or* $\mathsf{t} \neq \mathsf{t}'$ *but both* t *and* t' *are restricted tasks of* c_2*;*
- $c_1 \in C_g$, $c_2 \in C_g$, $c_1 = c_2$ *and* $\mathsf{t} \neq \mathsf{t}'$.

In simple terms, two constraint-task pairs inter-play if the two tasks are equal, one is the entailed task of the other or they are restricted tasks in the same global constraint.

A set of well-formed constraint-task pairs $C_e(\mathsf{t}) \times \{\mathsf{t}\}$ is denoted by $CTE(\mathsf{t})$. We call a set of well-formed constraint-task pairs an *inter-play set* if each pair of constraints inter-play. An inter-play set is denoted by CT. Each CT has a set of *affected tasks* whose completion will be affected by the consistency of CT. We denote this set of tasks by $AT(CT)$. There are four possibilities for CT:

1. $CT = CTE(\mathsf{t})$ if $|CTE(\mathsf{t})| \geqslant 2$: $AT(CT) = \{\mathsf{t}\}$;
2. $CT = CTE(\mathsf{t}) \cup \{((k, n), \mathsf{t})\}$ if $|CTE(\mathsf{t})| \geqslant 1$ and $n > 1$: $AT(CT) = \{\mathsf{t}\}$;
3. $CT = \{((k', n'), \mathsf{t}'), (c_l, \mathsf{t}), ((\mathbf{u}, \mathsf{t}', E, \neq), \mathsf{t})\}$ if $n' > 1$ and ($c_l = \mathtt{null}$ or $c_l = (k, n)$): $AT(CT) = \{\mathsf{t}\}$;

Input: n is the number of input sets; n_r is the required number of different roles of a legitimate role path; $S_0, S_1, \ldots, S_{n-1}$ are sets to hold roles.

Output: `true` if for each role in S_i ($i = 0, \ldots, n-1$) there exists at least one legitimate role path; `false` otherwise.

```
  $RP$ = null /* $RP$ stores legitimate role paths */
  $role\_assignment(0, [\,])$  /* [] denotes the empty vector */
  for $i = 0$ to $n-1$
    for all $r \in S_i$
      if there does not exist $r \in RP$ such that $v.i = r$ then
        /* $v.i$ is the $(i+1)$th element of vector $v$ */
        return false
  return true

procedure
$role\_assignment(current, path\_hyp)$
  $j = current$
  repeat
    if $r \in S_j$ and $r$ is unmarked
      mark $r$
      $path\_hyp[j] = r$
    if $j < n-1$
      $role\_assignment(j+1, path\_hyp)$
    if $j = n-1$
      if the number of different roles in $path\_hyp \geqslant n$
        insert $path\_hyp$ into $RP$
  until each $r \in S_j$ has been marked
  unmark the roles in $S_j$
```

Figure 2. Algorithm Θ

4. $CT = \bigcup_{\mathsf{t}^* \in \mathsf{T}''} CTE(\mathsf{t}^*) \cup \bigcup_{\mathsf{t}' \in \mathsf{T}'} \{(c, \mathsf{t}')\}$ if $|CTE(\mathsf{t}^*)| \geqslant 1$, where $c \in C_g$, T' is the set of restricted tasks of c,$\mathsf{T}'' = \{\mathsf{t} \in \mathsf{T}' : \mathbf{e}(\mathsf{t}) \in \mathsf{T}'\}$: $AT(CT) = \mathsf{T}'$.

Given PC_T we can determine the set of all possible CT's, denoted PCT. We require for each $(c, \mathsf{t}) \in PC_\mathsf{T}$, if there exists $CT_1 \in PCT$, $(c, \mathsf{t}) \in CT_1$ and $\mathsf{t} \in AT(CT_1)$, then there does not exist $CT_2 \in PCT$ with $CT_1 \neq CT_2$, $(c, \mathsf{t}) \in CT_2$ and $\mathsf{t} \in AT(CT_2)$. In simple words, we require each inter-play set to be as big as possible. We use $SC = \{(c, \mathsf{t}) \in PC_\mathsf{T} : \nexists CT \in PCT, (c, \mathsf{t}) \in CT \text{ or } \exists CT \in PCT \text{ and } (c, \mathsf{t}) \in CT \text{ and } \mathsf{t} \notin AT(CT)\}$ to indicate a set of constraint-task pairs that are not in any inter-play set or they are in some inter-play set but t is not an affected task of this set.

Definition 9 *A well-formed constraint-task pair* $(c,\ \mathsf{t})$ *is* self-consistent *if* $(c,\ \mathsf{t}) \in SC$ *and satisfies one of the following cases:*

Case 1 *if* $c = (\mathbf{r}, \mathsf{t}', E, pred) \in C_e(\mathsf{t})$, *then for any* $\alpha \in E$, $R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) \neq \emptyset$ *and* $\bigcup_{\alpha \in R(\mathsf{t}')} R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) = R(\mathsf{t})$*;*

Case 2 *if* $c = (\mathbf{u}, \mathsf{t}', E, pred) \in C_e(\mathsf{t})$, *then for any* $\alpha \in E$, $U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) \neq \emptyset$ *and* $\bigcup_{\alpha \in U(\mathsf{t}')} U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) = U(\mathsf{t})$*;*

Case 3 *if* $c = (k, n_u) \in C_l(\mathsf{t})$, *then* $n_u \leqslant |U(\mathsf{t})|$*;*

Case 4 *if* $c = (\mathsf{T}', n_r) \in C_g(\mathsf{t})$, *then* $\Theta(|\mathsf{T}'|, n_r, R_{\mathsf{t}'_1}, \ldots, R_{\mathsf{t}'_{|\mathsf{T}'|}}) = \mathtt{true}$*;*

Note that if a global cardinality constraint c does not relate to another constraint, that is, for any restricted task t, it has no entailment constraints, then this global cardinality constraint is checked for self-consistency just once, even though it is assigned to each of the restricted tasks $\mathsf{t}_1, \ldots, \mathsf{t}_i$, and constraint-task pairs $(\mathsf{t}'_1, c), \ldots, (\mathsf{t}'_{|\mathsf{T}'|}, c)$ inter-play.

Definition 10 *An inter-play set* CT *is* inter-play-consistent *when:*

Case 1 *if* $CT = CTE(\mathsf{t})$ *and* $|CTE(\mathsf{t})| \geqslant 2$, *then*

(a) *if $C_e(\mathsf{t})$ is a set of role-based constraints, then for any $\alpha \in R(\mathsf{t}')$, $R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) \neq \emptyset$ and $\bigcup_{\alpha \in R(\mathsf{t}')} R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) = R(\mathsf{t})$;*

(b) *if $C_e(\mathsf{t})$ is a set of user-based constraints, then for any $\alpha \in U(\mathsf{t}')$, $U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) \neq \emptyset$ and $\bigcup_{\alpha \in U(\mathsf{t}')} U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) = U(\mathsf{t})$;*

Case 2 *if $CT = CTE(\mathsf{t}) \cup \{((k, n), \mathsf{t})\}$, $n > 1$ and $|CTE(\mathsf{t})| \geqslant 1$, then*

(a) *if $|CTE(\mathsf{t})| = 1$, then for $(c', \mathsf{t}) \in CTE(\mathsf{t})$, (c', t) is self-consistent;*

(b) *if $|CTE(\mathsf{t})| > 1$, then $CTE(\mathsf{t})$ is inter-play-consistent;*

(c) *if $C_e(\mathsf{t})$ is in user case, then for any $\alpha \in U(\mathsf{t}')$, $|U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))| \geq n$;*

(d) *if $C_e(\mathsf{t})$ is in role case, then for any $\alpha \in R(\mathsf{t}')$, $|U(R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')))| \geq n$;*

Case 3 *if $CT = \{((k', n'), \mathsf{t}'), (c_l, \mathsf{t}), ((\mathbf{u}, \mathsf{t}', E, \neq), \mathsf{t})\}$ then*

(a) *if $c_l = \texttt{null}$ and $|U(\mathsf{t}) \cap U(\mathsf{t}')| \leq n'$, then $|U(\mathsf{t})| > |U(\mathsf{t}) \cap U(\mathsf{t}')|$;*

(b) *if $c_l \neq \texttt{null}$, $c_l = (k, n)$ and $|U(\mathsf{t}) \cap U(\mathsf{t}')| \geqslant n'$, then $|U(\mathsf{t})| \geqslant n + n'$;*

(c) *if $c_l \neq \texttt{null}$, $c_l = (k, n)$ and $|U(\mathsf{t}) \cap U(\mathsf{t}')| < n'$, then $|U(\mathsf{t})| \geqslant |U(\mathsf{t}) \cap U(\mathsf{t}')| + n$;*

Case 4 *if $CT = \bigcup_{\mathsf{t}^* \in \mathsf{T}''} CTE(\mathsf{t}^*) \cup \bigcup_{\mathsf{t}' \in \mathsf{T}'} \{(c, \mathsf{t}')\}$ given $|CTE(\mathsf{t}^*)| \geqslant 1$, where $c \in C_g$, T' is the set of restricted tasks of c and $\mathsf{T}'' = \{\mathsf{t} \in \mathsf{T}' : \mathbf{e}(\mathsf{t}) \in \mathsf{T}'\}$, then*

(a) *if $|CTE(\mathsf{t}^*)| = 1$, then for $(c', \mathsf{t}^*) \in CTE(\mathsf{t})$, (c', t^*) is self-consistent;*

(b) *if $|CTE(\mathsf{t}^*)| > 1$, then $CTE(\mathsf{t}^*)$ is inter-play-consistent;*

(c) *$\Theta'(|\mathsf{T}'|, n_r, R'_{\mathsf{t}'_1}, \ldots, R'_{\mathsf{t}'_{|\mathsf{T}'|}}, \mathbf{S}) = \texttt{true}$ with $\mathbf{S} = \bigcup_{\mathsf{t}^* \in \mathsf{T}''} CTE(\mathsf{t}^*)$;*

Again, the basic intuition behind inter-play-consistency is that a workflow instance in which all the constraints in a set of inter-playing constraint-task pairs are satisfied can always be completed. Algorithm Θ' is an variant of Θ that takes $\mathbf{S}$ as a parameter. $\mathbf{S}$ is the union of entailment constraint-task pairs for the restricted tasks. Since for the restricted tasks, if they have entailment constraints, then the selection of roles in a legitimate role path should correspond to these constraints, so we include $\mathbf{S}$ in Θ' for checking. Thus, before the role path is inserted into RP, besides checking the number of distinct roles, $\mathbf{S}$ should be checked for it to see if the entailment constraints are satisfied in terms of the relevant tasks: if the entailment constraint for a task $\mathsf{t} \in \mathsf{T}'$ is in role case, the checking is straightforward; if the entailment constraint for t is in user case, it should check if t and $\mathbf{e}(\mathsf{t})$ has the same role in the role path and if this role only has one authorized user. (If this role does have only one authorized user, it obviously contradicts Assumption 4, which requires that the user-based entailment constraint for t can only has $pred = \neq$).

Theorem 1 *A constrained workflow authorization schema $((\mathsf{T}, \leqslant, \rho), PA_{\mathsf{T}}, PC_{\mathsf{T}})$ is sound if and only if all constraint-task pairs in SC are self-consistent and all the inter-play sets are inter-play-consistent.*

In order to prove this theorem, we need to prove two lemmas.

Lemma 1 *Let $\mathsf{t} \in \mathsf{T}$ be a task such that for all $\mathsf{t}' < \mathsf{t}$, t' has completed. If for all $(c, \mathsf{t}) \in PC_{\mathsf{T}}$, either $(c, \mathsf{t}) \in SC$ and (c, t) is self-consistent or $(c, \mathsf{t}) \in CT$, and CT is inter-play-consistent, then*

- *for any authorized user $u \in U(\mathsf{t})$, there is some workflow instance I which completes at t with u;*
- *for any authorized role $r \in R(\mathsf{t})$, there is some workflow instance I which completes at t with r.*

A consequence of this lemma is that for any $u \in U(\mathsf{t})$ acting in role $r \in R(\mathsf{t})$, there exists some $I \in Pre(\mathsf{t})$ such that $[I, (\mathsf{t}, u, r)] \in Post(\mathsf{t})$ or $[I, (\mathsf{t}, u, r)] \in In(\mathsf{t})$. Furthermore, if $I \in In(\mathsf{t})$, then there exists a sequence of evolutions of I into I', where $I' \in Post(\mathsf{t})$. If for any $u \in U(\mathsf{t})$, there is some workflow instance I which completes at t with u, then a consequence following it is that any $r \in R(\mathsf{t})$, there is some workflow instance I which completes at t with r.

Lemma 2 *In a workflow instance I which completes at t with u or r, any $\mathsf{t}' > \mathsf{t}$ will complete, given I has completed at all $\mathsf{t}'' < \mathsf{t}$ and every constraint-task pairs in SC are self-consistent and all inter-play sets are inter-play-consistent.*

We refer the reader to Appendix for the formal proof of the theorem and two lemmas.

5.2 The consistency of the example

Now we can use the self-consistent and inter-play-consistent definition to check the consistency of our example in section 3. We first note that c_4 and c_5 are both assigned to t_2, but c_4 is a role-based entailment constraint and c_5 is user-based. This does not conform to our assumption that all the entailment constraints of a particular task should either be user-based or role-based. As a matter of fact, c_4 implies c_5 in our system because c_4 requires that task $\mathtt{t}_2$ should be executed by a role that dominates the role that executes $\mathtt{t}_1$, and two different roles executing two tasks implies two different users. Hence we can omit constraint c_5. We also find that some constraint-task pairs are not consistent. For example, $CT = \{(c_4', \mathtt{t}_2), (c_4'', \mathtt{t}_2), ((2,2), \mathtt{t}_2)\}$ inter-play, but it is not inter-play-consistent: if Bob, Carol or Fred executes $\mathtt{t}_1$, then the workflow instance can not complete at $\mathtt{t}_2$ because there are not two authorized users that can perform the two instances of $\mathtt{t}_2$. One solution to solve this inconsistency is to have more than three users assigned to GM. Another solution is to release the constraint c_4 but keep c_5, namely, task $\mathtt{t}_2$ is executed by any user who acts on GM or RM given this user does not execute $\mathtt{t}_1$.

5.3 Computational complexity

Of course, the soundness property of a constrained workflow authorization schema can be checked by exhaustively searching for legitimate role paths (corresponding to T) and user paths (similar to the definition of a role path), like the role and user planner algorithms proposed in [5].

These two algorithms are used to check consistency by assigning users and roles to tasks that constitute a workflow instance in such a way that no constraints are violated. The overall worst-case complexity of the role planner is $\mathcal{O}(N_R^n)$, where N_R is the the maximum number of roles associated with any task in the workflow and n is the number of tasks in the workflow. A rough user planner, that is for each role in a role path, assigning the users who act on this role so as to form a user path, has the worst complexity of $\mathcal{O}((N_R N_U N_{act})^n)$, where N_R is the maximum number of roles associated with any task in the workflow; N_U is the maximum number of users associated with any role in the workflow; N_{act} is the maximum number of activations associated with any task in the workflow.

Obviously, this user planner is likely to be too complicated to be useful in practice. The authors of [5] suggest a heuristic solution: for each role path obtained by the role planner, compute the maximum number of users required for each role r_i in the role path. This set is denoted as $U_{worstcase}(r_i)$. The worst case scenario arises when each task instance with r_i requires a different user in each such task instance, thus $U_{worstcase}(r_i) = \Sigma_{t_j \in TASK_{r_i}} act_j$ where $TASK_{r_i}$ indicates the set of tasks executed by r_i in this role path. The authors suggested to perform user planning for only those roles r_i such that $U_{worstcase}(r_i) + const_i \leqslant U_i$ where U_i is the set of users assigned to r_i; $const_i$ is a safety factor which can be decided by workflow designer. The overall worst-case complexity of this user planner is $\mathcal{O}((l \cdot N_U(l) \cdot N_{act}(l))^m)$, where l denotes the number of roles for which the user planning is done ($l = \epsilon_1 n N_R$, $0 < \epsilon_1 \leqslant 1$); $N_U(l)$ denotes the maximum number of users associated with any role for which the user planning is done; and m is the number of tasks in the workflow which is done user planning ($m = \epsilon_2 n$, $0 < \epsilon_2 \leqslant 1$).

In our system, we suppose there are n tasks in the workflow; N_R is the maximum number of roles associated with any task; N_U is the the maximum number of users associated with any task. We assume $N_U > N_R$. In consistency checking, many of the operations are relevant to sets of users or roles. In order to get efficient operations, we assign a different integer (starting from 0) to each role sequentially. We assume that each role set is sorted according to these integers using a sorting algorithms with linear time complexity, such as counting sort [9]. The same process should be applied to user sets.

If a constraint-task pair belongs to SC, then it should be checked for self-consistency as given by the four cases in definition 9. We now examine the time complexity of each of these cases.

Case 1 The complexity of computing $R(\mathtt{t}|\mathtt{t}', \alpha, c)$ is $\mathcal{O}(N_R)$ and there are at most N_R such choices of α, so the overall complexity is $\mathcal{O}(N_R^2)$.

Case 2 Similarly the complexity is $\mathcal{O}(N_U^2)$.

Case 3 The complexity is simply $\mathcal{O}(1)$.

Case 4 The complexity is $\mathcal{O}(N_R^l)$, where l is the number of restricted tasks.

We now examine the situation for an inter-play set CT.

Case 1 The worst case is subcase 2, which has complexity $\mathcal{O}(N_U^2)$.

Case 2 Checking the consistency of $CTE(\mathtt{t})$ is $\mathcal{O}(N_U^2)$; computing $U(\mathtt{t}|\mathtt{t}', \alpha, AC(\alpha, \mathtt{t}'))$ is $\mathcal{O}(N_U)$ and there are N_U such operations; computing $U(R(\mathtt{t}|\mathtt{t}', \alpha, AC(\alpha, \mathtt{t}')))$ is $\mathcal{O}(N_U N_R)$ and there are N_R such operations. Hence the total complexity in this case is $\mathcal{O}(N_U^2 + N_R^2 N_U)$.

Case 3 The complexity is $\mathcal{O}(N_U)$.

Case 4 Checking the consistency of $CTE(\mathtt{t})$ is $\mathcal{O}(N_U^2)$; computing Θ' is $\mathcal{O}(N_R^l)$. Hence the total complexity in this case is $\mathcal{O}(N_R^l + nN_U^2)$.

The complexity of computing SC and PCT is $\mathcal{O}(N_C)$, where N_C is the number of constraint-task pairs in the workflow specification. Therefore the complexity of consistency checking in our system is

$$\mathcal{O}(n(N_U^2) + n(N_R^2 N_U) + cN_R^l + N_C),$$

where c is a constant to indicate the number of global cardinality constraints in the workflow. (We have $0 \leqslant c \leqslant n/3$, since we require the number of restricted tasks in a global cardinality constraint should be at least 3.) It is easy to see the computation of our consistency checking is far more efficient than those role planner and user planner algorithms in [5].

6 Conclusion

We provide an approach to help workflow designers to define a sound constrained workflow authorization schema. We defined a set of consistency rules for constraints that guarantee there is no inconsistency and ambiguities within these constraints even when they inter-play with each other. When the constraints conform to these rules, then a sound constrained workflow authorization schema is guaranteed. We gave explicit definition of a workflow instance and a task instance. We defined a workflow authorization schema, which associates roles with tasks in the workflow specification, which is extended to a constrained workflow authorization schema. A constrained workflow authorization schema specifies the constraints that should be enforced on each instance of the workflow. We used a simple tuple-based method for constraint specification, making the expression and verification of authorization constraints easy for both system administrators and application writers. The specification language associates constraints with tasks rather than a workflow, thereby facilitating distributed authorization and enforcement of constraints.

Future Work We will extend our research in two directions: one is to extend our specification scheme to include inter-case constraints [8, 13]. Another is to define a more general and optimized framework for the consistency rules. Additionally, we hope to relax the restrictions that each task can only entail one task, and all the entailment constraints can only be either user-case or role-case.

Acknowledgement This work was supported by a gift from Microsoft University Relations, NSF grants CCR02-08996 and EIA00-88028, ONR grant N000014-02-1-0715, and ARO grant DAAD-19-01-1-0473.

References

[1] G.-J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and Systems Security*, 3(4):207–226, 2000.

[2] G.-J. Ahn, R. Sandhu, M. Kang, and J. Park. Injecting RBAC to secure a web-based workflow system. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pages 1–10, 2000.

[3] V. Atluri, S. Chun, and P. Mazzoleni. A Chinese wall security model for decentralized workflow systems. In *8th ACM Conference on Computer and Communication Security*, pages 48–57, 2001.

[4] V. Atluri and W. Huang. An authorization model for workflow. In *Proceedings of the Fifth Europe Symposium on Research in Computer Security*, pages 44–64. Lecture Notes in Computer Science, 1996.

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

[6] R. Botha and J. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–681, 2001.

[7] D. Brewer and M. Nash. The Chinese Wall security policy. In *Proceedings of 1989 IEEE Symposium on Security and Privacy*, pages 206–214, Oakland, California, 1989. IEEE Computer Society Press.

[8] F. Casati, S. Castano, and M. Fugini. Managing workflow authorization constraints through active database technology. *Information Systems Frontiers*, 3(3):319–338, 2001. Technical Report HPL-2000-156, Hewlett Packard Laboratories.

[9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2002.

[10] J. Crampton. Specifying and enforcing constraints in role-based access control. In *Proceedings of the Eighth ACM symposium on Access control models and Technologies*, pages 43–50, 2003.

[11] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modelling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[12] V. Gligor, S. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of 1998 IEEE Symposium on Research in Security and Privacy*, pages 172–183, Oakland, California, 1998.

[13] W. Huang and V. Atluri. Secureflow: A secure web-enabled workflow management system. In *ACM Workshop on Role-Based Access Control*, pages 83–94, 1999.

[14] M. Kang, J. Park, and J. Froscher. Access control mechanisms for inter-organizational workflow. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*, pages 66–74, 2001.

[15] K. Knorr and H. Stormer. Modeling and analyzing separation of duties in workflow environment. In *Proceedings of 16th International Conference on Information Security*, pages 199–212, 2001.

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

[17] R. Simon and M. Zurko. Separation of duty in role-based environments. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 183–194, Rockport, Massachusetts, 1997.

[18] J. Wainer, P. Barthelmess, and A. Kumar. W-RBAC - A workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, 12(4):455–486, 2003.

Appendix: Proofs of results

Proof of Theorem 1: We need to prove sufficiency and necessity.

Sufficiency If $((\mathsf{T}, \leqslant, \rho), PA_{\mathsf{T}}, PC_{\mathsf{T}})$ is a *sound* constrained workflow authorization schema, then for any $(c, \mathsf{t}) \in SC$, it is self-consistent; for any $CT \in PCT$, it is inter-play-consistent.

Sufficiency can be proved with counter examples for each case. We suppose $((\mathsf{T}, \leqslant, \rho), PA_{\mathsf{T}}, PC_{\mathsf{T}})$ is sound, but $\exists c \in C$ which is not self-consistent or $\exists CT$ that CT inter-play but it is not inter-play-consistent.

Case 1 If a constraint-task pair (c, t) does not inter-play with any other constraint-task pair but is not self-consistent, then there are three subcases.

Subcase 1 $c = (\mathbf{ur}, \mathsf{t}', E, pred) \in C_e(\mathsf{t})$ gives rise to two further possibilities:

- if $\exists \alpha \in E$ but $R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) = \emptyset$, then if $I \in Post(\mathsf{t}')$ completes with α, I cannot evolve to a complete workflow instance because no one is able to execute t. If $\bigcup_{\alpha \in R(\mathsf{t}')} R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) \neq R(\mathsf{t})($ according to the definition of $R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))$, $\bigcup_{\alpha \in R(\mathsf{t}')} R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) \subseteq R(\mathsf{t})$), thus $\exists r$ such that $r \in R(\mathsf{t})$ is not in any $R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))$, so for any $I \in Post(\mathsf{t}')$, I will not evolve to a complete workflow instance so that t complete with r in it.
- if $\exists \alpha \in E$ but $U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) = \emptyset$, then if $I \in Post(\mathsf{t}')$ completes with α, I cannot evolve to a complete workflow instance because no one is able to execute t. If $\bigcup_{\alpha \in U(\mathsf{t}')} U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) \neq U(\mathsf{t})($ according to the definition of $U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))$, $\bigcup_{\alpha \in U(\mathsf{t}')} U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')) \subseteq U(\mathsf{t})$), thus $\exists u$ such that $u \in U(\mathsf{t})$ is not in any $U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))$, so for any $I \in Post(\mathsf{t}')$, I will not evolve to a complete workflow instance so that t completes with u.

Subcase 2 $c = (k, n_u) \in C_l(\mathsf{t})$:

- if $n_u > |U(\mathsf{t})|$, then there are not enough authorized users to execute t and for any $I \in Pre(\mathsf{t})$, I cannot evolve to a I' such that $I' \in Post(\mathsf{t})$.

Subcase 3 $c = (\mathsf{T}', n_r) \in C_g(\mathsf{t})$:

- if $\Theta(|\mathsf{T}'|, n_r, R_{\mathsf{t}'_1}, \ldots, R_{\mathsf{t}'_{|\mathsf{T}'|}}) =$ **False**, then for a role $r' \in R(\mathsf{t}'_i)$ at $\mathsf{t}'_i(i = 1, .., |\mathsf{T}'|)$, there is no legitimate role path, thus for any $I \in Pre(\mathsf{t})$, I cannot evolve to a complete workflow instance so that t complete with r'.

Case 2 If CT inter-play, but is not inter-play consistent, then there are four subcases.

Subcase 1 $CT = CTE(\mathsf{t})$ given $|CTE(\mathsf{t})| \geqslant 2$:

- The argument is the same as subcase 1 in case 1.

Subcase 2 $CT = CTE(\mathsf{t}) \cup \{((k, n), \mathsf{t})\}$ such that $n > 1$, given $|CTE(\mathsf{t})| \geq 1$:

- $C_e(\mathsf{t})$ is in user case: for any $\alpha \in U(\mathsf{t}')$, if $|U(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))| < n$, then for those $I \in Pre(\mathsf{t})$ such that t' completes with α, I cannot evolve to a I' such that $I' \in Post(\mathsf{t})$ since there are not enough authorized users to execute t .
- if $C_e(\mathsf{t})$ is in role case: for any $\alpha \in R(\mathsf{t}')$, $|U(R(\mathsf{t}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}')))| < n$, then for those $I \in Pre(\mathsf{t})$ such that t' completes with α, I cannot evolve to a I' such that $I' \in Post(\mathsf{t})$ since there are not enough authorized users to execute t .

Subcase 3 $CT = \{(c'_l, \mathsf{t}'), (c_l, \mathsf{t}), (c_e, \mathsf{t})\}$, $c_e = (\mathbf{u}, \mathsf{t}', E, \neq)$, $c'_l = (k', n')$:

- when $c_l = \mathtt{null}$: if $|U(\mathsf{t}) \cap U(\mathsf{t}')| \leq n'$, but $|U(\mathsf{t})| = |U(\mathsf{t}) \cap U(\mathsf{t}')|$, then for any $I \in Pre(\mathsf{t})$ such that t' completes with all the users in $U(\mathsf{t}) \cap U(\mathsf{t}')$, then I cannot evolve at t since there is no user can execute t;
- when $c_l \neq \mathtt{null}$ and $c_l = (k, n)$: if $|U(\mathsf{t}) \cap U(\mathsf{t}')| \geqslant n'$, but $|U(\mathsf{t})| < n + n'$, then for any $I \in Pre(\mathsf{t})$ such that t' completes with a set of users and each of them is in $U(\mathsf{t}) \cap U(\mathsf{t}')$, then I cannot evolve to I' such that $I' \in Post(\mathsf{t})$ since there are not enough users who can execute t;
- when $c_l \neq \mathtt{null}$ and $c_l = (k, n)$: if $|U(\mathsf{t}) \cap U(\mathsf{t}')| < n'$, but $|U(\mathsf{t})| < |U(\mathsf{t}) \cap U(\mathsf{t}')| + n$, then for any $I \in Pre(\mathsf{t})$ such that t' completes with a set of users and each of them is in $U(\mathsf{t}) \cap U(\mathsf{t}')$, then I cannot evolve to I' such that $I' \in Post(\mathsf{t})$ since there are not enough users who can execute t;

Subcase 4 $CT = \bigcup_{\mathsf{t}^* \in \mathsf{T}''} CTE(\mathsf{t}^*) \cup \bigcup_{\mathsf{t}' \in \mathsf{T}'} \{(c, \mathsf{t}')\}$ given $|CTE(\mathsf{t}^*)| \geqslant 1$, where $c \in C_g$, T' is the set of restricted tasks of c and $\mathsf{T}'' = \{\mathsf{t} \in \mathsf{T}' : \mathbf{e}(\mathsf{t}) \in \mathsf{T}'\}$:

- if $\Theta'(|\mathsf{T}'|, n_r, R'_{\mathsf{t}'_1}, \ldots, R'_{\mathsf{t}'_{|\mathsf{T}'|}}, \mathbf{S}) =$ **False**, then for a role $r' \in R(\mathsf{t}'_i)$ at $\mathsf{t}'_i(i = 1, .., |\mathsf{T}'|)$, there is no legitimate role path, thus for any $I \in Pre(\mathsf{t})$, I cannot evolve to a complete workflow instance so that t complete with r'.

Necessity If for any $(c, \mathsf{t}) \in SC$, (c, t) is self-consistent and for any $CT \in PCT$, CT is inter-play-consistent, then $((\mathsf{T}, \leqslant, \rho), PA_{\mathsf{T}}, PC_{\mathsf{T}})$ is a *sound* constrained workflow authorization schema.

Before prove it, we make an assumption.

Assumption 5 *For any $r, r' \in R$, if $r \neq r'$, then $U(r) \neq U(r')$; if $U(r) \cap U(r') \neq \emptyset$, then $|U(r)| > 1$ and $|U(r')| > 1$.*

It certainly has no need to prove the well-formed property of a sound constrained workflow authorization schema since self-consistency and inter-play-consistency implies all the constraint-task pairs are well-formed. So we need to prove for any user authorized in t, there exists a complete workflow instance so that t completes with u. If for any user u authorized in t, there exists a complete workflow instance so that t completes with u, then it is straightforward to get the solution that any role r which is authorized in t, there exists a complete workflow instance so that t completes with r.

To prove each user $u \in U(\mathsf{t})$, there exists a complete workflow instance so that t completes with u, we need to prove Lemma 1 and Lemma 2.

Proof of Lemma 1: The result is proved by induction. We first need to prove this lemma holds for t_1, the first task in T. There are three cases to consider.

Case 1 No constraint is assigned to it. Then there is no restriction regarding the choice an authorized user in any workflow instance, so for any authorized user u in t_1, $\exists I \in Post(\mathsf{t}_1)$ so t_1 completes with u in I.

Case 2 A local cardinality constraint is assigned to it. Self-consistency guarantees the workflow instance can complete at t_1 with a set of required different authorized users. For any user u authorized in t_1, he can be in such a set. Thus $\exists I \in Post(\mathsf{t}_1)$ so that t_1 completes with u in I

Case 3 A local cardinality and a global cardinality constraint or a global cardinality constraint is assigned to it. In case 3, according to assumption 4, we know that if a local cardinality constraint is assigned to t_1 together with a global cardinality constraint, it has form $(k, 1)$. So it will not inter-play with any other constraint. After all, it is like an empty constraint in terms of consistency problem. So we only need to consider the global cardinality constraint. Either the self-consistency case 4 or inter-play consistency case 4 guarantees that each role can be in some legitimate role path, so t_1 can complete with any role, further with any user.

We suppose the lemma holds for $\mathsf{t}_1, \mathsf{t}_2, \ldots, \mathsf{t}_i$ ($1 \leqslant i < n$), now we will prove it holds for t_{i+1} ($\mathsf{t}_i < \mathsf{t}_{i+1} \leqslant \mathsf{t}_n$). There are three cases to consider.

Case 1 If t_{i+1} has no constraints, then there is no restriction about the choose of an authorized user in any workflow instance, so for any authorized user u in t_{i+1}, for any $I \in Pre(\mathsf{t}_{i+1})$, I can evolve to I' so that t_{i+1} completes with u in I'.

Case 2 t_{i+1} is assigned a constraint c, $(c, \mathsf{t}_{i+1}) \in SC$, so (c, t_{i+1}) is self-consistent.

Subcase 1 If $c = \{(\mathbf{ur}, \mathsf{t}', E, pred)\} \in C_e(\mathsf{t}_{i+1})$: if $\mathbf{ur} = \mathbf{r}$, $\bigcup_{\alpha \in R(\mathsf{t}')} R(\mathsf{t}_{i+1} | \mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))) = R(\mathsf{t}_{i+1})$ guarantees that for any $r \in R(\mathsf{t}_{i+1})$, $\exists \alpha \in R(\mathsf{t}')$ such that $r \in R(\mathsf{t}_{i+1} | \mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))$. Since we suppose lemma 1 holds for t', thus $\exists I \in Post(\mathsf{t}')$ and t' completes with α. Assume I evolves to I' so that $I' \in Pre(\mathsf{t}_{i+1})$, then t can complete with r and all the users acting in r. But if $r \neq \alpha$, then the user that executes t' should be different from the user executing t_{i+1}. According to assumption 5, any user assigned to r can find a different user for α even if he is authorized to α too. So for any $u \in U(\mathsf{t}_{i+1})$ acting in role r, t_{i+1} can complete with u in a I' given I' is evolved from I, $I \in Post(\mathsf{t}')$ and t' completes with u' ($u \neq u'$) acting in role α. If $\mathbf{ur} = \mathbf{u}$, the proof is similar to the role case, but simpler.

Subcase 2 If $c = (k, n_u) \in C_l(\mathsf{t}_{i+1})$: $n_u \leqslant |U(\mathsf{t}_{i+1})|$ guarantees the workflow instance can complete at t_{i+1} with a set of required different authorized users. So for any authorized user u in t_{i+1}, $\exists I \in Post(\mathsf{t}_{i+1})$ so t_{i+1} completes with u in I.

Subcase 3 If $c = (\mathsf{T}', n_r) \in C_g(\mathsf{t}_{i+1})$, $\Theta(|\mathsf{T}'|, n_r, R_{\mathsf{t}'_1}, \ldots, R_{\mathsf{t}'_{|\mathsf{T}'|}}) =$ **True** guarantees that r is in at least one legitimate role path, thus some workflow instance can complete at t_{i+1} with r. In the global cardinality constraint, it implicitly requires each different role in the role path should be acted by a different user. In case a user acts on more than one roles and these roles are in the same role path, then this user can only perform one restricted task given other tasks that he is authorized too is performed by a different user (according to assumption 5, this user can always find such a set of different users to execute these tasks). So for any $u \in U(\mathsf{t}_{i+1})$ acting in r, $\exists I \in Pre(\mathsf{t}_{i+1})$, for any $\mathsf{t}' \in \mathsf{T}'$ and $\mathsf{t}' < \mathsf{t}_{i+1}$, t' completes in I with r' and u', all r''s are in some legitimate role path, $u' \neq u$ if $r' \neq r$, then I can evolve to I' such that $I' \in Post(\mathsf{t}_{i+1})$ and t complete with u in I'.

Case 3 for any $(c, \mathsf{t}_{i+1}) \in PC_{\mathsf{T}}$, $(c, \mathsf{t}_{i+1}) \in CT$ and CT inter-play-consistent:

Subcase 1 $CT = CTE(\mathsf{t}_{i+1})$:

- The proof is similar to case 2 subcase 1.

Subcase 2 $CT = CTE(\mathsf{t}_{i+1}) \cup \{((k, n_u), \mathsf{t}_{i+1}\}$:

- if $C_e(\mathsf{t}_{i+1})$ are in user case: for any $\alpha \in U(\mathsf{t}')$ where $\mathsf{t}' = \mathbf{e}(\mathsf{t}_{i+1})$, $|U(\mathsf{t}_{i+1}|\mathsf{t}', \alpha, AC(\alpha, \mathsf{t}'))| \geq n$ guarantees for any $I \in Post(\mathsf{t}')$ such that t' completes with α in I, I can evolve to I' such that $I' \in Post(\mathsf{t}_{i+1})$. For any $u \in U(\mathsf{t}_{i+1})$, u corresponds to some α by the inter-play-consistency or self-consistency of $CTE(\mathsf{t}_{i+1})$, thus t_{i+1} can complete with u in I'. If $C_e(\mathsf{t}_{i+1})$ are in role case, the proof is similar to user case. The consideration of two different roles executing t' and t_{i+1} is similar to case 2 subcase 1.

Subcase 3 $CT = \{((k', n'), \mathsf{t}'), (c_l, \mathsf{t}_{i+1}), ((\mathbf{u}, \mathsf{t}', E, \neq), \mathsf{t}_{i+1})\}$:

- if $c_l = \texttt{null}$: if $u \notin U(\mathsf{t}')$, then any $I \in pre(\mathsf{t}_{i+1})$ can evolve to I' such that $I' \in post(\mathsf{t}_{i+1})$ and t_{i+1} complete with u in I'; if $u \in U(\mathsf{t}')$, then some $I \in pre(\mathsf{t}_{i+1})$ can evolve to I' such that $I' \in post(\mathsf{t}_{i+1})$, u is different for any user who executes t' (this is guaranteed by inter-play-consistent case 3, that is for any $u \in U(\mathsf{t}_{i+1}) \cap U(\mathsf{t}')$, there is some set of users who are authorized to execute t', so that u is different from any of them);

- if $c_l \neq \texttt{null}$, the proof is similar to that for $c_l = \texttt{null}$.

Subcase 4 $CT = \bigcup_{\mathsf{t}^* \in \mathsf{T}''} CTE(\mathsf{t}^*) \cup \bigcup_{\mathsf{t}' \in \mathsf{T}'} \{(c, \mathsf{t}')\}$, $\mathsf{t}_{i+1} \in \mathsf{T}'$:

- Θ' guarantees that r is in at least one legitimate role path and this role path does not contradict to any entailment constraints assigned to those restricted tasks. The proof is similar to case 2, subcase 3.

□

Proof of Lemma 2: The proof for this lemma is straightforward. For any $I \in post(\mathsf{t}_{i+1})$, assume t_{i+1} completes with a set of users $\widehat{U}$ acting in a set of roles $\widehat{R}$ in I (since a task might be executed for several times, so there might be more than one user and role in $\widehat{U}$ and $\widehat{R}$). $\widehat{U}$ and $\widehat{R}$ will only affect some t' if t' is the entailed task of t_{i+1} or $\mathsf{t}' \in \mathsf{T}^*$ where $\mathsf{T}^* \subseteq \mathsf{T}'$ if $\mathsf{t}_{i+1} \in \mathsf{T}'$ (T' is a set of restricted tasks of a global cardinality constraint and $\mathsf{T}^* = \{\mathsf{t} \in \mathsf{T}' : \mathsf{t} > \mathsf{t}_{i+1}\}$). For any $(c, \mathsf{t}') \in PC_{\mathsf{T}}$, if $(c, \mathsf{t}') \in SC$, then the cases in self-consistency guarantee there must be some user who can perform t'; if $(c, \mathsf{t}') \notin SC$, then $\exists CT \in PCT$ such that $(c, \mathsf{t}') \in CT$, then all kinds of cases in inter-play-consistency can guarantee there must be some user who can perform t'.

□

Necessity follows from Lemmas 1 and 2. □