On Run-time Enforcement of Authorization Constraints in Security-Sensitive Workflows

Daniel Ricardo dos Santos and Silvio Ranise

Fondazione Bruno Kessler (FBK)

Abstract. In previous work, we showed how to use an SMT-based model checker to synthesize run-time enforcement mechanisms for business processes augmented with access control policies and authorization constraints, such as Separation of Duties. The synthesized enforcement mechanisms are able to guarantee both termination and compliance to security requirements, i.e. solving the run-time version of the Workflow Satisfiability Problem (WSP). No systematic approach to specify the various constraints considered in the WSP literature has been provided. In this paper, we first propose a classification of these constraints and then show how to encode them in the declarative input language of the SMT-based model checker used for synthesis. This shows the flexibility of the SMT approach to solve the run-time version of the WSP in presence of different authorization constraints.

1 Introduction

A security-sensitive business process (BP) is a structured collection of tasks, defining a workflow, equipped with an authorization policy defining which users are entitled to execute which tasks, and authorization constraints such as Separation or Binding of Duties (SoD or BoD) defining that certain tasks must be executed by different users, or the same user, respectively. The authorization policy and constraints are crucial to comply with regulations and prevent frauds. It is, however, of utmost importance to ensure that business continuity is not endangered, i.e. it must be possible to complete the BP while satisfying the authorization policy and constraints. Finding the best possible trade-off between security and business continuity for BPs is called the Workflow Satisfiability Problem (WSP) [6,16,4,7].

There are business rules, regulations, and policies that either cannot be encoded or are very difficult to encode by using simple SoD/BoD constraints [15]. Some examples are: requiring that a user executes only a certain number of tasks or requiring that users from different (or the same) departments execute a set of tasks. Even more complex policies involving conflicts of interest [26], confidentiality [5], and integrity [8] require data-based constraints. These practical needs have motivated the definition of different types of authorization constraints. In the literature, a variety of authorization constraints have been considered but no systematic classification has been given. Additionally, the proposed approaches to solve the WSP only support certain classes of authorization constraints.

Related work. The seminal work of Bertino et al. [6] described the specification and enforcement of authorization constraints in workflow management systems, presenting constraints as clauses in a logic program and an exponential algorithm for assigning users and roles to tasks without violating them, but considering only linear workflows. Tan et al. [29] defined a model for constrained workflow systems that includes constraints such as cardinality, SoD and BoD. They specified a workflow as a partial order on the set of tasks; and a constrained workflow authorization schema, associating roles to tasks. Crampton [14] extended these ideas by defining Type 1 constraints, and developing an algorithm to determine whether there exists an assignment of users to tasks that satisfies the constraints.

Wang and Li. [31] proposed a role-and-relation based access control model to describe the relationships between users and specify complex authorization constraints. The authors reduced the WSP to SAT, showed that it is NP-complete in authorization systems supporting simple constraints and that it is fixed-parameter tractable (FPT) with only BoD and SoD.

Crampton et al. [17] showed that the WSP remains FPT with counting and equivalence constraints. Later [16], they used logical combinations of constraints to support conditional workflows and Type 3 constraints by splitting one instance of the problem into several instances. Cohen et al. [11] solved the WSP using techniques for the Constraint Satisfaction Problem, which allowed the authors to devise a general algorithm that works for several families of constraints. Cohen et al. [12] demonstrated the practicality of the previously designed algorithm by adapting it to the class of user-independent counting constraints and showing its superiority when compared with the classical SAT reduction of the problem. Crampton et al. [15] showed that the WSP remains FPT for class-independent constraints and provided an algorithm to solve it. Crampton et al. [18] used model checking on an NP-complete fragment of LTL to synthesize and validate plans for security-sensitive workflows and argued that this approach is more robust, uniform, and expressive than previous formalizations.

Li and Wang introduced the Separation of Duties Algebra (SoDA) [25] to express and formalize policies based on users' attributes and the number of users executing tasks. The policies are enforced by low-level mechanisms such as static and dynamic SoD in RBAC [27]. Basin et al. [3] generalized SoDA's semantics to workflow traces and refined it for control-flow and role-based authorizations, implementing a SoD enforcement monitor for workflow engines. Later [4], the same authors used Hoare's Communicating Sequential Processes (CSP) to model workflows in two levels: control-flow and task execution, allowing them to synthesize monitors that enforce at run-time obstruction-free, or satisfying, workflow executions.

This work. Our previous approaches to solve the WSP [7,21,13] use an SMT-based model checker to synthesize run-time enforcement mechanisms for business processes augmented with access control policies and SoD/BoD constraints. We focused on these constraints because those works were developed in collaboration with SAP. SAP was mainly interested in SoD/BoD because these constraints are the most widely used by their customers. In this paper, we show how to extend

our previous encoding of authorization constraints to handle a large number of constraints that we have encountered in the literature and classified. After providing the background concepts on the WSP (Section 2), we propose the first classification of authorization constraints (Section 3). Then, show how to encode them in the declarative input language of an SMT-based model checker (Section 4.3) that is used in a tool called CERBERUS (Section 4.2), which is capable of solving the run-time version of the WSP. The tool has been integrated in an industrial framework for workflow management of SAP (Section 4.1). We finish by drawing some conclusions and discussing related and future work (Section 5).

2 Background

Let T be a finite set of tasks and U a finite set of users. A *scenario* is a finite sequence of pairs of the form (t, u), written as $t(u)$, where $t \in T$ and $u \in U$. The intuitive meaning of a scenario $\eta = t_1(u_1), \ldots, t_n(u_n)$ is that task t_i is executed before task t_j for $1 \leq i < j \leq n$ and that task t_k is executed by user u_k for $k = 1, \ldots, n$. A *workflow* $W(T, U)$ is a set of scenarios. There are various ways to specify security-sensitive workflows. For instance, [17] introduces the notion of "constrained workflow authorization schema," [4] uses CSP, and many works use (extensions of) Petri nets. We adopt the last approach, as it is one of the standard ways to formalize the semantics of workflows specified in BPMN [20]. We illustrate the specification of a security-sensitive workflow in (a variant of) BPMN using an example.

Example 1. The left side of Figure 1 shows a simple Loan Origination Process, with fours tasks: *Request Loan* ($t1$), *Evaluate External Credit Rating* ($t2$), *Evaluate Internal Credit Rating* ($t3$), and *Approve Loan* ($t4$). Task $t1$ has to be executed first, followed by $t2$ and $t3$ (in any order), followed by $t4$, so the behaviors $t1, t2, t3, t4$ and $t1, t3, t2, t4$ are allowed, whereas, e.g., $t1, t4, t3, t2$ is not (where $t1, \ldots, tn$ represents a sequence of n tasks executed in order, i.e. t_{i+i} is executed after t_i). Now imagine that $t3$ is only executed for loans of more than 10k Euro, then behavior $t1, t2, t4$ becomes allowed, but only for some instances (those where the data object "loan amount" is less than 10k). If the organization running this workflow adopts the authorization policy shown at the center of the Figure and the SoD constraints between $t2$ and $t3$ and between $t3$ and $t4$ (shown as dashed lines labeled by $\neq$ in the Figure), then any behavior containing, e.g., $t2(a)$ and $t3(a)$ is not allowed (where $t(u)$ means that user u executes task t).

The right side of Figure 1 shows the extended Petri net that can be automatically derived from the BPMN on the left side and that represents its semantics (see, e.g., [20,7]). Tasks are modeled as transitions or events (the boxes in the Figure) whereas places (the circles in the Figure) encode their enabling conditions. At the beginning, there will be just one token in place $p0$ which enables the execution of transition $t1$. This corresponds to the execution constraint that task $t1$ must be performed before all the others. The execution of $t1$ removes the token in $p0$ and puts a token in $p1$ and another in $p2$; this enables the execution

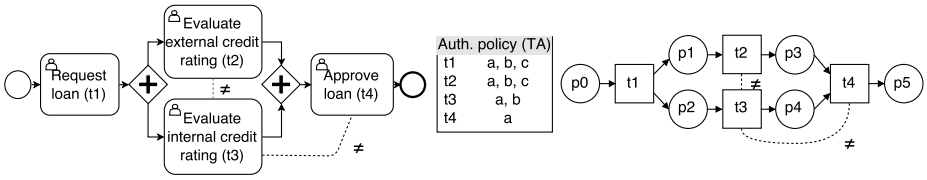

Fig. 1. Loan Origination Process in BPMN (left) and as a Petri net (right)

of $t2$ and $t3$. Indeed, this corresponds to the causality constraint that $t2$ and $t3$ can be executed in any order after $t1$ and before $t4$. The executions of $t2$ and $t3$ remove the tokens in $p1$ and $p2$ and put a token in $p3$ and one in $p4$, which, in turn, enables the execution of $t4$. This removes the token in $p3$ and $p4$ and puts a token in $p5$, which enables no more transitions. This corresponds to the fact that $t4$ is the last task to be executed. □

Among the scenarios in a workflow, we are interested in those that describe successfully terminating executions in which users execute tasks satisfying the authorization constraints and the authorization policy. Since the notion of successful termination depends on the definition of the workflow (e.g., in case of a conditional choice, we will have two acceptable execution sequences according to the Boolean value of the condition), in the following we focus only on the authorization policy and the authorization constraints while assuming that all the scenarios in the workflow characterize successfully terminating behaviors.

Given a workflow $W(T, U)$, an *authorization relation* TA is a sub-set of $U \times T$. Intuitively, $(u, t) \in TA$ means that u is authorized to execute task t. We say that a scenario η of a workflow $W(T, U)$ is *authorized* according to TA iff (u, t) is in TA for each $t(u)$ in η. An *authorization constraint* over a workflow $W(T, U)$ can be seen as a pair (T', Θ), where $T' \subseteq T$ is called the scope of c and Θ is a set of functions $\theta : T' \to U$ [15]. The functions in Θ specify the assignments of tasks to users that satisfy the constraint. Instead of enumerating every function $\theta \in \Theta$, it is common to define Θ implicitly by using a specification device. A catalog of such devices is presented in Section 3 below. Let C be a (finite) set of authorization constraints, a scenario η satisfies C iff η satisfies c, for each c in C. A scenario η of a workflow $W(T, U)$ is *eligible according to a set* C *of authorization constraints* iff η satisfies C.

The problems raised by the conflicting goals of business compliance and business continuity are further complicated by the interplay between control-flow, data-flow, and authorization. Notice that a common practice in the analysis of workflow satisfiability is to abstract away from parts of a workflow specification. No work besides [7] takes into account the data-flow (some completely disregard it, e.g., [15], and some model it with non-deterministic decisions, e.g., [4]). It is also common to limit the allowed control-flow constructs and supported authorization constraints. There are different versions of the WSP and one main distinction is whether the order of execution of the tasks is taken into account. Crampton [16] defines a plan $\pi : T \mapsto U$ and a schedule as a tuple $(t_1, \ldots, t_k)$ such that

$\{t_1, \ldots, t_k\} = T$ and $t_j \not\leq t_i$ for each $1 \leq i < j \leq k$. The unordered WSP admits as solution a valid plan π, whereas the ordered version admits as solution a plan π with a schedule σ, i.e. the plan must respect the ordering of tasks defined by the control-flow. The two versions of the WSP are only equivalent for well-formed workflows [16], i.e. workflows where for all tasks t_i and t_j that can be executed in any order, $(t_i, t_j, \rho) \in C$ iff $(t_j, t_i, \tilde{\rho}) \in C$ (where $\tilde{\rho}$ is defined as $\{(u, u') \in U \times U : (u', u) \in \rho\}$). We define the (Ordered) WSP as follows.

Definition 1 ((Ordered) Workflow Satisfiability Problem (WSP)). *Given a workflow $W(T, U)$, an authorization relation TA, and a set C of authorization constraints, return (if possible) a scenario η which is authorized according to TA and eligible according to C.*

3 A catalog of authorization constraints

Several classes of authorization constraints for workflows have been identified in the literature. They can all be used, with some ingenuity, to define the functions $\theta \in \Theta$, so they can be recast in the form (T', Θ) shown above [11].

Counting constraints are of the form (t_l, t_r, T'), where $1 \leq t_l \leq t_r \leq k$. A plan satisfies a counting constraint if a user performs either no tasks in T' or between t_l and t_r tasks. One example of counting constraint is $(1, 2, \{t1, t2, t3\})$, which is satisfied if a user $u1$ executes 0, 1 or 2 tasks among those in $\{t1, t2, t3\}$.

Entailment constraints are of the form (T_1, T_2, ρ), where $T_1 \cup T_2 = T'$ and $\rho \subseteq U \times U$. A plan satisfies an entailment constraint iff there exist $t_1 \in T_1$ and $t_2 \in T_2$ such that $(\pi(t_1), \pi(t_2)) \in \rho$. Entailment constraints can be further subdivided in three types. In Type 1 constraints, both sets T_1 and T_2 are singletons. In Type 2 constraints, at least one of the sets must be a singleton, whereas in Type 3 there are no restrictions on the cardinality of sets. Examples of Type 1, 2, and 3 constraints are $(\{t1\}, \{t2\}, \neq)$, $(\{t1, t2\}, \{t3\}, \neq)$, and $(\{t1, t2\}, \{t3, t4\}, \neq)$, respectively. The first constraint is satisfied if a user $u1$ executes $t1$ and $u2$ executes $t2$ (because $u1 \neq u2$). The second and third constraints are satisfied if $u1$ executes $t1$ and $u2$ executes $t3$. Those are examples of SoD constraints, BoD constraints can be similarly defined by using $=$ instead of $\neq$. A special class of Type 1 constraints are equivalence-based constraints, of the form $(t_1, t_2, \sim)$, where $\sim$ is an equivalence relation on U. A plan satisfies this kind of constraint if the user who executes t_1 and the user who executes t_2 belong to the same equivalence class, e.g., same role (or to different classes for $\not\sim$ constraints).

User-independent constraints c are those where given a plan π that satisfies c and any permutation $\phi : U \to U$, the plan $\pi' = \phi(\pi(s))$ also satisfies c [11]. I.e. user-independent constraints are those whose satisfaction does not depend on the individual identities of users. The SoD constraints presented so far are user-independent, whereas a constraint requiring a specific user to perform at least one task in a set is not user-independent [12].

Class-independent constraints are those whose satisfaction depends only on the equivalence classes that users belong to [15]. Formally, let c be a constraint,

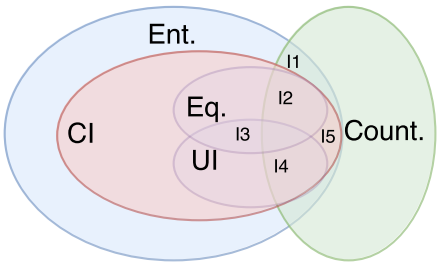

Fig. 2. Relations between constraint classes

$\sim$ be an equivalence relation on U, $U^{\sim}$ be the set of equivalence classes induced by $\sim$, and $u^{\sim} \in U^{\sim}$ be the equivalence class containing u. Then, for any plan π, we can define a function $\pi^{\sim} : T \to U^{\sim}$ as $\pi^{\sim}(t) = (\pi(t))^{\sim}$. Finally, c is class-independent for $\sim$ if for any function θ, $\theta^{\sim} \in \Theta$ implies $\theta \in \Theta$, and for any permutation $\phi : U^{\sim} \to U^{\sim}$, $\theta^{\sim} \in \Theta^{\sim}$ implies $\phi \circ \theta^{\sim} \in \Theta^{\sim}$ [15]. One example of class-independent constraint is $(\{t1\}, \{t2\}, \sim)$, where the classes induced by $\sim$ corresponds to departments of a company. This constraint is satisfied if $u(t1) \sim u(t2)$, i.e. the user executing $t1$ and the user executing $t2$ are in the same department. Indeed, every equivalence constraint $(t_1, t_2, \sim)$ (or $(t_1, t_2, \not\sim)$) is class-independent and every user-independent constraint is class-independent with respect to the identity relation [15].

3.1 Classification of constraints

It is not easy to classify authorization constraints in terms of expressiveness, partly because there are many different frameworks to express them. For instance, entailment constraints of Type 3 clearly include those of Types 1 and 2, but counting constraints can also be used to express some forms of SoD [33], so entailment and counting constraints are not disjoint (i.e. in some cases, it is possible to express the same set of behaviors using a counting constraint or an entailment one). Also, clearly user-independent and class-independent constraints subsume parts of the other classes, but it is not clear which parts.

Figure 2 shows an attempt to systematically classify some classes of authorization constraints for workflow systems presented in the literature. The Figure shows the sets *Ent.* of entailment constraints (the subsets of constraints of Types 1, 2, and 3 are not shown to keep the Figure readable), *Count.* of counting constraints, *Eq.* of equivalence constraints, *CI* of class-independent constraints and *UI* of user-independent constraints. Naturally, $Eq. \subset Ent.$ and $CI. \subset Ent.$, since an equivalence relation is an instance of a binary relation. The facts $UI \subset CI$ and $Eq. \subset CI$ were shown by Crampton et al. [15].

The Figure also shows the following intersections: $I1 = Ent. \cap Count.$, $I2 = Eq. \cap Count.$, $I3 = Eq. \cap UI$, $I4 = Count. \cap UI$, $I5 = Count. \cap CI$. We can

show that these intersections are non-empty by using SoD and BoD constraints as examples. $I1$ and $I2$ are non-empty because SoD and BoD can be specified using entailment: $(t1, t2, \neq)$ and $(t1, 2, =)$, respectively; counting: $(1, 1, \{t1, t2\})$ and $(2, 2, \{t1, t2\})$, respectively; or equivalence, since $=$ is an equivalence relation. $I3$, $I4$, and $I5$ are non-empty because both constraints are user-independent [12], which also makes them class-independent [15].

To the best of our knowledge, there has never been a comparison between the expressive power of other frameworks, e.g., SoDA and the constraint classes defined by Crampton et al.

3.2 Data-based constraints

In business processes, authorization policies and constraints are usually specified and enforced based on the tasks. But policies and constraints can also be defined based on data objects. This allows increased expressiveness (policies such as Chinese Wall [9] cannot be expressed solely on the tasks), as well as simplified specification (at design-time) and enforcement (at run-time), since some policies may require many more task-based constraints than data-based ones. Below, we motivate some well-known classes of data-based policies from the security literature.

Data authorization refers to a user's permission to access a data object in a workflow. An example of the need for data authorization on top of task authorization is to manage conflict of interests (CoI) in contract tender evaluations [2]. In this example, if a user is authorized to perform task *Evaluate Tender*, but they work for one of the companies proposing a tender, they should not be authorized to evaluate the tender of their own company. To perform this task, users should have permission not only to execute the task, but also to access the tender data.

Chinese Wall is used to prevent the CoI that arises when a user has access to the data of two competing organizations. To avoid this kind of conflict, the data is separated into sets representing the classes of conflict and when a user has access to the data of one of the elements of the set, they cannot access the data of the other elements. More examples of CoI policies can be found in [26].

Need to know and privacy constraints can be used to block, for instance, the access to two or more data objects that, taken together, can reveal information that a single object cannot (e.g., a relation of names of patients and time they came in with a relation of medical procedures and time they were performed can be used to identify patients). Need to know means that a user should only know the minimum amount of information required to complete a task. One example is that to approve a loan, financial data is required, but not personal data, so the user who approves the loan should not know the personal data of the applicant. In this example a constraint could be defined between the data objects personal data and financial data, so that any user will only have access to one of them.

Other confidentiality [5] and integrity [8] policies can be modeled with data authorization and data constraints, but they require a separation between read access and write access. In the low-water-mark integrity policy [8], for instance, users and data objects have integrity levels ($l[\cdot]$) and whenever a user reads a

data object, his/her integrity level is updated ($l[s] \leftarrow l[s] \wedge l[o]$, where $\wedge$ stands for the glb between integrity levels), whereas writing is permitted if $l[o] \leq l[s]$.

4 Encoding constraints in Cerberus

We have implemented an approach to solve the WSP by synthesizing run-time monitors for security-sensitive workflows in a tool called Cerberus [13]. Below, we first present a high-level overview of the tool and its integration in an industrial environment. Instead of providing full details, we focus on those aspects that are relevant to model the authorization constraints considered in Section 3 and show the termination of monitor synthesis.

4.1 Overview of Cerberus

A reference architecture for Workflow Management (WFM) systems is composed of the five blue elements shown in Figure 3. *Workflow Modeling* is a user interface for a Process Designer to create workflow models in a modeling language, e.g., BPMN. Models are stored in a *Workflow Model Repository*, while the *Workflow Engine* interprets the models and directs the execution to *Invoked Applications*, in the case of system and script tasks, or to a *Graphical User Interface* (GUI), in the case of user tasks, which are performed by Process Participants.

On top of the WFM components, we add the Cerberus components shown in red in Figure 3. The *Monitor Synthesizer* is responsible for interpreting the workflow model and translating it into a transition system format accepted by a *Symbolic Model Checker* (SMC) capable of computing a reachability graph whose paths are all possible executions of the workflow. To solve the WSP, a monitor needs to look to the current state, the past execution history and possible future executions to check if there is any possibility to finish the process. Therefore, we need to be able to pre-compute all possible executions. Notice that pre-computing a reachability graph in the presence of data values is infeasible, but, as already mentioned, abstracting away run-time data is standard practice for approaches solving the WSP. Note that only the workflow model (representing the execution constraints) extended with authorization constraints is input to the monitor synthesis. This allows the synthesized monitor to support different authorization policies at run-time. The reachability graph is translated into a language such as Datalog or SQL and stored in the *Monitor Repository*. The *Monitor* itself sits between the GUI and the workflow engine and grants or denies user requests to execute tasks (users only access tasks through the GUI and automatic tasks are not part of the authorization policy or constraints).

Cerberus is implemented on top of the SAP HANA Operational Intelligence platform (OpInt)[1], which offers a BPMN modeling and enactment environment to synthesize, store, combine, and retrieve run-time monitors for security-sensitive workflows therein modeled and enacted. HANA Studio is the IDE that acts as the

[1] `https://help.sap.com/hana-opint`

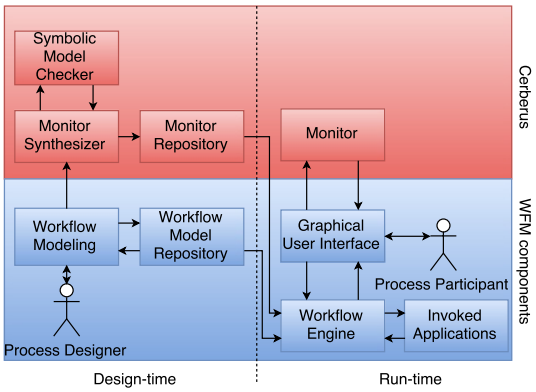

Fig. 3. The architecture of CERBERUS and its interface with a WFM system

Workflow Modeling component, while the HANA Repository implements both the Workflow Model Repository and the Monitor repository. We added the constraint specification and monitor synthesis capabilities in the IDE and used MCMT [23] as the SMC (we explain this choice below). The Monitor Synthesizer is written in Python (core algorithms) and JavaScript (IDE and repository integration). The monitors are output in SQL as a view that is queried by the execution engine. The result of this query is used to grant or deny a user's request to execute a task. The OpInt Workflow Engine translates BPMN models to executable JavaScript and SQL code that manage and perform the tasks in the workflows. The invoked applications are handled by SQL procedure calls and the GUI for user tasks is integrated in a web task management dashboard.

The termination of the various modules in CERBERUS is obvious, except for the SMC. Thus, below, we discuss under which hypotheses termination is guaranteed for this module.

4.2 Run-time monitor synthesis

The *Monitor Synthesizer*—by invoking the SMC—solves the WSP by synthesizing run-time monitors capable of ensuring that all executions terminate and authorization constraints in a workflow are satisfied using the approach described in [7]. Here, for lack of space, we focus on the SMC and we discuss the assumptions under which it is guaranteed to terminate.

The SMC takes as input a symbolic transition system S whose executions correspond to those of the security-sensitive workflow. S is *automatically* derived from the (extended) Petri net defining the semantics of a BPMN specification by using standard techniques (see, e.g., [28,7]). The symbolic transitions derived in this way have the following form:

$$t(z) : en_{CF} \wedge en_{Auth} \rightarrow act_{CF} || act_{Auth} \tag{1}$$

where $t(z)$ identifies a transition t executed by a user identified by the variable z; en_{CF} and en_{Auth} are enabling conditions (on the control-flow and authorization, respectively); act_{CF} and act_{Auth} are the effects of the execution of the transition and $||$ represents a parallel update of variables. The variable z occurs in the enabling condition en_{Auth} and, possibly, in some of the updates in act_{Auth}.

Example 2. To illustrate, recall Figure 1 and observe that the fact that there is at most one token per place is an invariant of the Petri net. This allows us to symbolically represent the net as follows: we introduce a Boolean variable per place (named as the places in Figure 1) together with a Boolean variable representing the fact that a task has already been executed (denoted by d_t and if assigned to true implies that task t has been executed). So, for instance, the enabling condition for the execution constraint on task $t1$ can be expressed as $p0 \wedge \neg d_{t1}$ meaning that the token is in place $p0$ and transition $t1$ has not yet been executed. The effect of executing transition $t1$ is to assign F(alse) to $p0$ and T(rue) to $p1$, $p2$ and d_{t1}; in symbols, we write $p0, p1, p2, d_{t1} := F, T, T, T$. The other transitions are modeled similarly.

Besides the constraints on the execution of tasks, The Petri net in Figure 1 shows also the same authorization constraints of the BPMN model. These are obtained by taking into consideration both the access control policy P granting or denying users the right to execute tasks and the SoD constraints between pairs of tasks. To formalize these, we introduce two functions a_t and h_t from users to Boolean, for each task t, which are such that $a_t(u)$ is true iff u has the right to execute t according to the policy P and $h_t(u)$ is true iff u has executed task t. Notice that a_t is a function that behaves as an abstract interface to the policy P whereas h_t is a function that evolves over time and keeps track of which users have executed which tasks. For instance, the enabling condition for the authorization constraint on task $t1$ is simply $a_{t1}(u)$, i.e. it is required that the user u has the right to execute $t1$, and the effect of its execution is to record that u has executed $t1$, i.e. $h_{t1}(u) := T$ (notice that this assignment leaves unchanged the value returned by h_{t1} for any user u' distinct from u). As another example, let us consider the enabling condition for the authorization constraint on $t2$: besides requiring that u has the right to execute $t2$ (i.e. $a_{t2}(u)$), we also need to require the SoD constraints with $t3$ (i.e. $\neg h_{t3}(u)$). The authorization constraints on the other tasks are modeled in a similar way.

Table 1. Workflow as symbolic transition system

event	enabled		action	
	CF	Auth	CF	Auth
$t1(u)$	$p0 \wedge \neg d_{t1}$	$a_{t1}(u)$	$p0, p1, p2, d_{t1} := F, T, T, T$	$h_{t1}(u) := T$
$t2(u)$	$p1 \wedge \neg d_{t2}$	$a_{t2}(u) \wedge \neg h_{t3}(u)$	$p1, p3, d_{t2} := F, T, T$	$h_{t2}(u) := T$
$t3(u)$	$p2 \wedge \neg d_{t3}$	$a_{t3}(u) \wedge \neg h_{t2}(u)$	$p2, p4, d_{t3} := F, T, T$	$h_{t3}(u) := T$
$t4(u)$	$p3 \wedge p4 \wedge \neg d_{t4}$	$a_{t4}(u) \wedge \neg h_{t3}(u)$	$p3, p4, p5, d_{t4} := F, F, T, T$	$h_{t4}(u) := T$

Table 1 shows the formalization of all transitions in the extended Petri net of Figure 1. The first column reports the name of the transition together with the fact that it is dependent on the user u taking the responsibility of its execution. The second column shows the enabling condition divided in two parts: CF, pertaining to the execution constraints, and Auth, to the authorization constraints. The third and last column list the effects of the execution of the transition again divided in two parts: CF, for the workflow, and Auth, for the authorization.

The set of final states can be specified by the following formula:

$$\neg p0 \wedge \neg p1 \wedge \neg p2 \wedge \neg p3 \wedge \neg p4 \wedge p5 \wedge d_{t1} \wedge d_{t2} \wedge d_{t3} \wedge d_{t4}$$

saying that there is just one token in $p5$ and that all tasks have been executed. The set of initial states can be specified, dually, by the formula:

$$p0 \wedge \bigwedge_{i=1,\ldots,5} \neg p_i \wedge \bigwedge_{i=1,\ldots,4} \neg d_{ti} \wedge \forall u.(\neg h_{t1}(u) \wedge \neg h_{t2}(u) \wedge \neg h_{t3}(u) \wedge \neg h_{t4}(u))$$

saying that there is just one token in $p0$, no task has been executed yet, and no user has executed any task. □

After building the symbolic transitions and the formulae defining the sets of initial and final states, the SMC computes a symbolic representation of the set of states that are (backward) reachable from the set of final states. In other words, the WSP is reduced to a reachability problem under the assumption that no transition can be enabled infinitely often without being executed. This assumption (called strong fairness in the literature) is considered reasonable in the context of workflow management [30] since decisions to execute tasks are under the responsibility of applications or humans.

To compute the fix-point, MCMT (an SMT-based model checker) computes a directed graph $RG = (N, \lambda, E)$, called reachability graph, whose edges in E are labeled by task-user pairs in which users are symbolically represented by variables and whose nodes in N are labeled—according to the labeling function λ—by a formula of first-order logic. We omit the details of the construction of the full reachability graph and point the interested reader to [7]. Here, it is enough to say that it is built in two steps: a fix-point procedure and a post-processing. The resulting graph is such that its paths describe all possible executions of a transition system that terminate and satisfy the authorization constraints. While the termination of the post-processing step is guaranteed by adopting a suitable semantics for loops, namely the one based on "release-point semantics" of [10] which requires to consider only the user who executed the last iteration of the loop, forgetting all the others), the termination of the fix-point computation for transition systems with a finite but unbounded number of users is non-obvious.

By using MCMT as the SMC, we get the following two advantages. First, MCMT is capable of introducing on-demand new (existential) variables to symbolically represent enough users to satisfy authorization constraints without bounding their number *a priori*. Second, the following theorem is an easy consequence of the results in [22]. Preliminary, generalizing the observations in Example 2, we assume

that the formulae describing the transitions as well as the initial and final sets of states are built out of a set V of state predicates (whose values evolve over time) of arity 0 for the places $(p0, p1, \dots)$ and transitions $(d_{t1}, d_{t2}, \dots)$ and of arity 1 for tracking the history of which user has executed a certain task $(h_{t1}, h_{t2}, \dots)$; the set $H \subset A$ contains only the history variables $h_{t1}, \dots$ We also permit that the static predicates (whose values stay constant over time) of arity 1 from a given finite set A may also occur in such formulae and constitute the interfaces to the authorization policy. Below, a quantifier-free (existentially or universally quantified) formula built out of the predicates in a set $X \in \{H, V, A, V \cup A\}$ is termed X-quantifier-free (existentially or universally quantified, respectively) formula.

Theorem 1. MCMT *terminates when computing the reachability graph of a symbolic transition system in which transitions are of the form (1) where en_{CF} is a V-quantifier-free formula, en_{Auth} is a A-quantifier-free formulae, the final formula is a V-quantifier-free formula, and the initial formula is a conjunction of a V-quantifier-free formula with a H-universally quantified formula.*

Below, we show how Theorem 1 can help showing the termination of SMC on several classes of authorization constraints in Section 3.

4.3 Encoding constraints

We illustrate the main ideas of our symbolic encoding by considering the SoD constraint between $t3$ and $t4$ in Figure 1. The constraint can be specified as an additional condition that must hold in every state of the executions of the Loan Origination Process (LOP): $\forall w.\neg(h_{t3}(w) \wedge h_{t4}(w))$ or, equivalently,

$$\forall w.(\neg h_{t3}(w) \vee \neg h_{t4}(w))\,. \tag{2}$$

To enforce that (2) is satisfied in every state of every possible execution of the LOP, we can conjoin it with the enabling condition en_{Auth} of each transition in S. For instance, transition $t4$ becomes

$$t4(z) : en_{CF} \wedge a_{t1}(z) \wedge \forall w.(\neg h_{t3}(w) \vee \neg h_{t4}(w)) \rightarrow act_{CF} || h_{t4}(z) := T$$

where en_{CF} and act_{CF} abbreviate the symbolic representations of the enabling condition and effect, of the control-flow. We can eliminate the universal quantifier by instantiating w with z by using the results in [1], i.e. it is sufficient to consider the following transition:

$$\overline{t4}(z) : en_{CF} \wedge a_{t1}(z) \wedge (\neg h_{t3}(z) \vee \neg h_{t4}(z)) \rightarrow act_{CF} || h_{t4}(z) := T\,. \tag{3}$$

Notice that (3) can be further simplified to

$$\overline{t4}(z) : en_{CF} \wedge a_{t1}(z) \wedge \neg h_{t3}(z) \rightarrow act_{CF} || h_{t4}(z) := T\,.$$

since if $t4$ has not yet been executed, then $\neg h_{t4}(z)$ must hold (indeed, the last formula is equivalent to that in the last line of Table 1).

Theorem 2. *Let T be the set of transitions of the form*

$$t(z) : en_{CF} \wedge a_t(z) \wedge \forall w. \neg h_{t'}(w) \rightarrow act_{CF} || act_{Auth} \tag{4}$$

for $h_{t'}$ in H and $\overline{T}$ be the set of transitions obtained by instantiating w with z. Under the same assumptions of Theorem 1, the set of possible executions of T and $\overline{T}$ are the same.

This result significantly broadens the scope of applicability of Theorem 1, thereby enabling CERBERUS to cover a large variety of authorization constraints used in security-sensitive workflows.

Interestingly, similar results can be derived for entailment constraints of the form (T_1, T_2, ρ) for T_1 and T_2 sub-sets of the set of tasks and ρ a binary relation over the set of users and for counting constraints (t_l, t_r, T') for $1 \leq t_l \leq t_r$ and T' sub-set of the set of tasks. We omit the details, for lack of space, and just explain how such constraints can be expressed as logical expressions to be conjoined to the enabling condition en_{Auth} in (1). The first type of constraints can be expressed by the formula

$$\forall z_1, z_2. \bigvee_{t_1 \in T_1} \bigvee_{t_2 \in T_2} h_{t1}(z_1) \wedge h_{t2}(z_2) \Rightarrow \rho(z_1, z_2)$$

with ρ a relation that can be specified by sentences that are universally quantified formulae and built out of predicate symbols with equality (no function symbols are allowed). The second type of constraints can be expressed by the formula

$$\forall U_{T'}. \left(\bigwedge_{t' \in T'} h_{t'}(u_{t'}) \Rightarrow \mathsf{AtMost}(U_{T'}, t_r) \wedge \mathsf{AtLeast}(U_{T'}, t_l) \right)$$

where $U_{T'} = \{u_{t'} | t' \in T'\}$, $\mathsf{AtMost}(U_{T'}, t_r)$ abbreviates the disjunction of all formulae of the form $\forall \overline{U_{T'}}. \bigvee_{x \neq y \in U_{T'}} x \neq y$ for $\overline{U_{T'}} \subseteq U_{T'}$ of cardinality t_r and $\mathsf{AtLeast}(U_{T'}, t_l)$ abbreviates the conjunction of all formulae of the form $\exists \overline{U_{T'}}. \bigwedge_{x \neq y \in U_{T'}} x \neq y$ for $\overline{U_{T'}} \subseteq U_{T'}$ of cardinality t_l.

It is also possible to express the data-based authorization constraints of Section 3.2 by using logical expressions to define the a_t's in A, even including constraints that are history-dependent in a way similar to the h_t's in H. We omit the details, but emphasize that since the expressions needed to express these policies are quantifier-free, Theorem 1 applies straightforwardly. Thus, CERBERUS is able to synthesize monitors for security-sensitive workflows containing also these constraints.

We conclude by observing that CERBERUS is capable of synthesizing monitors for security-sensitive workflows containing a mixture of the classes of constraints considered above.

5 Conclusion

We have motivated and presented a classification of authorization constraints in security-sensitive workflows, showed how to encode them in the declarative

input language of an SMT-based model checker, and described applications of this approach. This work shows the flexibility of the SMT approach to solve the run-time version of the WSP in the presence of different authorization constraints.
Future work. Instance-spanning constraints [24] restrict what users can do across several instances of the same workflow (inter-instance), across several instances of different workflows (inter-process), or across workflows in different organizations (inter-organization). The most usual case is inter-instance authorization constraints, which have been studied in, e.g., [32]. Since we adopt the approach of having one monitor for each instance, support for inter-instance constraints would require a global synchronization of the states of each monitor, possibly using a global execution history. A possibility would be to design a central entity to which selected parts of the state of each monitor are communicated so that it can take the right decision to avoid that some inter-instance constraint is violated. Indeed, each monitor should ask the decision of the central entity before taking a decision. Although the design of this central entity may be challenging, we could take inspiration from cache-coherence protocols (see, e.g., [19]).

References

1. F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, and G.P. Rossi. Universal Guards, Relativization of Quantifiers, and Failure Models in Model Checking Modulo Theories. *JSAT*, 8:29–61, 2012.
2. B. Alhaqbani, M. Adams, C.J. Fidge, and A.H.M. ter Hofstede. Privacy-aware workflow management. In *Proc. of BPM*. Springer, 2013.
3. D. Basin, S.J. Burri, and G. Karjoth. Dynamic enforcement of abstract separation of duty constraints. *TISSEC*, 15(3):13:1–13:30, 2012.
4. D. Basin, S.J. Burri, and G. Karjoth. Obstruction-free authorization enforcement: Aligning security and business objectives. *JCS*, 22(5):661–698, 2014.
5. D. Bell. The bell-lapadula model. *JCS*, 4(2):3, 1996.
6. E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *TISSEC*, 2(1):65–104, 1999.
7. C. Bertolissi, D.R. dos Santos, and S. Ranise. Automated Synthesis of Run-time Monitors to Enforce Authorization Policies in Business Processes. In *Proc. of ASIACCS*. ACM, 2015.
8. K. Biba. Integrity considerations for secure computer systems. Technical report, DTIC Document, 1977.
9. D. Brewer and M.J. Nash. The chinese wall security policy. In *Proc. of S&P*. IEEE, 1989.
10. S.J. Burri and G. Karjoth. Flexible scoping of authorization constraints on business processes with loops and parallelism. In *Proc. of BPMW*. Springer, 2012.
11. D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones. Iterative plan construction for the workflow satisfiability problem. *JAIR*, 51:555–577, 2014.
12. D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones. Algorithms for the workflow satisfiability problem engineered for counting constraints. *J. Comb. Optim.*, 32(1):3–24, 2016.

13. L. Compagna, D.R. dos Santos, S.E. Ponta, and S. Ranise. Cerberus: Automated Synthesis of Enforcement Mechanisms for Security-sensitive Business Processes. In *Proc. of TACAS*. Springer, 2016.
14. J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proc. of SACMAT*. ACM, 2005.
15. J. Crampton, A. Gagarin, G. Gutin, M. Jones, and M. Wahlström. On the workflow satisfiability problem with class-independent constraints for hierarchical organizations. *TOPS*, 19(3):8:1–8:29, 2016.
16. J. Crampton and G. Gutin. Constraint expressions and workflow satisfiability. In *Proc. of SACMAT*. ACM, 2013.
17. J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity and kernelization of the workflow satisfiability problem. *TISSEC*, 16(1):4, 2013.
18. J. Crampton, M. Huth, and J. Kuo. Authorized workflow schemas: deciding realizability through ltl(f) model checking. *STTT*, 16(1):31–48, 2014.
19. G. Delzanno. Automatic verification of parameterized cache coherence protocols. In *Proc. of CAV*. Springer, 2000.
20. R.M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in bpmn. *Inf. and Soft. Tech.*, 50(12):1281 – 1294, 2008.
21. D.R. dos Santos, S. Ranise, and S.E. Ponta. Modular Synthesis of Enforcement Mechanisms for the Workflow Satisfiability Problem: Scalability and Reusability. In *Proc. of SACMAT*. ACM, 2016.
22. S. Ghilardi and S. Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *LMCS*, 2010.
23. S. Ghilardi and S. Ranise. Mcmt: A model checker modulo theories. In *Proc. of IJCAR*. Springer, 2010.
24. M. Leitner, J. Mangler, and S. Rinderle-Ma. Definition and enactment of instance-spanning process constraints. In *Proc. of WISE*. Springer, 2012.
25. N. Li and Q. Wang. Beyond separation of duty: An algebra for specifying high-level security policies. *J. ACM*, 55(3):12:1–12:46, 2008.
26. N. Nassr and E. Steegmans. Mitigating conflicts of interest by authorization policies. In *Proc. of SIN*. ACM, 2015.
27. R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-based access control models. *IEEE Computer*, 2(29):38–47, 1996.
28. S. Sankaranarayanan, H. Sipma, and Z. Manna. Petri net analysis using invariant generation. In *In Verification: Theory and Practice*. Springer, 2003.
29. K. Tan, J. Crampton, and C.A. Gunter. The consistency of task-based authorization constraints in workflow. In *Proc. of CSF*. IEEE, 2004.
30. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Comp.*, 23(3):333–363, 2011.
31. Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *TISSEC*, 13(4):40:1–40:35, 2010.
32. J. Warner and V. Atluri. Inter-instance authorization constraints for secure workflow management. In *Proc. of SACMAT*. ACM, 2006.
33. C. Wolter, A. Schaad, and C. Meinel. Task-based entailment constraints for basic workflow patterns. In *Proc. of SACMAT*. ACM, 2008.