Solving Multi-Objective Workflow Satisfiability Problems with Optimization Modulo Theories Techniques

Clara Bertolissi
Aix Marseille Univ. & CNRS
clara.bertolissi@lis-lab.fr

Daniel R. dos Santos
Eindhoven University of Technology
d.r.dos.santos@tue.nl

Silvio Ranise
Fondazione Bruno Kessler
ranise@fbk.eu

ABSTRACT

Security-sensitive workflows impose constraints on the control-flow and authorization policies that may lead to unsatisfiable instances. In these cases, it is still possible to find "least bad" executions where costs associated to authorization violations are minimized, solving the so-called Multi-Objective Workflow Satisfiability Problem (MO-WSP). The MO-WSP is inspired by the Valued WSP and its generalization, the Bi-Objective WSP, but our work considers quantitative solutions to the WSP without abstracting control-flow constraints. In this paper, we define variations of the MO-WSP and solve them using bounded model checking and optimization modulo theories solving. We validate our solutions on real-world workflows and show their scalability on synthetic instances.

CCS CONCEPTS

• **Security and privacy → Software security engineering**; *Formal methods and theory of security*;

KEYWORDS

Business process, Workflow Satisfiability, Optimization Modulo Theories

ACM Reference format:
Clara Bertolissi, Daniel R. dos Santos, and Silvio Ranise. 2018. Solving Multi-Objective Workflow Satisfiability Problems with Optimization Modulo Theories Techniques. In *Proceedings of The 23rd ACM Symposium on Access Control Models & Technologies (SACMAT), Indianapolis, IN, USA, June 13–15, 2018 (SACMAT '18),* 13 pages.
https://doi.org/10.1145/3205977.3205982

1 INTRODUCTION

A workflow specifies a collection of tasks, whose execution is initiated by humans or software agents executing on their behalf, and the constraints on the order of execution of those tasks. Security-related dependencies are specified as authorization policies, stating which users can execute which tasks, and authorization constraints imposed on task execution, e.g., Separation of Duties (SoD) whereby two distinct users must execute two tasks.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SACMAT '18, June 13–15, 2018, Indianapolis, IN, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5666-4/18/06…$15.00
https://doi.org/10.1145/3205977.3205982

Authorization policies and constraints are crucial to ensure the security of workflow systems and to avoid errors and frauds [28], but they may also lead to situations where a workflow instance cannot be completed because no task can be executed without violating either the authorization policy or the constraints. These deadlocks are conflicts between compliance and continuity which may be resolved by administrators granting additional permissions to users (thus hindering compliance) or canceling the execution (precluding continuity). Depending on the scenario, it may be preferable to guarantee either security or continuity. In all cases, it is desirable to have "minimal" (in some sense) violations. The Multi-Objective Workflow Satisfiability Problem (MO-WSP), considered in this paper, amounts to strike the best possible trade-off between security and continuity while minimizing the costs of violations to a policy or constraints.

The **main contributions** of this paper are the definition and solution of the MO-WSP using Bounded Model Checking (BMC) and Optimization Modulo Theories (OMT). The MO-WSP is inspired by the Valued WSP [12] and its generalization, the Bi-Objective Workflow Satisfiability Problem (BO-WSP) [13], but our work considers quantitative solutions to the WSP with an ordered execution of tasks, i.e., without abstracting the control-flow constraints. Our symbolic solution is also able to handle control-flow patterns [33], such as alternative execution, since we can encode these patterns directly in the transition system used by BMC (instead of splitting a workflow into multiple deterministic instances, as in [11, 14]). The use of off-the-shelf OMT solvers, instead of custom algorithms, provides a uniform toolkit to explore different optimization modes (such as Pareto and those based on linear cost functions), thereby gaining the freedom to evaluate the trade-offs offered by different optimization strategies. **Another contribution** is the implementation and evaluation of the proposed solution on real and synthetic workflows. The results show that the technique has a good performance due to an ingenious encoding of the problem that exploits the parallel executions of tasks in the workflow.

The rest of this paper is organized as follows. Sec. 2 discusses the original Workflow Satisfiability Problem and its valued versions; Sec. 3 presents our solution to the MO-WSP; in Sec. 4, we evaluate the performance of our solution; Sec. 5 discusses related work; and Sec. 6 concludes the paper.

2 WORKFLOW SATISFIABILITY

Given the control-flow (e.g., a task should be executed before all the others) and the authorization constraints (e.g., two tasks should be executed under the responsibility of two distinct users, known as Separation of Duties), a (decision) problem is to check for the existence of an assignment of the tasks to entitled users such that all the control-flow and authorization constraints are satisfied. In the literature, this is called the (ordered version of the) Workflow Satisfiability Problem (WSP) [34].

Example 2.1. The goal of the Trip Request Workflow (TRW) is to request trips for employees in an organization. TRW is composed of five tasks: Trip request ($t1$), Car rental ($t2$), Hotel booking ($t3$), Flight reservation ($t4$), and Trip validation ($t5$). The execution of the tasks is constrained as follows: $t1$ must be executed first, then $t2$, $t3$ and $t4$ can be executed in any order, with $t4$ being an optional task (only performed for long trips), and when all have been performed, $t5$ can be executed. Overall, there are six possible task execution sequences of length 5, in which the first is always task $t1$, the last is always task $t5$, and—in between—there is any one of the six permutations of $t2$, $t3$ and $t4$; there are also 2 sequences of length 4: $t1, t2, t3, t5$ and $t1, t3, t2, t5$.

The TRW can be modeled in BPMN [35] as shown in Figure 1: the circle on the left represents the start event (triggering the execution of the workflow), whereas that on the right represents the end event (terminating the execution of the workflow), tasks are depicted by labeled boxes, the constraints on the execution of tasks are shown as solid arrows for sequence flows and diamonds labeled by '+' for parallel flows or by 'X' for alternative flows.

Besides control-flow constraints, the BPMN in Figure 1 shows also authorization constraints. The man icon inside the box of a task t means that t must be executed under the responsibility of a user u according to an access control policy TA, specified by the table in Figure 1: a user u is entitled to execute t iff there is a line of the table in which u is associated with t.

A dashed line labeled by $\neq$ connecting two tasks t and t' denotes a Separation of Duties (SoD) constraint (see, e.g., [10]), i.e. there must exist two distinct users $u, u' \in U$ such that u executes t and u' executes t'. SoD constraints are typically used to prevent frauds. In Figure 1, five SoD constraints are depicted, requiring the following pairs of tasks to be executed by distinct users in any sequence of task executions of the workflow: $(t1, t2)$, $(t1, t4)$, $(t2, t3)$, $(t2, t5)$, and $(t3, t5)$. □

To formalize the WSP, we need to introduce some preliminary notions. Given a finite set T of tasks and a finite set U of users, an *execution scenario* (or, simply, a *scenario*) is a finite sequence of pairs of the form (t, u)—also written as $t(u)$—for $t \in T$ 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 $k = 1, \ldots, n$. Among the scenarios in a workflow, we are interested in those that describe successfully terminating executions. Since the notion of successful termination depends on the application, from now on we consider only successfully terminating behaviors scenarios. A *workflow* $W(T, U)$ is a (finite) set of scenarios. To illustrate, consider the TRW in Example 2.1 and let $TRW(T, U)$ be its formalization. Then, among the following scenarios:

$$\eta_1 = t1(a), t2(a), t3(c), t4(a), t5(b); \quad \eta_4 = t1(c), t3(c), t2(a), t5(b)$$
$$\eta_2 = t1(c), t2(a), t3(c), t4(a), t5(b); \quad \eta_5 = t1(a), t2(b), t3(b);$$
$$\eta_3 = t1(a), t3(c), t2(a), t5(b); \quad \eta_6 = t2(b), t3(b), t4(a), t5(b);$$

only $\eta_1, \ldots, \eta_4$ are in $TRW(T, U)$ because they represent sequences of task executions that are compliant with the BPMN in Figure 1; whereas η_5 and η_6 cannot be in $TRW(T, U)$ as the execution of task $t5$ and of task $t1$, respectively, is missing and thus the scenarios are not compliant with the BPMN specification.

Given a workflow $W(T, U)$, an *authorization relation* TA is a sub-set of $U \times T$ where $(u, t) \in TA$ means that u is entitled to execute task t. Following [15], we call *authorized* a scenario η of a workflow $W(T, U)$ according to TA iff (u, t) is in TA for each $t(u)$ in η. For instance, η_1 and η_3 are authorized, whereas η_2 and η_4 are not (since $(c, t1) \notin TA$). An *authorization constraint* over a workflow $W(T, U)$ is a tuple $(t_1, t_2, \bowtie)$ where $t_1, t_2 \in T$ and $\bowtie$ is a sub-set of $U \times U$. For instance, a SoD constraint between tasks t and t' can be formalized as $(t, t', \neq)$ with $\neq$ being the complement of the identity relation over U. A scenario η of $W(T, U)$ *satisfies* the authorization constraint $(t_1, t_2, \bowtie)$ over $W(T, U)$ iff for $t_1(u_1)$ and $t_2(u_2)$ in η we have that $(u_1, u_2) \in \bowtie$. Let K be a (finite) set of authorization constraints, a scenario η satisfies K iff η satisfies each constraint of K. Again following [15], we call *eligible (according to a set K of authorization constraints)* a scenario η of a workflow $W(T, U)$ iff η satisfies K. For instance, η_2 and η_4 are eligible, whereas η_1 and η_3 are not (since there is a SoD between $t1$ and $t2$, but they are executed by the same user, a).

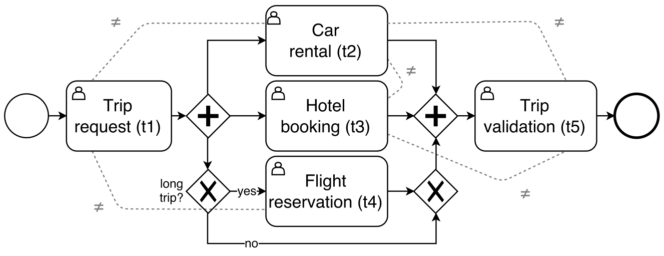
Following [3], we call *security-sensitive workflow* (*SSW*) the triple $(W(T, U), TA, K)$ where $W(T, U)$ is a workflow, TA an authorization relation, and K a (finite) set of authorization constraints. The SSW $(W(T, U), TA, K)$ defines a sub-set of the scenarios in $W(T, U)$ that are both authorized with respect to TA and eligible with respect to K.

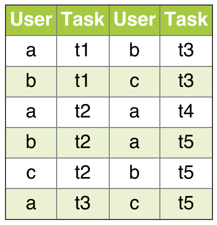
Given a SSW $(W(T, U), TA, K)$, the WSP consists of checking whether there exists an execution scenario in $W(T, U)$ which is both authorized and eligible. To illustrate, consider again the Example 2.1: it is easy to verify that the execution scenarios $\eta_1, \ldots, \eta_4$ presented above are not solutions to the WSP because they are either not authorized or not eligible. On the other hand, $t1(b), t3(c), t4(a), t2(a), t5(b)$ is both authorized (with respect to the TA shown in the table of Figure 1) and eligible (with respect to the five SoD constraints shown in the figure), and is thus a solution to the WSP.

2.1 Multi-objective Workflow Satisfiability

The solvability of the WSP provides some evidence about the possibility to find the best trade-off between security (by satisfying all authorization policies and constraints) and business continuity (by considering only successfully terminating executions). In some situations, this may be too restrictive. For instance, if we delete the lines containing $(a, t2)$ and $(b, t2)$ from the table in Figure 1 specifying the authorization policy (imagine that users a and b cannot access the TRW for some reason), the WSP for the TRW is no more solvable since no user is entitled to execute task $t2$. For the sake of business continuity, it would be important to understand which users can execute task $t2$ (despite none being entitled to do so) to ensure termination while minimizing security issues. This becomes possible as soon as we define the cost of violating an authorization policy and, in the general case, an authorization constraint.

Example 2.2. Recall the authorization policy TA defined in Example 2.1. Let $TA' := TA \setminus \{(a, t2), (b, t2)\}$. Following [12], we introduce a cost function w_P such that $w_P(u, t) = 1$ if $(u, t) \notin TA'$ and $w_P(u, t) = 0$ if $(u, t) \in TA'$. The idea is to associate a cost of





User	Task	User	Task
a	t1	b	t3
b	t1	c	t3
a	t2	a	t4
b	t2	a	t5
c	t2	b	t5
a	t3	c	t5

Figure 1: TRW in BPMN with an associated authorization policy

one to the situation in which a user executes tasks which they are not entitled to execute according to TA'; instead, if users are entitled to execute tasks, then the cost is zero since there is no violation of the policy TA'. To measure the cost of the violations to the policy TA' over the execution of an entire scenario, a possibility is to take the sum of the costs of each violation (if any); formally, $w_P(\eta) = \Sigma_{(u,t)\in\eta} w_P(u,t)$. We are now interested to find the scenarios in TRW (containing task $t4$) that minimize the cost function w_P. For instance, the two scenarios $\eta_1 = t1(b), t3(c), t4(a), t2(a), t5(b)$ and $\eta_2 = t1(b), t3(c), t4(b), t2(a), t5(b)$ are such that $w_P(\eta_1) = w_P(\eta_2) = 1$. Notice that one is the optimal cost as there is no scenario η' such that $w_P(\eta') < 1$.

In the same spirit, we can introduce an additional cost function w_C for the authorization constraints in K such that $w_C(\eta)$ is equal to the cardinality of the set $\{k \in K \mid \eta \text{ does not satisfy } k\}$ where K contains the SoD constraints of Figure 1. Intuitively, w_C counts how many authorization constraints are violated by the scenario under consideration; in the case of the TRW, this means to identify all pairs (u, t) and (u, t') such that $(t, t', \neq) \in K$. We are then interested to find the scenarios in TRW that minimize both cost functions w_P and w_C. There are several reasonable ways to solve this problem. For instance, one can minimize the combined cost of w_P and w_C (e.g., by taking their sum) or minimize each one of them. In the first case, the execution scenario η_2 above has a cost of 2 (1 for the violation w.r.t. TA and 1 for the violation w.r.t. K) whereas η_1 has a cost of 1 (as there is a violation w.r.t. TA and none w.r.t. K). In the second case, η_2 has a cost of $(1, 1)$ whereas η_1 has a cost of $(1, 0)$. Thus, η_1 is an optimal solution with respect to both criteria. □

In some situations, it may be unclear which solutions to consider as optimal. Consider, for instance, the criterion of minimizing the two cost functions at the same time and the situation in which two scenarios have costs $(1, 2)$ and $(2, 1)$, respectively: the former is better than the latter with respect to the first cost function, but it is worse with respect to the second cost function. An obvious question arises: which solution should be preferred? In order to address this kind of questions, we have decided to define a quantitative version of the WSP, by borrowing some notions from the framework of Multi-Objective Optimization (MOO) problems [30].

Indeed, the main goal of MOO techniques is to simultaneously optimize a collection of cost functions. In general, this is impossible since (as shown in the example above) a solution that minimizes one cost may not minimize another. In general, for any non-trivial MOO problem, there is no single solution that is simultaneously optimal for every objective. Instead, there may exist (possibly infinitely) many solutions that can be considered equally good, called Pareto optimal (see, e.g., [30]). Formally, a scenario η^* is Pareto optimal iff there is no scenario $\eta \neq \eta^*$ such that $w_C(\eta) \leq w_C(\eta^*)$, $w_P(\eta) \leq w_P(\eta^*)$, and $w_C(\eta) < w_C(\eta^*)$ or $w_P(\eta) < w_P(\eta^*)$. I.e. a scenario is Pareto optimal if there does not exist another scenario that improves one cost function without detriment to the other.

Several methods have been devised to help the process of choosing one or more solutions among those that are Pareto optimal (see, e.g., [30, 32]); we discuss some of them and show how they relate to quantitative versions of the WSP that have been studied in the literature.

Definition 2.3. Given a SSW $(W(T, U), TA, K)$ with functions w_P and w_C associating scenarios in $W(T, U)$ with the costs of violating the authorization policy TA and the authorization constraints in K, respectively; the *Multi-Objective WSP* (MO-WSP) amounts to

$$\underset{\eta}{\text{minimize}} \quad (\, w_P(\eta), w_C(\eta)\,) \quad \text{subject to} \quad \eta \in \mathcal{S}$$

where $\mathcal{S} \subseteq W(T, U)$ is the set of *scenarios of interest.* □

The MO-WSP is a MOO problem that consists of optimizing—at the same time—the two functions w_P and w_C that measure the costs of violating the authorization policy and the authorization constraints, respectively, of a SSW while considering the sub-set $\mathcal{S}$ of scenarios in $W(T, U)$. The definition of the set $\mathcal{S}$ requires some care as it may be meaningless to solve the MO-WSP for all scenarios in $W(T, U)$ when some of these are executed only as alternatives. To understand why this is so, consider Example 2.1: it is not appropriate to solve the MO-WSP with $\mathcal{S}$ being the set of all scenarios in TRW as those containing task $t4$ are likely to have higher costs than those not containing it; in fact, $t4$ is included in a scenario only when *long trip* is true, and it is not so when *long trip* is false. It would be desirable to solve two distinct MO-WSPs: one for the set of scenarios in TRW when *long trip* is true and another when *long trip* is false. Then, one can compare the resulting solutions and, if appropriate, take their maximum. Similar observations can be found in [14]. We will elaborate on this issue further in Section 3.

Since the MO-WSP is a MOO problem, to solve it, we can reuse the cornucopia of techniques available in the literature (see, e.g., [8, 30, 32]). Many of these transform a MOO problem into one or more optimization problems whose solutions are Pareto optimal (under reasonable additional assumptions). In the rest of this section, we discuss the application of such techniques to the MO-WSP.

Weighted sum. This technique translates the MO-WSP into a standard optimization problem that amounts to minimizing a single cost function defined as the weighted sum of w_P and w_C, i.e. $a \cdot w_P(\eta) + b \cdot w_C(\eta)$, provided that $\eta \in \mathcal{S}$. The constants a and b, called *weights*, model the severity of violating the authorization policy and constraints, respectively. We assume $a > 0$ and $b > 0$ to guarantee that the solution of the transformed problem belongs to the set of Pareto optimal solutions of the original problem.

In the security literature, the use of the weighted sum of w_P and w_C to define a quantitative version of the WSP, called the Valued WSP, has been introduced in [12]. There are two main differences between the MO-WSP and the Valued WSP. First, the set $W(T, U)$ may contain scenarios of various lengths (because of the presence of conditionals) whereas the class of workflows for which the Valued WSP is defined gives rise to scenarios with the same length (as it cannot specify conditional executions). Second, the MO-WSP takes into consideration control-flow constraints whereas the Valued WSP does not. A solution to the Valued WSP is an optimal plan (a function assigning tasks to users), whereas a solution to the MO-WSP is an optimal execution scenario. In general, there are valid plans which cannot become valid execution scenarios, as observed in [11].

The main problem with using the weighted sum technique to solve the MO-WSP is the *a priori* selection of non-arbitrary values for the weights a and b. To make the technique usable in practice, it would be interesting to study the several methods available in the literature to guide the weight selection process (see again [30] for a brief introduction and pointers to the relevant literature) and adapt them to the solution of the MO-WSP.

Lexicographic. This technique is used when assigning values to the weights a and b is difficult (or even impossible) but, according to some qualitative criterion, the order of importance between the cost functions w_P and w_C is clear, reflecting the fact that it is preferable to violate either the authorization policy or the authorization constraints over the other. The first step is to find the solution η^* to the following optimization problem:

$$\underset{\eta}{\text{minimize}} \quad f_1(\eta) \quad \text{subject to} \quad \eta \in \mathcal{S}$$

where f_1 is the first according to the order of importance between the two cost functions w_P or w_C. The second step is to solve another optimization problem

$$\underset{\eta}{\text{minimize}} \quad f_2(\eta) \quad \text{subject to} \quad \eta \in \mathcal{S} \;\&\; f_2(\eta) \leq f_2(\eta^*)$$

where f_2 is the second according to the order of importance between the two cost functions w_P and w_C. The term $f_2(\eta^*)$ in the additional constraint $f_2(\eta) \leq f_2(\eta^*)$ of the second problem represents the optimal value for the first problem. The value $f_2(\eta^*)$ is not necessarily the same as the independent minimum of $f_2(\eta)$.

To the best of our knowledge, this variant of the MO-WSP has never been considered before in the literature about quantitative approaches to the WSP. Because of the difficulties in selecting weights, we believe this to be an interesting alternative to the weighted sum technique discussed above. We show our preliminary experience with solving this variant of the MO-WSP in Section 4.

Bounded cost. The first step of the technique consists of identifying f_1 and f_2 as the more and less (respectively) important function between the cost functions w_P and w_C. Then, it requires to solve the following optimization problem:

$$\underset{\eta}{\text{minimize}} \quad f_1(\eta) \quad \text{subject to} \quad \eta \in \mathcal{S} \;\&\; l \leq f_2(\eta) \leq u$$

where l and u are the lower and upper bounds on f_2. Usually, l is omitted unless the intent is to achieve a goal or fall within a range of values for f_2 rather than to determine a minimum. When omitting l, it is possible to obtain a collection of Pareto optimal solutions by a systematic variation of the upper bound u.

In the security literature, the use of the bounded cost technique to define a quantitative version of the WSP, called Bi-Objective WSP Pareto Optimal (BO-WSP-PO), has been considered in [13]. The main differences with the MO-WSP are similar to those discussed above for the Valued WSP.

Boxed. When even establishing an order of importance between the two cost functions w_P and w_C is difficult or there is no preference in violating the authorization policy or the authorization constraints, it is possible to consider the following two separate single-objective optimization problems

$$\underset{\eta}{\text{minimize}} \quad w_C(\eta) \quad \text{subject to} \quad \eta \in \mathcal{S}$$
$$\underset{\eta}{\text{minimize}} \quad w_P(\eta) \quad \text{subject to} \quad \eta \in \mathcal{S}\,.$$

Indeed, the solutions of these problems are the best possible ones for the two cost functions when considered in isolation and provide bounding box values for the set of Pareto optimal solutions of the original MOO problem (cf. Definition 2.3).

Similarly to the lexicographic method, also this variant of the MO-WSP has never been considered before in the literature about the WSP. Preliminary results about using the boxed technique are in Section 4.

Pareto front. While the boxed optimization method can provide decision-makers with a first idea of what optimal solutions for the MO-WSP look like, an approach returning a set of Pareto optimal solutions is more desirable. In this way, decision-makers can pick one of the available solutions a posteriori rather than a priori as done with the approaches above. In the context of the MO-WSP, a *Pareto front* is a maximal set of Pareto optimal scenarios that are not pairwise weight-equal. A pair of scenarios η and η' is *weight-equal* iff $w_P(\eta) = w_P(\eta')$ and $w_C(\eta) = w_C(\eta')$.

In the security literature, the use of the Pareto front technique to define a quantitative version of the WSP, called Bi-Objective WSP Pareto Front (BO-WSP-PF), has been introduced in [13]. The main differences with the MO-WSP are similar to those discussed above for the Valued WSP.

3 ENCODING THE MO-WSP AS AN OMT PROBLEM

To encode the MO-WSP as an OMT problem, we translate—by using the approach in [22]—the Petri net semantically associated to a BPMN workflow—see, e.g., [18]—to a symbolic representation in (a fragment of) first-order logic. The latter is enriched with the authorization policies and constraints, also expressed in (a fragment of) first-order logic—see, e.g., [1]—together with the cost functions, and used as input to an OMT solver. The model returned by the

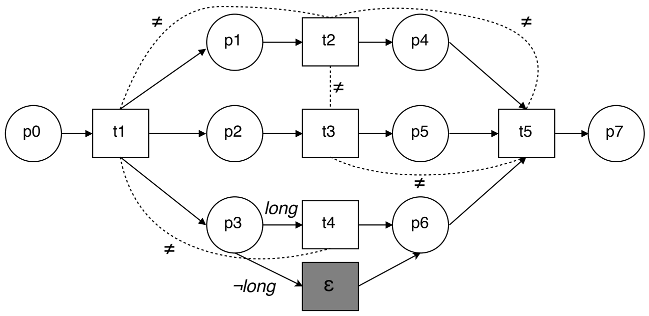


Figure 2: The Petri net for the TRW

solver represents an execution of the workflow that is optimal w.r.t. the input problem.

Background on Petri nets. A tuple (P, T, F) is a *(Place/Transition) Petri net* [16] for P a set of places, T a set of transitions disjoint from P, and $F \subseteq (P \cup T) \times (P \cup T)$ the flow relation such that $F \cap (P \times P) = F \cap (T \times T) = \emptyset$. The *pre-set* ${}^\bullet x$ and the *post-set* $x^\bullet$ of $x \in (P \cup T)$ are the sets $\{y \mid (y, x) \in F\}$ and $\{y \mid (x, y) \in F\}$, respectively. A *marking* m is a mapping from the set P of places to the natural numbers. A transition $t \in T$ is *enabled* by a marking m if $m(p) > 0$ for each $p \in {}^\bullet t$. If a transition $t \in T$ is enabled by a marking m, its occurrence transforms m into m' such that

$$m'(p) \quad := \quad \begin{cases} m(p) - 1 & \text{if } p \in {}^\bullet t \setminus t^\bullet \\ m(p) + 1 & \text{if } p \in t^\bullet \setminus {}^\bullet t \\ m(p) & \text{otherwise.} \end{cases}$$

We write $m \xrightarrow{t} m'$ when t is enabled by marking m with m' being the marking obtained by the occurrence of t. A (finite) sequence $t_1, \ldots, t_k$ of transitions in T is a *(finite) occurrence sequence enabled at marking* m if there exist markings $m_1, \ldots, m_k$ such that $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \cdots \xrightarrow{t_k} m_k$. A marking m' is *reachable* from m when there exists an occurrence sequence enabled at m whose last marking is m'; also written as $m \xrightarrow{t_1, \ldots, t_k} m'$. The Petri net (P, T, F) with the marking m is 1-*bounded* when each place in P contains at most 1 token in any marking reachable from m. Here, we consider only 1-bounded Petri nets as it is known that most business processes have associated such a type of net as their semantics; see, e.g., [18]. Notice that it is possible to express iteration in 1-bounded Petri nets; see again [18].

Example 3.1. A Petri net can be graphically shown as a bipartite graph where places are drawn as circles and transitions as squares that are connected by arrows either from places to transitions or from transitions to places. Figure 2 shows a Petri net corresponding to the BPMN of the TRW at the left of Figure 1. The dashed lines labeled by $\neq$ in the Figure represent authorization constraints, whereas the arrows labeled by '*long*' and '$\neg$*long*' represent conditional branches. Notice that the Petri net in Figure 2 has an "extra" transition, represented by a gray box labeled by ϵ. This is an automatic transition, that is fired instead of $t4$ for short trips. □

Symbolic representation. We show how to represent a Petri net (P, T, F) and an initial marking m_i with a symbolic transition system (S, I, Tr) where S is the set of state variables, I is a symbolic description of m_i, and Tr is a symbolic description of F. We use formulae of (a fragment of) first-order logic to represent I and Tr.

Since we consider only 1-bounded nets, the set S of state variables contains a Boolean variable p for each place $p \in P$ and a Boolean variable d_{ti} for each transition $ti \in T$ for $i = 1, \ldots, K$ and $K > 0$ is the number of transitions in T. The variable p encodes the presence of a token in place p whereas d_{ti} represents the fact that transition ti has been executed. The initial state formula I is induced by the initial marking m_0 as follows:

$$I \quad := \quad \bigwedge_{m_0(p)=1} p \wedge \bigwedge_{m_0(p)=0} \neg p \wedge \bigwedge_{i=1}^{K} \neg d_{ti} .$$

Following [22], we derive Tr from F by taking the conjunction of the following formulae:

$$\bigvee_{i=1}^{K} d_{ti} \tag{1}$$

$$\left(\bigvee_{i=1}^{j-1} d_{ti} \right) \Rightarrow \neg d_{tj} \qquad \text{for } 2 \leq j \leq K \tag{2}$$

$$d_{ti} \Rightarrow \left(\bigwedge_{p \in {}^\bullet ti} p \right) \qquad \text{for } 1 \leq i \leq K \tag{3}$$

$$d_{ti} \Rightarrow \left(\bigwedge_{p \in {}^\bullet ti} \neg p' \wedge \bigwedge_{p \in ti^\bullet} p' \right) \qquad \text{for } 1 \leq i \leq K \tag{4}$$

$$\left(\neg \bigvee_{t \in ({}^\bullet p \cup p^\bullet)} d_t \right) \Rightarrow (p' \Leftrightarrow p) \qquad \text{for each } p \in P \tag{5}$$

where an unprimed (primed) variable in S denotes its content immediately before (after) the execution of the transition. Intuitively, (1) and (2) require that exactly one transition is executed; (3) that only enabled tasks are executed; (4) and (5) that the execution of ti transforms the symbolic representation of a marking m_1 into the symbolic representation of marking m_2 where $m_1 \xrightarrow{ti} m_2$.

It is possible to represent conditionals (as it is the case for the execution of tasks $t4$ and ϵ in Figure 1) by considering an extra (disjoint from S) set Π of Boolean variables and then appending an additional conjunct to the consequent of the implication (3) with the appropriate Boolean expression. For instance, in case of $t4$, (3) is instantiated as $d_{t4} \Rightarrow p3 \wedge long$ whereas for ϵ as $d_\epsilon \Rightarrow p3 \wedge \neg long$.

Adding authorization policies and constraints. It is easy to include an authorization policy TA and a set K of authorization constraints in the symbolic representation introduced above. It is sufficient to add to S a state variable h_{ti} such that $h_{ti}(u)$ holds whenever u has executed the transition ti and assume the availability of a predicate a_{ti} such that $a_{ti}(u)$ holds whenever $(u, ti) \in TA$ for each $ti \in T$. The intuition is that the a_{ti}'s are defined so as to represent the authorization policy TA and do not change over time (there is a rich literature about using logic to represent a variety of authorization conditions; see, e.g., [1]). The initial formula I

is extended by conjoining $\forall u.\neg h_{ti}(u)$ expressing the requirement that users have executed no tasks. The formula *Tr* is modified by conjoining the following two formulae to (1), (2), (3), (4), and (5):

$$d_{ti} \Rightarrow \exists u. \left(\begin{array}{l} a_{ti}(u) \wedge \bigwedge_{\{ti,tj\} \in K} \neg h_{tj}(u) \wedge \\ h'_{ti}(u) \wedge \forall w. w \neq u \Rightarrow h'_{ti}(w) \Leftrightarrow h_{ti}(w) \end{array} \right) \quad (6)$$
$$\text{for } 1 \leq i \leq K$$

$$\bigwedge_{i=1}^{K} \left(\neg d_{ti} \Rightarrow \forall w. h'_{ti}(w) \Leftrightarrow h_{ti}(w) \right) \quad (7)$$

where $\{ti, tj\} \in K$ abbreviates "for each $(ti, tj) \in K$ and for each $(tj, ti) \in K$." Intuitively, (6) says that task ti can be executed when there exists a user u entitled to do so (cf. $a_{ti}(u)$), no SoD constraints in which ti is involved are violated (cf. $\bigwedge_{\{ti,tj\} \in K} \neg h_{tj}(u)$), it is recorded that u has executed ti and no other user has done so (cf. $h'_{ti}(u) \wedge \forall w. w \neq u \Rightarrow h'_{ti}(w) \Leftrightarrow h_{ti}(w)$) and (7) asserts that all the history variables associated to a task that is not executed are unchanged.

An authorization constraint is called user-independent if it does not depend on identities of users. In particular, SoD constraints are user-independent. We observe that, instead of just SoD constraints, we could include arbitrary user-independent constraints (see, e.g., [27]) because of the expressiveness of (the fragment of) first-order logic that we use, as explained in [19]. Also, notice that, since the set U of users is finite, the existential and universal quantifiers in the formulae above are equivalent to disjunction and conjunction, respectively, over U. Below, we call *extended* a Petri net with authorization policies, authorization constraints, or additional Boolean variables to simplify the representation of control-flow constructs in BPMN.

Bounded Model Checking (BMC). Before explaining how to encode the MO-WSP into an OMT problem, we discuss how BMC [7] relates to the WSP. To this end, observe that the satisfiability of the formula

$$[I(S_0)]_U \wedge [Tr(S_0, S_1)]_U \wedge \cdots \wedge [Tr(S_{\tau-1}, S_\tau)]_U \wedge G(S_\tau) \quad (8)$$

amounts to establishing the reachability in $\tau \geq 1$ steps of the goal formula G from the initial formula I by means of the transition formula *Tr*. In (8), the expressions $I(S_0)$ and $G(S_\tau)$ denote the formulae obtained from I and G, respectively, by replacing the variables in S and G with renamed copies with subscripts 0 and τ, respectively; the expression $Tr(S_{t-1}, S_t)$ denotes the formula obtained from *Tr* by replacing the variables in S and their primed versions with renamed copies with subscripts $t-1$ and t, respectively, for $t = 1, \ldots, n$; finally, $[X]_U$ denotes the formula obtained from X by expanding the universal or existential quantification over the set U of users as a (finite) conjunction or disjunction (respectively) over U for X being I or $Tr(S_{t-1}, S_t)$. The idea to construct formula (8) is to make "timed" copies of the state variables so that the variables in S_t represent the t-th state in a bounded execution of length τ.

When (S, I, Tr) encodes an (extended) 1-bounded Petri net corresponding to a security-sensitive workflow $(W(T, U), TA, K)$ and G characterizes the set of final markings, it is clear that (8) is satisfiable iff the security-sensitive workflow has an execution of $\tau \geq 1$ steps, i.e. a scenario of the form $t_1(u_1), \ldots, t_\tau(u_\tau)$ such that $m_i \xrightarrow{t_1, \ldots, t_\tau} m_f$ for m_i an initial marking and m_f a final marking. By increasing the value τ, we can explore the whole set $W(T, U)$ of execution scenarios and check if they are both eligible and authorized. To mechanize this process, we need a Satisfiability Modulo Theories (SMT) solver, or simply a SAT solver after encoding (8) into a purely Boolean formula (which is possible because the set U of users is finite). Furthermore, it is possible to reconstruct an authorized and eligible scenario of length τ from the assignment returned (if the case) by the solver.

Symbolic representation of scenarios of interest. It is easy to symbolically represent the set $\mathcal{S}$ of scenarios of interest in $W(T, U)$ (recall Definition 2.3) by means of a first-order formula. To illustrate, consider the TRW in Example 2.1: by simply conjoining the formula *long* to the corresponding instance of (8), it is possible to consider only those scenarios for which the trip requires the execution of task $t4$. In general, given the set $\mathcal{S}$ of scenarios of interest, it is possible to build a formula $[\mathcal{S}_t]_\Pi$ corresponding to a Boolean assignment to the variables in the set Π representing the conditionals of the 1-bounded Petri net for each time instant $t = 0, \ldots, \tau$. It is then sufficient to conjoin the formula $\bigwedge_{t=0}^{\tau} [\mathcal{S}_t]_\Pi$ to (8).

Causal nets. We now identify a sub-class of 1-bounded Petri nets for which it is possible to solve the WSP by checking the satisfiability of (8) for finitely many values of τ.

A *causal net* [16] is a Petri net (P, T, F) satisfying the following properties: (i) F is acyclic (i.e. no path with at least two elements leads from an element to itself), (ii) for each $p \in P$, the cardinality of both $^\bullet p$ and $p^\bullet$ is at most one, (iii) only finitely many places $p \in P$ have an empty pre-set, (iv) for each transition $t \in T$, both $^\bullet t$ and $t^\bullet$ are finite and non-empty, and (v) for each element $x \in P \cup T$, only finitely many different paths lead to x. A causal net induces a partial order $\geq$ on its elements as follows: $x \geq x'$ iff there exists a path leading from x to x'. An element x is *maximal* (*minimal)* with respect to $\geq$ iff there is no element x' such that $x' \geq x$ ($x \geq x'$, respectively) and $x \neq x'$. A *canonical initial marking* m_i (*final marking* m_f) of a causal Petri net assigns to each maximal/minimal (w.r.t. $\geq$) place one token and no token to all other places. Notice that m_f enables no transition. Three important properties of a causal net are that (P1) it is 1-bounded when considered with its canonical initial marking, (P2) each transition of an occurrence sequence can eventually occur, and (P3) no transition occurs more than once in an occurrence sequence [16]. Because of (P1), we can derive a symbolic representation from a causal net as described above. (P2) can be lifted to augmented causal nets provided that any user entitled to execute a task does not delay its execution once this has become enabled. (P2) and (P3) set an upper bound on the length of the scenarios of an augmented causal net to the cardinality of the largest set of transitions that are totally ordered with respect to $\geq$. Because of (P1), (P2), and (P3), the following method is a decision procedure for the WSP. For $\tau = 0, \ldots, L$, if (8) is satisfiable, return that the WSP is solvable. If $\tau = L$ and no formula has been found satisfiable, return that the WSP is unsolvable.

In the rest of the paper, we consider only causal nets. We notice that some of the most important control flow patterns in BPMN for parallel and non-deterministic/conditional executions can be expressed in this class of nets; see, e.g., [18]. One of the most important omissions is iteration (as considered in, e.g., [14]), which we leave as future work. The advantage of considering causal nets is a simplified symbolic representation of the set $\mathcal{S}$ of the scenarios of interest. In fact, the conditionals (such as *long* in TRW) can be executed only once and we can drop the subscript t from the formula $[\mathcal{S}_t]_\Pi$ corresponding to their Boolean assignment (see paragraph 'Symbolic representation of scenarios of interest' above). It is thus sufficient to conjoin the formula $[\mathcal{S}]_\Pi$ to (8).

3.1 From the MO-WSP to the OMT Problem

Optimization Modulo Theories (OMT) is an extension of SMT which aims to solve the problem of finding a model for an input formula φ which is optimal with respect to one or more objective functions.

An important sub-case of OMT is the *(weighted partial) Min-SMT problem* that can be stated as follows. A clause is a disjunction of literals, i.e. atoms or their negations, over some theory. A soft clause is a pair consisting of a clause with a weight (a non-negative number). A hard clause is a pair consisting of a clause with an infinite weight. Given a formula of the form $\varphi_h \wedge \varphi_s^1 \wedge \cdots \wedge \varphi_s^q$, where φ_h is a conjunction of hard clauses and φ_s^k is a conjunction of soft clauses for $k = 1, \ldots, q$, the *(weighted partial) Min-SMT problem* amounts to finding a model of φ_h that minimizes the tuple $(w_s^1, \ldots, w_s^q)$ of weights obtained by taking the sum of the weights of the satisfied soft clauses in $\varphi_s^1, \ldots, \varphi_s^q$, respectively.

We now explain how to reduce the MO-WSP to a Min-SMT problem. The idea is to consider the formula (8) $\wedge\, [\mathcal{S}]_\Pi$, take $q = 2$ sets φ_s^1 and φ_s^2 of soft clauses representing the constraints imposed by the authorization policy and the authorization constraints, respectively, and associate to each clause in these sets a weight derived from the functions w_P and w_C. The details of the reduction can be summarized in the following six steps.

(1) From an instance of the MO-WSP (c.f. Definition 2.3), build the formula φ as the conjunction of (8) and $[\mathcal{S}]_\Pi$.
(2) Replace all the atoms of the forms $a_{ti}(u)$ and $h_{tj}(u)$, for some $u \in U$, that occur in φ as instances of the atoms in (6) with fresh Boolean variables b_a and b_h; let φ' be the resulting formula and *Abs* be the set of all pairs $(b_a, a_{ti}(u))$ and $(b_h, h_{tj}(u))$.
(3) Transform φ' into an equisatisfiable conjunction φ_h of clauses by using well-known logical transformations (see, e.g., [23]); the size of φ_h can be linear in the size of φ'.
(4) Take φ_s^1 to be a conjunction of clauses of the form $\neg b_a \vee a_{ti}(u)$ for each $(b_a, a_{ti}(u))$ in *Abs* and φ_s^2 to be a conjunction of clauses of the form $\neg b_h \vee h_{tj}(u)$ for each $(b_h, h_{tj}(u))$ in *Abs*.
(5) Assign weights w_{ai} to each b_a and w_{hj} to each b_h, such that $w_s^1 = \sum w_{ai}$ and $w_s^2 = \sum w_{hj}$ encode the cost functions $w_P(\eta)$ and $w_C(\eta)$, respectively. Notice that the cost functions are defined on execution scenarios, whereas we need "local" weights w_{ai} and w_{hj}. As a reasonable simplification, we assume that $w_P(\eta)$ and $w_C(\eta)$ are linear combinations of terms that depend only on a task t being executed, the user u executing t, and the execution history of u and another task t'[1]. Then, we can map each term in a cost function to a weight w_{ai} or w_{hj} or ask the user directly for the weights. Recall the cost functions in Example 2.2: in this case, $w_{ai} = w_{hj} = 1$ for each b_a and b_h.
(6) Find a model of $\varphi_h \wedge \varphi_s^1 \wedge \varphi_s^2$ that minimizes the pair (w_s^1, w_s^2). The model can be used to compute one or more scenarios that are solutions to the MO-WSP (how to do this is illustrated in an example below).

Observe that formula φ' computed at step 2 is an over-approximation of (8) as it abstracts away from the constraints imposed by the authorization policies and authorization constraints (that are replaced by fresh Boolean variables), thus guaranteeing only the control-flow constraints of the SSW. Also notice that, since the fresh Boolean variables b_a's and b_h's are constrained by the soft clauses in φ_s^1 and φ_s^2, respectively, the solver is free to choose their truth value to minimize the pair of weights (w_s^1, w_s^2) according to one of the criteria discussed in Section 2.1.

3.2 Parallel encoding

The transition formula *Tr* presented above uses an interleaving semantics for the execution of tasks in a workflow, i.e. it assumes that only one task is executed at each step τ (parallel tasks can be executed in any order).

We now explore an optimized encoding of *Tr* based on a $\forall$-step semantics, as defined in [22], which exploits the parallelism in workflow specifications to model the execution of (possibly) several tasks in one step, thus compressing the number of steps in *Tr*. It also uses properties of causal nets to encode in each step only those tasks that may actually be executed. This encoding is slightly more complex to understand and requires a pre-processing phase, but it is also faster for an OMT solver (see Section 4.3).

To formalize the encoding, we need to introduce some preliminary notions. An *anti-chain* is a sub-set of the elements in a partial order $\geq$ in which no two distinct element are comparable, i.e. neither $t \geq t'$ nor $t' \geq t$ for every pair of elements in the sub-set. A *cut* of a causal net is a maximal (w.r.t. inclusion) set of elements in $P \cup T$ that are pairwise not ordered by $\geq$ (where $\geq$ is the partial order induced by the net).

Property 1. *Let N be the causal net associated to a security-sensitive workflow $(W(T,U), TA, K)$. Then, if C is a cut of N, then $C \cap T$ is a maximal anti-chain of $\geq$.*

To encode the fact that more than one task can be executed in a given step of *Tr*, we can reuse formulae (1) and (3–5). We only need to change formula (2), the one that says that only one task is executed in each step. The revised version of (2) is

$$\left(\bigvee_{i=1}^{j-1} dt_i\right) \Rightarrow \bigwedge_{p \in t_i{}^\bullet \cap {}^\bullet t_j} \neg dt_j \qquad \text{for } 2 \leq j \leq K \tag{9}$$

which means that possibly all tasks can be executed at each step. This formula delegates to the OMT solver the job of finding a satisfying assignment that respects the control-flow constraints of the workflow.

[1]Defining cost functions on scenarios is more general, as it allows us to have different costs for specific user-task pairs or costs that depend on the order of execution of the tasks. It is possible to define complex cost functions as local weights by encoding the clauses b_a and b_h using, e.g., if-then-else conditions, but that increases the complexity both for humans to express and for solvers to find optimal models.

We obtain the sets of tasks that can be executed in parallel at each step of Tr in (8) by computing the lattice of maximal anti-chains (see, e.g., [24]) and traversing it breadth-first. We construct the BMC encoding by starting from the bottom of the lattice and, in each level (which corresponds to a step in Tr), we encode only the tasks in the anti-chain of that level in (9). If there is more than one anti-chain in a level, this is encoded as an exclusive disjunction. The number of steps in Tr becomes equal to the number of maximal anti-chains, instead of equal to the number of tasks in the workflow.

After discharging the BMC formula to an OMT solver, the resulting model is a compact representation of several possible interleaving executions, which can be linearized. We illustrate how this is done by an example.

Example 3.2. We consider TRW and assume that *long* is true, i.e. we want to execute $t4$ (for the sake of simplicity, we omit the conjunct *long* in the formulae below). In this case, we have 3 anti-chains: $\{t1\}$, $\{t2, t3, t4\}$, and $\{t5\}$, each representing a set of tasks that can be executed in parallel. Using the original interleaving encoding, we would have modeled the TRW with 5 steps, one for each task (considering that either $t4$ or ϵ can be executed). With the parallel encoding, we model it using 3 steps, one for each anti-chain. Step 0 is obtained by applying (1), (9), (3), (4), and (5) only to task $t1$:

$$d^0_{t1} \wedge \left(\bigwedge_{j=2}^{5} d^0_{t1} \Rightarrow \neg d^0_{tj} \right) \wedge \left(d^0_{t1} \Rightarrow p0^0 \right) \wedge$$

$$\left(d^0_{t1} \Rightarrow \neg p0^1 \wedge p1^1 \wedge p2^1 \wedge p3^1 \right) \wedge \left(\neg \bigvee_{j=2}^{5} d^0_{tj} \Rightarrow \bigwedge_{j=4}^{7} pj^1 = pj^0 \right),$$

where the first line indicates that only $t1$ can be executed and it must be enabled, whereas the second line specifies both the variables that are updated and those that remain unchanged. Step 2 is obtained by applying the same formulae only to $t5$:

$$d^2_{t5} \wedge \left(\bigwedge_{j=1}^{4} d^2_{t5} \Rightarrow \neg d^2_{tj} \right) \wedge \left(d^2_{t5} \Rightarrow p4^2 \wedge p5^2 \wedge p6^2 \right) \wedge$$

$$\left(d^2_{t5} \Rightarrow \neg p4^3 \wedge \neg p5^3 \wedge \neg p6^3 \wedge p7^3 \right) \wedge \left(\neg \bigvee_{j=1}^{4} d^2_{tj} \Rightarrow \bigwedge_{j=0}^{3} pj^3 = pj^2 \right)$$

and the meaning of the formula is similar to the previous one (replacing $t1$ by $t5$). Step 1 encompasses $t2$, $t3$, $t4$, and ϵ and is encoded as follows:

$$\left(\bigvee_{j=2}^{4} d^1_{tj} \vee d^1_{\epsilon} \right) \wedge \left(\bigvee_{j=2}^{4} d^1_{tj} \vee d^1_{\epsilon} \Rightarrow \neg d^1_{t1} \wedge \neg d^1_{t5} \right) \wedge$$

$$\left(\begin{array}{r} d^1_{t2} \Rightarrow p1^1 \wedge \\ d^1_{t3} \Rightarrow p2^1 \wedge \\ d^1_{t4} \Rightarrow long \wedge p3^1 \wedge \\ d^1_{\epsilon} \Rightarrow \neg long \wedge p3^1 \end{array} \right) \wedge \left(\begin{array}{r} d^1_{t2} \Rightarrow \neg p1^2 \wedge p4^2 \wedge \\ d^1_{t3} \Rightarrow \neg p2^2 \wedge p5^2 \wedge \\ d^1_{t4} \Rightarrow \neg p3^2 \wedge p6^2 \wedge \\ d^1_{\epsilon} \Rightarrow \neg p3^2 \wedge p6^2 \end{array} \right) \wedge$$

$$\left(\neg (d^1_{t1} \vee d^1_{t5}) \Rightarrow p0^3 = p0^2 \wedge p7^3 = p7^2 \right).$$

A model for the BMC formula constructed as above assigns True to the following variables (and False to all the others):

$$d^0_{t1}, d^1_{t2}, d^1_{t3}, d^1_{t4}, d^2_{t5}, p0^0, p1^1, p2^1, p3^1, p4^2, p5^2, p6^2,$$

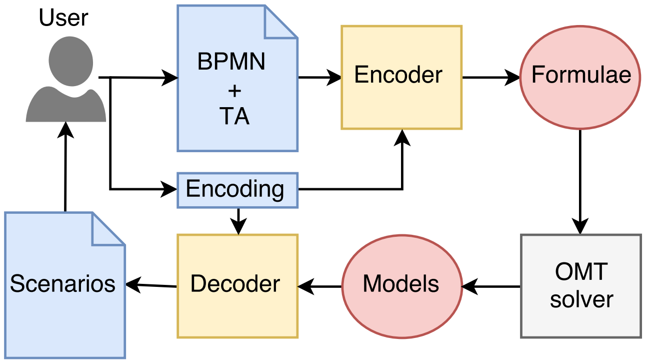


Figure 3: Architecture of the implementation

$$h^0_{t1}(b), h^1_{t2}(a), h^1_{t3}(c), h^1_{t4}(a), h^2_{t5}(b), long$$

which represents the following scenarios:

$$\begin{aligned} \eta_1 &= t1(b), t2(a), t3(c), t4(a), t5(b); \\ \eta_2 &= t1(b), t2(a), t4(a), t3(c), t5(b); \\ \eta_3 &= t1(b), t3(c), t2(a), t4(a), t5(b); \\ \eta_4 &= t1(b), t3(c), t4(a), t2(a), t5(b); \\ \eta_5 &= t1(b), t4(a), t2(a), t3(c), t5(b); \\ \eta_6 &= t1(b), t4(a), t3(c), t2(a), t5(b). \end{aligned}$$

□

4 EVALUATION

We implemented a prototype and experimented with two sets of benchmarks: real-world workflows and synthetic, randomly generated, ones. The benchmarks contain only causal nets and sets of scenarios of interest that refer to parallel executions of tasks (e.g., those either containing $t4$ or not in TRW).

4.1 Implementation

Figure 3 shows the architecture of our implementation. The user inputs a *BPMN model* of the workflow (including authorization constraints), an authorization policy, and an *Encoding*. The Encoding includes the option of semantics to use for the transition system (interleaving or parallel), the costs associated to violating the policy and constraints, and the optimization mode. These artifacts are given to an *Encoder*, which translates the model and options to a set of formulae that can be fed to an *OMT solver*. The solver outputs one or more optimal models, which are passed to a *Decoder* module to transform them into actual execution scenarios that are presented to the user.

We used Python, PySMT [25], and SageMath [17] to implement the Encoder and Decoder and interface with the OMT solvers. We support the solvers OptiMathSAT [32] and Z3 [8]. Both natively support Boxed, Lexicographic, and Pareto optimization. Weighted sum and Bounded cost optimization can be easily encoded, but we skip this in the experiments for the sake of space.

4.2 Real-world workflows

We applied our approach to two workflows inspired by real-world examples[2] (ITIL and ISO, shown in Figure 4), besides the TRW presented before. These examples include all the basic workflow control-flow patterns [33] (sequential, parallel, and exclusive executions). Each workflow has 10 users authorized for each task, except for the last task, which has no authorized users, so that the workflow becomes unsatisfiable.

Table 1 shows the results.[3] For each workflow instance, we used 12 configurations, obtained from the combinations of: 2 encodings ('Interleaving' and 'Parallel'); 2 OMT solvers ('OptiMathSat' and 'Z3'); and 3 modes of optimization ('Lexic.' for Lexicographic, 'Boxed', and 'Pareto'). Each configuration was executed 10 times and we report the median execution time (in seconds).

All solutions are found in less than 1 second (most in less than 0.5 seconds). It is easy to see that the parallel encoding has a superior performance in every case (far superior, in many cases). It is also clear that Pareto is the slowest optimization mode, while Boxed and Lexicographic have almost the same performance, and that Z3 is faster than OptiMathSAT when both are using the same optimization mode, except for two cases (ITIL and ISO with the interleaving encoding and Boxed optimization).

These results show that solving the MO-WSP with optimization modulo theories is feasible for average instances found in real-world use cases. To further test the effects of the parallel encoding and the scalability of the techniques in larger instances, we consider synthetic benchmarks.

4.3 Synthetic benchmarks

We adapted the random workflow generator used in [6] to generate workflows with *TA*'s and SoD constraints that are not satisfiable. The tool generates workflows with a given number of tasks n, $10n$ users, *TA*'s with up to 5 tasks with no authorized users, and at least 1 unsatisfiable SoD constraint. There are two more parameters: d is the number of user-task pairs in *TA* out of the possible number (e.g., with 100 users and 10 tasks, there could be 1000 user-task pairs; if $d = 10\%$, the tool generates 100 of those); e is similar and specifies the (relative) number of authorization constraints out of the number of tasks. The generated workflows have sequential and parallel tasks (for every task in a workflow, there is a 15% probability to branch).

Encodings. To confirm that the parallel encoding is superior to the interleaving encoding even as the size of the input workflow grows, we ran an experiment where we generated workflows with 10, 13, 16, 19 tasks (100, 130, 160, 190 users). We tested with the same 12 configurations as before (2 encodings, 2 solvers, 3 optimization modes) and a fixed $d, e = (10\%, 10\%)$. Again, each configuration was executed 10 times and we report the median execution time (in seconds).

Table 2 shows the results. The parallel encoding is always superior and the gains obtained with it become more substantial as the workflow size grows in most configurations. The time to compute the lattice of maximal anti-chains used for the parallel encoding is negligible and not reported separately (it is already included in the time reported for the execution with the parallel encoding).

For workflows with 20 or more tasks, we started observing timeouts in the execution of the solvers when using the interleaving encoding (we set the timeout to 1 hour). To show the scalability of the technique beyond 20 tasks, we used only the parallel encoding and expanded our tests to include different d, e configurations.

Scalability. We generated workflows with 10 to 30 tasks (100 to 300 users) and the d, e configurations $\{(10\%, 10\%), (10\%, 30\%), (20\%, 10\%), (20\%, 30\%)\}$, as done in [13]. For each configuration, we generated 10 random workflows and solved the MO-WSP using the parallel encoding, both OMT solvers and the three optimization modes (Lexicographic, Boxed, and Pareto).

Figure 5 shows the results. Each graph shows the results of one OMT solver in one optimization mode, where the x axes show the number of tasks, the y axes show the median time to solve an instance, and each line represents a (d, e) configuration. Notice that the scales are different for each graph.

There is an (expected) exponential growth in the time to solve the MO-WSP, nevertheless, even for workflows of up to 30 tasks and 300 users—large configurations for realistic use cases—the solution is found in under 30 seconds. The time to compute a lattice of maximal anti-chains and obtain the scenarios from the generated models is negligible (less than 100 ms) and thus not reported. It

[2]http://www.signavio.com/reference-models/

[3]All the experiments in this paper were performed on a laptop with a quad-core 2.6GHz Core i7 processor and 16GB of RAM running Ubuntu 16.04.

Table 1: Results for real-world workflows (in seconds)

	OptiMathSAT			Z3		
Encoding	Lexic.	Boxed	Pareto	Lexic.	Boxed	Pareto
	TRW					
Interleaving	0.178	0.302	0.192	0.045	0.046	0.155
Parallel	0.034	0.032	0.035	0.019	0.019	0.024
	ITIL					
Interleaving	0.498	0.280	0.615	0.034	0.033	0.255
Parallel	0.051	0.056	0.061	0.021	0.021	0.031
	ISO					
Interleaving	0.502	0.301	0.731	0.045	0.045	0.255
Parallel	0.049	0.048	0.051	0.025	0.025	0.037

Table 2: Results for synthetic workflows (in seconds)

	OptiMathSAT			Z3		
Encoding	Lexic.	Boxed	Pareto	Lexic.	Boxed	Pareto
	n = 10 tasks					
Interleaving	1.219	1.325	1.209	0.429	0.083	0.708
Parallel	0.117	0.125	0.123	0.030	0.024	0.061
	n = 13 tasks					
Interleaving	5.353	6.335	5.948	3.330	32.093	3.771
Parallel	0.270	0.224	0.234	0.054	0.045	0.078
	n = 16 tasks					
Interleaving	19.058	18.021	17.721	5.243	157.120	28.551
Parallel	1.053	0.699	0.516	0.158	0.110	0.291
	n = 19 tasks					
Interleaving	71.607	50.214	128.333	5.496	216.106	50.075
Parallel	2.062	0.906	1.127	0.298	0.140	1.224

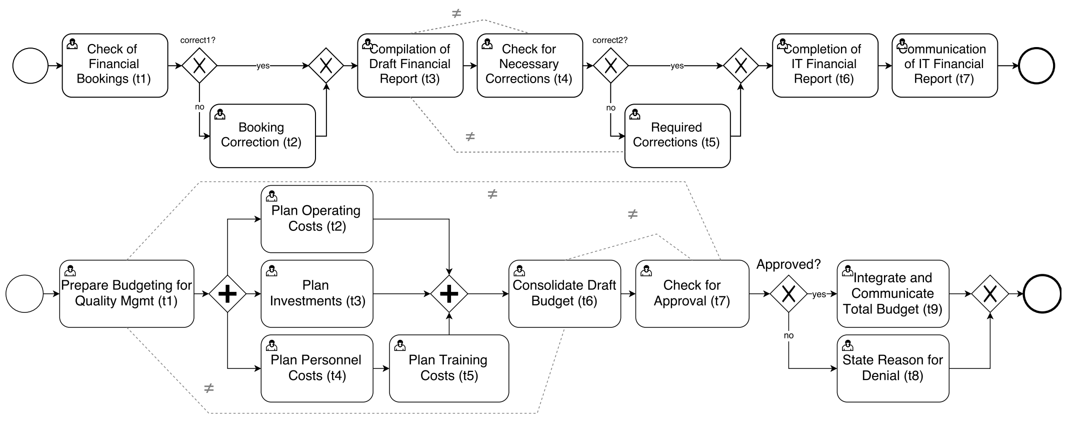

Check of Financial Bookings (t1)
correct1?
yes
no
Booking Correction (t2)
Compilation of Draft Financial Report (t3)
Check for Necessary Corrections (t4)
correct2?
yes
no
Required Corrections (t5)
Completion of IT Financial Report (t6)
Communication of IT Financial Report (t7)
≠
≠
Prepare Budgeting for Quality Mgmt (t1)
Plan Operating Costs (t2)
Plan Investments (t3)
Plan Personnel Costs (t4)
Plan Training Costs (t5)
Consolidate Draft Budget (t6)
Check for Approval (t7)
Approved?
yes
no
Integrate and Communicate Total Budget (t9)
State Reason for Denial (t8)
≠
≠
≠

Figure 4: ITIL process (top) and ISO process (bottom) in extended BPMN.

is also clear that Z3 is faster than OptiMathSAT in almost every instance.

4.4 Discussion

In many cases, off-the-shelf solvers provide greater flexibility at the cost of decreased performance when compared to ad hoc algorithmic solutions, since the latter can be optimized for specific applications. However, in [27] it is shown that the difference in performance for moderate-size WSP instances can be practically insignificant if appropriate modelling of constraints is used. In our experiments, off-the-shelf solvers show good performance, however more experiments would be needed to confirm that our solution can outperform bespoke algorithms for the kinds of WSPs considered.

The algorithmic solution that is closest to our work is that of [13]. However, it is hard to directly compare our experiments with those reported in [13] because the settings are different. First, they consider a wide family of user-independent constraints—i.e. those constraints whose satisfaction does not depend on the identity of the users—whereas we only experiment with SoD constraints. Second, the platforms on which the experiments were run are different, e.g., they do not exploit concurrency, whereas the solvers that we use do exploit this feature. Nevertheless, to give an idea of the orders of magnitude, while our average worst case execution time for workflows of 30 tasks was under 30 seconds, they report times of 10^3 seconds in the worst case. On the other hand, their best case scenarios (for workflows of 10 tasks) run in 10^{-4} seconds, whereas our best performance for a similar configuration was 10^{-2} seconds. A systematic comparison using the available code and benchmarks of [13] is planned as future work.

5 RELATED WORK

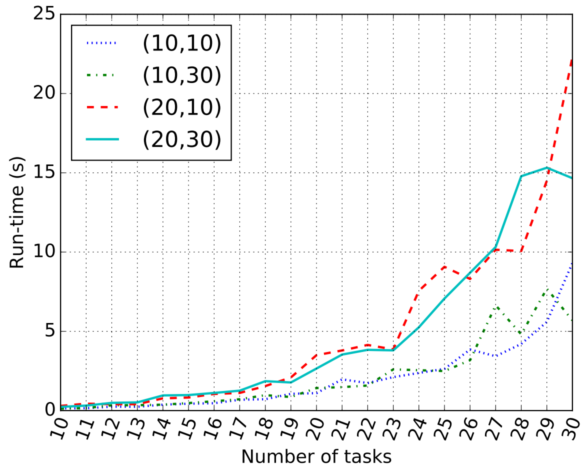
The seminal work of Bertino et al. [5] described the specification and enforcement of authorization constraints in workflow management systems. Wang and Li [34] showed that the WSP is NP-complete even with simple constraints and reduced the problem to SAT. See [20, 26] for a survey on workflow satisfiability approaches. Bertolissi et al. [6] presented a solution to the run-time WSP that relies on pre-computing all eligible execution scenarios of a security-sensitive workflow as a symbolic reachability graph. This graph is refined in [21] to find execution scenarios that satisfy properties defined by the user. It is not possible to reuse the solutions in [6, 21] to solve the MO-WSP because the pre-computed graphs do not consider constraint violations.

Basin et al. [4] studied how to optimally modify an authorization policy to render a workflow instance satisfiable, by associating a cost to each possible change to the policy. On the other hand, Crampton et al. [12, 13] first studied how to find minimal violating assignments of users to tasks, without changing the policy. They first defined the Valued WSP [12] and later the BO-WSP [13], then solved both using a bespoke algorithm and showed that their solution is superior to a mixed integer programming approach in terms of performance. The authors also showed how to solve two related problems by encoding them as cost functions: the quantitative resiliency problem [29], which amounts to assigning a probability to the successful termination of a workflow even in the absence of some users; and the Cardinality-constrained Minimum User Problem (CMUP) [31], which consists in finding the minimum number of users required to satisfy a workflow instance. The main difference between our work and Crampton et al. [12, 13] is that we consider an ordered execution of workflows, whereas they take as solution a valid plan, which is an unordered assignment of tasks to users. They also considered user-independent constraints in their experiments, which we did not implement, but, as already observed in Section 3, can be expressed in the fragment of first-order logic that we use.

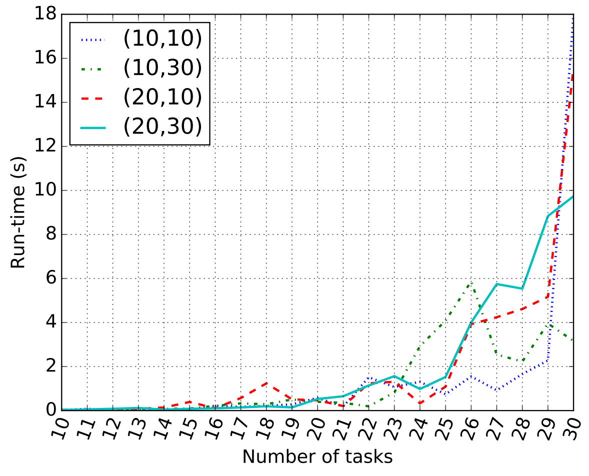
Crampton et al. [14] extended their algorithmic solution to support conditional workflows with release points—which specify that a constraint may be active only for some scenarios—by splitting a workflow instance into many deterministic ones. We believe that release points can also be incorporated in our solution by using an approach similar to [4]; we leave this to future work. A challenge is to adapt the parallel encoding of Section 3.2 to this generalized problem so to have better scalability. For this, we believe that the techniques in [22] can be useful.

6 CONCLUSION

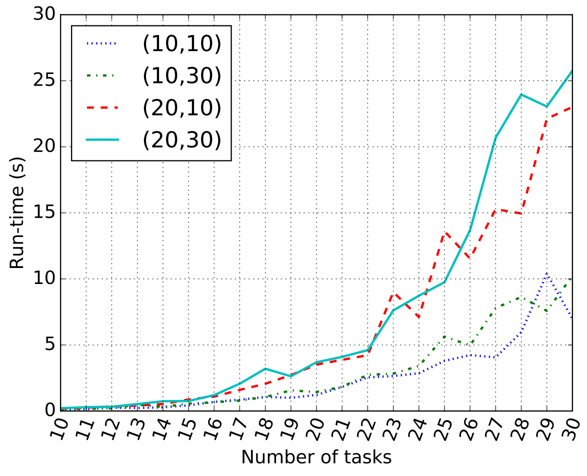
We have motivated, defined, and solved the MO-WSP. This work is the first to consider quantitative solutions to the WSP with an



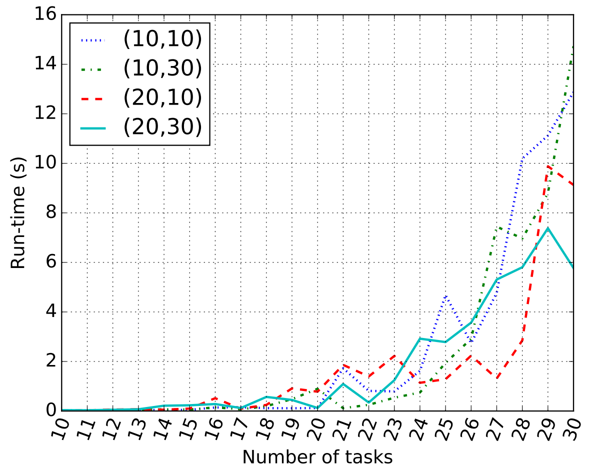
(a) OptiMathSAT/Lexicographic



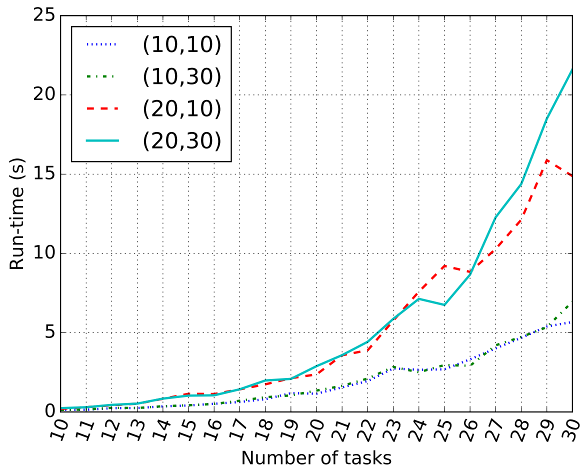
(b) Z3/Lexicographic



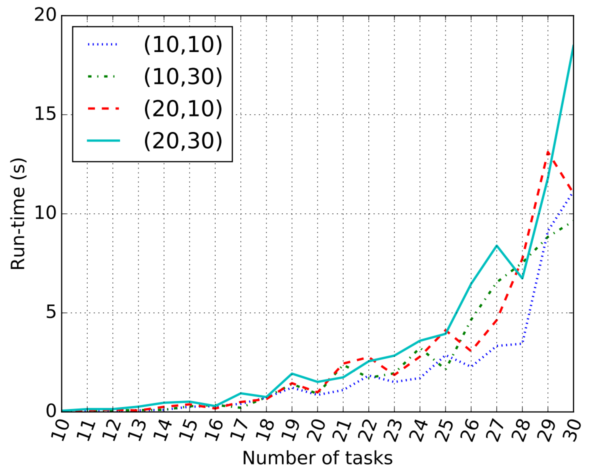
(c) OptiMathSAT/Boxed



(d) Z3/Boxed



(e) OptiMathSAT/Pareto



(f) Z3/Pareto

Figure 5: Scalability benchmarks

ordered execution of tasks, i.e., without abstracting the control-flow constraints, and control-flow patterns such as alternative execution. Our solution, based on the use of off-the-shelf OMT solvers is flexible enough to handle several version of the problem by simply alternating between optimization modes.

The solution was also implemented and evaluated on real and synthetic instances of the problem, showing good performance due to the use of an ingenious encoding of the problem that exploits the parallel executions of tasks in the workflow.

6.1 Future work

We plan to investigate a generalization of the MO-WSP that considers all scenarios at the same time, not restricting to the set of scenarios of interest as done in Definition 2.3. The idea is to return either all optimal solutions associated to alternatives (e.g., the set of optimal solutions for the scenarios including $t4$ and another set for those not including $t4$ in TRW) or the maximum of the optimal solutions. We intend to generalize the approach in Section 3 to solve also this problem as follows. First, use the encoding in Section 3.1 by taking (8) only as the formula φ (i.e. disregarding the formula $[\mathcal{S}]_\Pi$ representing the set $\mathcal{S}$ of scenarios of interest) and solve the resulting optimization problem. Take the assignment of the Boolean variables in Π and negate them; let δ be the resulting formula. Then, take the conjunction of (8) and δ as φ and solve the new (refined) OMT problem: the solver will search for optimal solutions that refer to an assignment of the variables in Π that is different from the previous one. We repeat the process until no more assignments to the variables in Π are found. The main challenge to make the approach practical is to reduce the number of OMT problems to solve in the worst case, which is equal to the number $2^{|\Pi|}$ of alternative execution scenarios induced by the Boolean variables in Π. To this end, we plan to develop heuristics to synthesize formulae expressing the fact that sequences of tasks contained in alternative scenarios are equivalent with respect to the costs considered. Such formulae will be conjoined to (8) to hopefully avoid the complete enumeration of the exponential number of alternative scenarios.

As already mentioned throughout the paper, we intend to study how to support iterations and release points. Another line of future work is to consider a version of the MO-WSP where deviations from the modeled control-flow are allowed. In practice, control-flow deviations are common in, e.g., healthcare systems [2] and finding executions that are optimal w.r.t. control-flow, authorization policies, and authorization constraints may further expand the applicability of our technique.

Finally, we intend to integrate our work into a workflow management system, so that users can get solutions with the push of a button in an integrated environment (as done for run-time monitoring in [9]).

ACKNOWLEDGEMENTS

We thank the reviewers for their useful comments and criticisms that significantly contributed to improvement of the manuscript.

REFERENCES

[1] M. Abadi. *Logic in Access Control (Tutorial Notes).* Springer, 2009.

[2] A. Adriansyah, B. van Dongen, and N. Zannone. Controlling break-the-glass through alignment. *ASE Science Journal*, 2(4):198–212, 2013.

[3] A. Armando and S.E. Ponta. Model checking of security-sensitive business processes. In *Proc. of FAST*, 2009.

[4] D. Basin, S.J. Burri, and G. Karjoth. Optimal workflow-aware authorizations. In *Proc. of SACMAT*, 2012.

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

[6] 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*, 2015.

[7] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *Proc. of TACAS*, 1999.

[8] N. Bjørner, A. Phan, and L. Fleckenstein. νz - an optimizing SMT solver. In *Proc. of TACAS*, 2015.

[9] 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*, 2016.

[10] J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proc. of SACMAT*, 2005.

[11] J. Crampton and G. Gutin. Constraint expressions and workflow satisfiability. In *Proc. of SACMAT*, 2013.

[12] J. Crampton, G. Gutin, and D. Karapetyan. Valued workflow satisfiability problem. In *Proc. of SACMAT*, 2015.

[13] J. Crampton, G. Gutin, D. Karapetyan, and R. Watrigant. The bi-objective workflow satisfiability problem and workflow resiliency. *JCS*, 25(1):83–115, 2017.

[14] J. Crampton, G. Gutin, and R. Watrigant. On the satisfiability of workflows with release points. In *Proc. of SACMAT*, 2017.

[15] J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity of the workflow satisfiability problem. In *Proc. of CCS*, 2012.

[16] J. Desel and W. Reisig. Place or transition petri nets. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets.* Springer, 1996.

[17] The Sage Developers. *SageMath, the Sage Mathematics Software System*, 2017. `http://www.sagemath.org`.

[18] 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.

[19] D.R. dos Santos and S. Ranise. On run-time enforcement of authorization constraints in security-sensitive business processes. In *Proc. of SEFM*, 2017.

[20] D.R. dos Santos and S. Ranise. A survey on workflow satisfiability, resiliency, and related problems. *CoRR*, abs/1706.07205, 2017.

[21] D.R. dos Santos, S. Ranise, L. Compagna, and S. E. Ponta. Assisting the Deployment of Security-Sensitive Workflows by Finding Execution Scenarios. In *Proc. of DBSec*, 2015.

[22] J. Dubrovin, T.A. Junttila, and K. Heljanko. Exploiting step semantics for efficient bounded model checking of asynchronous systems. *Sci. Comput. Program.*, 77(10-11):1095–1121, 2012.

[23] H.B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York-London, 1972.

[24] V.K. Garg. Maximal antichain lattice algorithms for distributed computations. In *Proc. of ICDCN*, 2013.

[25] M. Gario and A. Micheli. pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop*, 2015.

[26] J. Holderer, R. Accorsi, and G. Müller. When four-eyes become too much: a survey on the interplay of authorization constraints and workflow resilience. In *Proc. of SAC*, 2015.

[27] D. Karapetyan, A. J. Parkes, G. Gutin, and A. Gagarin. Pattern-based approach to the workflow satisfiability problem with user-independent constraints. *CoRR*, abs/1604.05636, 2016.

[28] M. Leitner and S. Rinderle-Ma. A systematic review on security in process-aware information systems–constitution, challenges, and future directions. *Inf. and Soft. Tech.*, 56(3):273–293, 2014.

[29] J.C. Mace, C. Morisset, and A. Moorsel. Quantitative workflow resiliency. In *Proc. of ESORICS*, 2014.

[30] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[31] A. Roy, S. Sural, A.K. Majumdar, J. Vaidya, and V. Atluri. Minimizing organizational user requirement while meeting security constraints. *ACM Trans. Manage. Inf. Syst.*, 6(3):12:1–12:25, 2015.

[32] R. Sebastiani and P. Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Proc. of CAV*, 2015.

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

[34] Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *TISSEC*, 13, 2010.

[35] M. Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer, Secaucus, NJ, USA, 2007.