Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Authorized workflow schemas

Deciding realizability through LTL(F) model checking

Jason Crampton[1], **Michael Huth**[2], **Jim Huan-Pu Kuo**[2]

[1] Information Security Group, Royal Holloway, University of London
[2] Department of Computing, Imperial College London

Received: date / Revised version: date

Abstract. Many business processes are modeled as workflows, which often need to comply with business rules, legal requirements, and authorization policies. *Workflow satisfiability* is the problem of determining whether there exists a workflow instance that realizes the workflow specification while simultaneously complying with such constraints. This problem has already been studied by the computer security community, with the development of algorithms and the study of their worst-case complexity.

These solutions are often tailored to a particular workflow model and are, therefore, of little or no use in analyzing different models; their worst-case complexities are likely to be an unreliable judge of their feasibility; and they lack support for other forms of analysis such as the determination of the smallest number of users required to satisfy a workflow specification.

We propose model checking of an NP-complete fragment LTL(F) of propositional linear-time temporal logic as an alternative solution. We report encodings in LTL(F) that can not only decide satisfiability but that also compute a whole set of satisfiability witnesses simultaneously, compute minimal user bases and a safe bound on the resiliency of satisfiability under the removal of users. These theoretical contributions are validated through detailed experiments whose results attest to the viability of our proposed approach.

Key words: computer security – model checking – business processes – linear-time temporal logic – abstraction

1 Introduction

Authorized workflows are sets of tasks that are orchestrated or choreographed in some way and where each task has a designated set of users authorized to execute that task.

Send offprint requests to: Jason Crampton, Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK.

We here study abstract representations of authorized workflows that are constrained in various ways: by specifying the order of execution for some tasks, by restricting task execution to certain user sets, and by constraining the binding of users to some pairs of tasks.

The central analysis problem we consider is that of *workflow satisfiability*, which determines whether there is a mapping of tasks to users that meets all articulated constraints. Satisfiability is also referred to as *consistency* or *realizability* and we will use these three terms interchangeably.

The literature in computer security already offers algorithms and complexity analyses for this problem in various of its instances, we mention here [16,5,2,20,21,6]. The computational complexity of this workflow satisfiability problem is typically NP-hard [20]. And algorithms proposed in the literature are mostly investigated in order to show their correctness. The aims of this paper are rather different, though.

LTL(F) is a well known NP-complete fragment of propositional linear-time temporal logic (LTL) [15]. LTL may be used to reason about propositions whose truth value changes over time. Its fragment LTL(F) contains logical conjunction and negation, and makes use of a sole temporal operator F – where Fϕ holds if ϕ holds at some point in the future. We study reductions of workflow satisfiability to LTL(F) model checking. In doing so, we want to understand how such reductions perform in practice, by focusing on the execution and evaluation of experiments on a set of synthetic workflows.

We also want to see whether LTL(F) model checking may offer opportunities for implementing novel analyses that support important validation activities for authorized workflows, including, but not limited to, the following examples:

- to compute a representation of an entire *set* of realizability witnesses as opposed to a sole witness
- to compute the smallest number of users required in order to make a workflow realizable (minimal user base), and
- to learn how many users can be removed from an authorized workflow without jeopardizing its realizability.

The latter measure is known as *resiliency* [20] in the computer security literature. We will show that LTL(F) model checking can be used to compute compact representations of sets of realizability witnesses, safe bounds for resiliency, and the size of a minimal user base.

1.1 Workflow schemas

Let us describe workflows in more detail now. Informally, a workflow is a set of logically related tasks that are performed in some sequence in order to achieve some business objective. But the order of execution may only be partially defined and so tasks may be executed in parallel or in either order.

A *workflow schema* is an abstract specification of the set of workflow tasks, indicating the order in which the tasks in every workflow instance of that schema should be performed.

For each task in the workflow schema, we identify a set of authorized users. (A "user" may well be, or be represented by, a software agent within an IT system, often known as a subject in the access-control literature.) Each workflow task in a workflow instance is assigned to an authorized user, who is responsible for *performing* or *executing* the task.

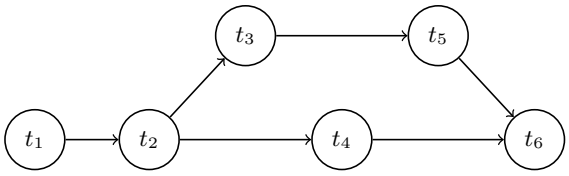
In addition, a workflow schema may have constraints on the users that may be assigned to particular pairs of tasks in any workflow instance. Such constraints may be imposed for a variety of reasons: audit requirements, access control, business logic, etc. Two examples of such constraints are *separation of duty* (also known as the *two-man rule*) and *binding of duty* (where a user is required to perform one task if she has performed some earlier task in the workflow instance).

An illustrative example of a constrained workflow for purchase order processing is shown in Figure 1. The purchase order is created and approved (and then dispatched to the supplier). The supplier will present an invoice, which is processed by the create-payment task. When the supplier delivers the ordered goods, a goods received note must be signed and countersigned; only then may the payment be approved. As we noted above, a workflow specification need not be linear: the processing of the goods received note and of the invoice can occur in parallel, for example.

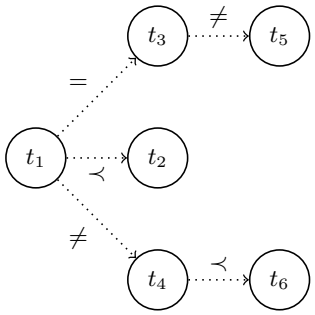
In addition to constraining the order in which tasks are to be performed, some business rules are specified to prevent fraudulent use of the purchase order processing system. These rules take the form of constraints on users that can perform pairs of tasks in the workflow: for example, the same user may not sign and countersign the goods received note.

It is readily apparent that the imposition of constraints and the limited number of authorized users for particular tasks may result in a workflow schema being *unsatisfiable*, meaning that there would be no way of selecting an authorized user for each task in such a way that all constraints are satisfied.

A workflow can therefore only be realized if it is satisfiable. It is therefore important to be able to decide workflow satisfiability, both *statically* (realizability under full control over user-task assignment) and *dynamically* (realizability when some tasks have already been executed by authorized users).



(a) Task ordering



(b) Constraints

t_1	create-purchase-order
t_2	approve-purchase-order
t_3	sign-goods-received-note
t_4	create-payment
t_5	countersign-goods-received-note
t_6	approve-payment
$\neq$	users performing tasks must be different
$=$	users performing tasks must be the same
$\prec$	user performing second task must be senior to first user

(c) Figure legend

Fig. 1. A simple constrained workflow for purchase order processing

The problems of user-task assignment and workflow satisfiability have been studied by researchers in the security community using a variety of *bespoke* methods and algorithms [2,5,20]. The bespoke nature of these methods makes it hard to assess whether they adapt easily to other, more expressive workflow models or to more sophisticated analyses – such as those that cross workflow instances.

The extant work in the literature does, however, tell us about the computational complexity of these problems for specific types of models. Wang and Li [20], for example, show that workflow satisfiability is NP-hard for many types of models as soon as constraints that either bind or separate users are present. They prove this through a simple reduction from graph-colorability to workflow satisfiability.

These complexity results inform the search for more adaptable and general solutions, since we may be well advised to only choose solutions with matching complexities. But these results also suggest that it might be reasonable to ask how reliable a guide worst-case complexities are when it comes to applying such techniques as decision procedures to workflows in practice.

We therefore explore here whether tried and tested techniques and tools from the formal-methods community can provide a more robust, more expressive, more adaptable, and

more informative foundation for authorized workflow systems. Concretely, we consider model checking for $\mathsf{LTL}(\mathsf{F})$, an NP-complete fragment of the propositional linear-time temporal logic [4].

One reason why we are interested in $\mathsf{LTL}(\mathsf{F})$ is that we want to investigate to what extent a bounded model checker can bring additional capabilities to the realizability problem. In particular, we want to research how bounded model checking can compute not just one realizability witness but a compact representation of many such witnesses. The latter will provide more flexibility in scheduling authorized workflows.

We experimentally evaluate and stress test the suitability of $\mathsf{LTL}(\mathsf{F})$ model checking for deciding workflow realizability through the study of three reductions of workflow satisfiability to the problem of satisfiability of $\mathsf{LTL}(\mathsf{F})$ formulas.

This then gives us a reduction to $\mathsf{LTL}(\mathsf{F})$ model checking via a standard reduction of LTL satisfiability to LTL model checking: a formula of LTL is satisfiable if, and only if, its negation is false in a certain fully connected model for LTL.

It is that fully connected model on which our experiments are then performed. The experimental results we report here suggest that these reductions have good potential for workflow realizability checking in practice.

Outline of paper. In Section 2, we provide technical background from temporal-logic model checking and present a particular model of workflow authorization schemas. In Section 3, we show how the realizability problem for this model can be encoded as a satisfiability problem in an NP-complete fragment of propositional linear-time temporal logic. Two more abstract encodings in that temporal logic are then developed in Section 4. Our experimental insights for all three encodings are reported and compared against each other in Section 5. In Section 6, we look at workflow satisfiability in practice to assess the nature and scope of needs for solutions. In particular, we demonstrate that propositional linear-time temporal logic can indeed express richer types of analyses. Related work is discussed in Section 7. Finally, in Section 8 we conclude the paper, and point out future work.

2 Technical background

We recall the definition of a representative authorization model for workflow systems, and the definition of workflow satisfiability for that model [5]. Then we define the propositional linear-time temporal logic $\mathsf{LTL}(\mathsf{F})$ [15] that we will use to reduce the workflow satisfiability problem for that model to one of the satisfiability of a formula in $\mathsf{LTL}(\mathsf{F})$. Throughout, we write $<$ for the strict version of a partial order $\leq$.

2.1 Constrained workflow authorization schemas

We formally define workflow authorization schemas such as the one shown graphically in Figure 1.

Definition 1. A *constrained workflow authorization schema* is a tuple (T, U, A, C) where

- $(T, \leq)$ is a finite partial order of *tasks*, where $t < t'$ means that task t has to be completed before task t';
- U is a finite set of users;
- $A \subseteq T \times U$ is the *authorization relation*;
- C is a finite set of *entailment constraints*, tuples of form $(D, t \to t', \rho)$ where $D \subseteq U$, $t, t' \in T$ and $\rho \subseteq U \times U$.

The partial order $(T, \leq)$ expresses constraints on the order in which tasks may be executed: any topological sort of T consistent with $\leq$ defines a legal execution sequence, which we call a *linearization* (of T). Relation $t < t'$ therefore specifies that task t must occur prior to task t' in all such linearizations. Additionally, the partial order expresses which tasks (those incomparable with respect to $\leq$) may be executed independently (that is, concurrently or in either order).

Both A and C are defined with reference to the set of users U. Henceforth, we will write (T, A, C) to denote a workflow; that is, U is assumed to be implicit in the definition of A and C.

Relation A specifies the inherent authorization policy for the workflow. The informal interpretation of A is that u is authorized to perform task t if and only if (t, u) is in A. Note that authorization relation A abstracts away the particularities of the underlying access-control model, simplifying our exposition. Typically, this model will be role-based [13], where users are authorized to perform certain roles and roles are authorized to execute certain tasks. In other words, it is very easy to deduce A from the access-control policy.

A constraint $(D, t \to t', \rho)$ is used to encode restrictions on the users that can perform certain pairs of tasks, in order to enforce business rules or regulatory requirements. The constraint $(D, t \to t', \rho)$ states that if there are users u and u' in U assigned to tasks t and t', respectively, and if u happens to be from target set D, then (u, u') must belong to relation ρ. Relation ρ and set D can vary with each element of C.

We emphasize that such a constraint $(D, t \to t', \rho)$ between t and t' is here not necessarily one of temporal order, but one of logical implication: if a user from a designated set D is assigned to task t, then constraints apply on the assignment of users to task t'. This is in contrast to the manner in which such entailment constraints were defined in [5], where $t' \not\leq t$ is an additional requirement. We dropped this requirement here, as it may be useful to have entailments between tasks that are not merely temporal in nature or that may refer back to past tasks.

2.2 Authorized plans

We now formalize and discuss what constitutes a solution to the realizability of authorized workflow schemas.

Definition 2. Let (T, A, C) be a constrained workflow authorization schema.

1. An *authorized plan* for (T, A, C) is a pair (L, α), where
   - $\alpha \colon T \to U$ is a total function that assigns tasks to users such that $(t, \alpha(t))$ is in A for all t in T;

- L is a linearization of T; and
- for all $(D, t \to t', \rho)$ in C, we have that if $\alpha(t)$ is in D then $(\alpha(t), \alpha(t'))$ is in ρ.

2. We say that (T, A, C) is *satisfiable* if and only if it has an authorized plan.

Authorized plans are implementable realizations of a constrained workflow authorization schema: they assign all tasks only to authorized users such that all entailment constraints are met, and they provide a linearization of all tasks as a possible task scheduler.

Let us return to our purchase order example and suppose that the underlying access-control mechanism is role-based [13]. There are two roles, FinAdm and FinClrk (corresponding to the jobs of financial administrator and financial clerk, respectively), with FinClrk $\prec$ FinAdm in the role hierarchy (meaning that administrators are more senior than clerks).

Role FinAdm is authorized to perform tasks t_2 and t_6 – corresponding to purchase order approval and payment approval, respectively. The FinClrk role is authorized to perform the remaining tasks (as is FinAdm by virtue of seniority).

Suppose, finally, that there are only two users $Alice$ and Bob, where $Alice$ is assigned to the FinAdm role and Bob is assigned to the FinClrk role (with the consequence that $Alice$ is assumed to be more senior than Bob).

Now, if $Alice$ executes t_1 (as she is authorized to do by role inheritance), the workflow instance becomes unsatisfiable, because the constraint on tasks t_1 and t_2 requires that t_2 be executed by a user more senior than $Alice$. Hence, an attempt by $Alice$ to perform this task should be prevented (even though she is authorized according to relation A).

However, if Bob executes t_1, then the constraint on tasks t_1 and t_4 requires that $Alice$ performs t_4, with the consequence that there is no authorized user to perform t_6 (since it must be performed by a user more senior than the user that performed t_4).

In other words, the workflow schema is unsatisfiable, given the current user population and authorization policy. If we were to add a third user $Carol$ and assign her to role FinClrk, the schema becomes satisfiable: one such authorized plan, represented as a (temporal) list of task-user pairs, is

$$[(t_1, Bob), (t_2, Alice), (t_4, Carol),$$
$$(t_3, Bob), (t_5, Carol), (t_6, Alice)].$$

Note that if we added the constraint that t_2 (approve-purchase-order) and t_6 (approve-payment) are performed by different users – which is not unreasonable – then the workflow schema becomes unsatisfiable again, because there is a single user assigned to FinAdm.

2.3 Constraint solving and temporal scheduling

We now study whether and when we can independently solve the constraints of the form $(D, t \to t', \rho)$ and solve the temporal execution of all tasks. Such a decomposition is attractive as it helps with taming the algorithmic complexity of realizability in practice, as we will see in Section 5.

Crampton [5] introduced the notion of a *well-formed* workflow schema, and showed that if (L, α) is an authorized plan for the well-formed schema $\mathcal{AS} = (T, A, C)$, then (L', α), where L' is any other linearization of T, is also an authorized plan of $\mathcal{AS}$.

Well-formedness is a restriction on the constraints that may be specified on tasks that are not comparable to each other (with respect to $\leq$); such tasks may appear in either order in a linearization of T. Informally, well-formedness requires that constraints on such tasks are mutually consistent in some appropriate sense.

In Figure 1, every constraint is defined between a pair of tasks that belong to the partial ordering. Well-formedness would require that if we introduced a constraint on t_3 and t_4 with constraint relation ρ, then there should also be a constraint on t_4 and t_3 with constraint relation $\tilde{\rho}$, where $\tilde{\rho} = \{(y, x) : (x, y) \in \rho\}$ denotes the *converse* of ρ.

The most natural type of relation to use in this situation would be one that is symmetric (whence ρ equals $\tilde{\rho}$); both $\neq$ and $=$ are symmetric, for example. But seniority $\prec$ would also comply since if u is senior to u', then u' is junior (the converse of senior) to u.

We would expect that workflow schemata will be well-formed in many application domains. For such schemata, the results in [5] mean that we can fix a particular linearization of L before considering the problem of identifying a suitable α. And this problem decomposition may be exploited in solutions to the satisfiability problem.

2.4 Linear-time temporal logic LTL(F)

Below we show that the satisfiability problem for the workflow model of Definition 1 can be expressed as a satisfiability problem in a fragment of propositional linear-time temporal logic that has NP-complete satisfiability checks [15].

We therefore provide a brief review of this logic. Given a finite set AP of atomic propositions, the propositional temporal logic LTL(F) is generated by the following grammar:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{F}\,\phi$$

where p is from AP and F is the temporal connective "Future" such that $\mathsf{F}\,p$ states that p will be true at some point in the future.

A *model* of a formula ϕ is an infinite sequence of states $\pi = s_0 s_1 \ldots$, where each s_i is a subset of AP. We write $\pi \models \phi$ if π is a model for ϕ. We say that a formula ϕ is *satisfiable* if and only if it has a model. We write π^i to denote the infinite suffix $s_i s_{i+1} \ldots$ of π. The formal semantics of formulas is then given in Figure 2.

We use the usual abbreviations for disjunction ($\vee$), implication ($\rightarrow$), logical equivalence ($\leftrightarrow$) and the "Global" temporal connective $\mathsf{G}\,\phi$, which stands for $\neg\mathsf{F}\neg\phi$ (the informal interpretation being "always ϕ"). Also, we write S^* for the set of finite words and S^ω for the set of infinite words over state set S. For finite words v and w from S^* we write vw^ω for the word $vww\ldots$ (that has prefix v and then repeats w infinitely often) and w^ω for the word $ww\ldots$.

$$\begin{array}{l}\pi \models p \quad \text{iff} \quad p \in s_0\\ \pi \models \neg\phi \quad \text{iff} \quad \text{not } \pi \models \phi\\ \pi \models \phi_1 \wedge \phi_2 \quad \text{iff} \quad (\pi \models \phi_1 \text{ and } \pi \models \phi_2)\\ \pi \models \mathsf{F}\,\phi \quad \text{iff} \quad \text{there is } i \geq 0 \colon \pi^i \models \phi\end{array}$$

Fig. 2. Formal semantics of temporal logic $\mathsf{LTL}(\mathsf{F})$ over models π

2.5 LTL *satisfiability checking through* LTL *model checking*

We describe a standard reduction of LTL satisfiability checking to model checking. A *transition system* M for LTL is a tuple (S, R, I) where S is a set of LTL states, $R \subseteq S \times S$ is a binary relation over S, and $I \subseteq S$ is a set of initial states. Each transition system M determines its set of paths Π_M. Formally,

$$\Pi_M \stackrel{\text{def}}{=} \{\pi = s_0 s_1 \cdots \in S^\omega \mid s_0 \in I, \forall i \geq 0 \colon (s_i, s_{i+1}) \in R\}$$

Informally, the paths of transition system M are infinite sequences of states in S, connected via R, such that the first state is in the set I of initial states of M.

A transition system M is then said to satisfy an LTL formula ϕ, written $M \models \phi$, iff all paths of M satisfy ϕ; i.e. if all paths of M are models of ϕ. Formally,

$$M \models \phi \stackrel{\text{def}}{=} \forall \pi \in \Pi_M \colon \pi \models \phi$$

Let ϕ be a formula where $A \subseteq \mathsf{AP}$ is the set of atomic propositions occurring in ϕ. To simplify the presentation, we assume that A equals AP. Then we define a fully connected transition system M_{AP} that contains all possible paths for ϕ as follows:

$$S_{\mathsf{AP}} \stackrel{\text{def}}{=} 2^{\mathsf{AP}}, \;\; I_{\mathsf{AP}} \stackrel{\text{def}}{=} S_{\mathsf{AP}}, \;\; R_{\mathsf{AP}} \stackrel{\text{def}}{=} S_{\mathsf{AP}} \times S_{\mathsf{AP}} \qquad (1)$$

Informally, all subsets of AP are states in M_{AP}, all states are initial in M_{AP}, and all states are connected to all states in M_{AP}. Therefore, $\Pi_{M_{\mathsf{AP}}}$ is indeed the set of *all possible paths* for atomic proposition set AP.

Thus, we infer that the LTL formula ϕ is satisfiable iff its negation is not satisfied by transition system M_{AP}:

$$\phi \text{ is satisfiable if, and only if, } M_{\mathsf{AP}} \not\models \neg\phi \qquad (2)$$

and this is what the NuSMV model checker will decide in our experiments.

3 Expressing workflow satisfiability in LTL(F)

Next we show how to construct an $\mathsf{LTL}(\mathsf{F})$ formula $\phi_{\mathcal{AS}}$ from a constrained workflow authorization schema $\mathcal{AS} = (T, A, C)$ – be it well-formed or not – such that $\phi_{\mathcal{AS}}$ is satisfiable as a formula of $\mathsf{LTL}(\mathsf{F})$ if and only if $\mathcal{AS}$ is satisfiable as a workflow in the sense given in Definition 2.

This correspondence is constructive in that the respective models of LTL formulas and workflows intertranslate directly and in a meaningful way.

$$\phi_{\mathsf{F}T} \stackrel{\text{def}}{=} \bigwedge_{t \in T} \mathsf{F}\,t \qquad (3)$$

$$\phi_{\mathsf{G}T} \stackrel{\text{def}}{=} \mathsf{G}\Big(\bigvee_{t \in T} t\Big) \qquad (4)$$

$$\phi_{seT} \stackrel{\text{def}}{=} \bigwedge_{t \in T} \mathsf{G}\Big(t \rightarrow \bigwedge_{t' \in T \setminus \{t\}} \neg t'\Big) \qquad (5)$$

$$\phi_{\mathsf{G}U} \stackrel{\text{def}}{=} \mathsf{G}\Big(\bigvee_{u \in U} u\Big) \qquad (6)$$

$$\phi_{seU} \stackrel{\text{def}}{=} \bigwedge_{u \in U} \mathsf{G}\Big(u \rightarrow \bigwedge_{u' \in U \setminus \{u\}} \neg u'\Big) \qquad (7)$$

$$\phi_{\leq} \stackrel{\text{def}}{=} \bigwedge_{t \in T} \mathsf{G}\Big(t \rightarrow \mathsf{G}\big(\bigwedge_{t' \sqsubset t} \neg t'\big)\Big) \qquad (8)$$

$$\phi_A \stackrel{\text{def}}{=} \bigwedge_{t \in T} \mathsf{G}\Big(t \rightarrow \bigvee_{(t,u) \in A} u\Big) \qquad (9)$$

$$\phi_C \stackrel{\text{def}}{=} \bigwedge_{(D, t \rightarrow t', \rho) \in C} \phi_{(D, t \rightarrow t', \rho)} \qquad (10)$$

$$\phi_{(D, t \rightarrow t', \rho)} \stackrel{\text{def}}{=} \bigwedge_{u \in D} \big(\mathsf{F}\,(t \wedge u)\big) \rightarrow \mathsf{G}\Big(t' \rightarrow \bigvee_{(u,u') \in \rho} u'\Big) \qquad (11)$$

Fig. 3. Defining the conjuncts of $\phi_{\mathcal{AS}}$ in $\mathsf{LTL}(\mathsf{F})$ for constrained workflow authorization schema $\mathcal{AS} = (T, A, C)$.

3.1 *Encoding in* LTL(F)

We first define the set of atomic propositions AP to be

$$\mathsf{AP} \stackrel{\text{def}}{=} U \cup T$$

where we assume that the sets U and T are mutually disjoint. Below, we write $t \sqsubset t'$ iff $t < t'$ and if there is no third task t'' such that $t < t''$ and $t'' < t'$. Formally

$$t \sqsubset t' \text{ iff } (t < t', \forall t'' \in T \colon (t \leq t'' \leq t') \text{ implies } t'' \in \{t, t'\})$$

Formula $\phi_{\mathcal{AS}}$ is a conjunction of formulas

$$\phi_{\mathcal{AS}} \stackrel{\text{def}}{=} \phi_{\mathsf{F}T} \wedge \phi_{\mathsf{G}T} \wedge \phi_{seT} \wedge \phi_{\mathsf{G}U} \wedge \phi_{seU} \wedge \phi_{\leq} \wedge \phi_A \wedge \phi_C,$$

where each of the conjuncts is defined in Figure 3. The conjuncts of $\phi_{\mathcal{AS}}$ together guarantee that infinite sequences π with $\pi \models \phi_{\mathcal{AS}}$ correspond to authorized plans of $\mathcal{AS}$; and conversely, that authorized plans of $\mathcal{AS}$ give rise to infinite sequences π with $\pi \models \phi_{\mathcal{AS}}$. We will sketch the proof of these claims below.

The intuition for this encoding of $\mathcal{AS}$ is that

- $t \in s_i$ means that task t is scheduled in state s_i
- $u \in s_i$ means that user u is assigned to any task scheduled at state s_i.

By abuse of language, we will speak below of states scheduling tasks and assigning users in accordance with the above intuition.

3.2 Discussion of encoding

We now discuss the intended meaning of each formula specified in Figure 3. Let π be any infinite sequence of states such that $\pi \models \phi_{\mathcal{AS}}$, keeping in mind that no such π exists if $\phi_{\mathcal{AS}}$ is unsatisfiable.

(1) Formula $\phi_{\mathsf{F}T}$ states that all tasks are eventually scheduled in π.
(2) Formula $\phi_{\mathsf{G}T}$ states that all states of π assign some task from T.
(3) Formula ϕ_{seT} is a kind of single-event condition. It specifies that whenever a state in π schedules any task t from T, then that state cannot schedule any other task.
(4) Formula $\phi_{\mathsf{G}U}$ states that all states of π assign some user.
(5) Formula ϕ_{seU} also captures a kind of single-event condition. It specifies that, for all users u and at all states of π, if a state assigns user u, then that state does not assign any other users.
(6) Formula $\phi_{\leq}$ specifies, for each task t from T, that whenever t is scheduled at a state in π, then all future states can only schedule a task t' such that $t' \not\sqsubset t$.
(7) Formula ϕ_A encodes the authorization schema A. It specifies that for all tasks $t \in T$ and states s_i of π, if s_i schedules t, then s_i assigns some user u with (t, u) in A.
(8) Formula ϕ_C encodes the entailment constraints in C. It is a conjunction of formulas $\phi_{(D,t\to t',\rho)}$, one for each element $(D, t \to t', \rho)$ of C.
(9) Formula $\phi_{(D,t\to t',\rho)}$ encodes $(D, t \to t', \rho)$ as a conjunction over each user u from D: it specifies that if there is a state of π that schedules task t and assigns user u, then all states of π that schedule task t' are such that they assign at least one user u' such that (u, u') is in ρ.

In the formula $\phi_{\leq}$ we rule out the scheduling of elements t' with $t' \leq t$ once t has occurred by ruling out such scheduling for all immediate predecessors of t. This is sound since the partial order $\leq$ is finite and so well-founded. We use this indirect encoding as it involves fewer conjuncts in $\phi_{\mathcal{AS}}$.

Most of the conjuncts of $\phi_{\mathcal{AS}}$ over-approximate the intended behavior of a constrained workflow authorization schema $\mathcal{AS}$: for example, $\phi_{(D,t\to t',\rho)}$ ensures that there is some assignment of users that satisfies this entailment constraint, but it does not prevent the assignment of users that violate these constraints; rather, it is the interaction of all conjuncts in $\phi_{\mathcal{AS}}$ that gives these approximations the desired precision.

We observe that the encoding $\phi_{\mathcal{AS}}$ is such that there are no modalities under the scope of other modalities, with the exception of conjunct $\phi_{\leq}$ where one G modality is under the scope of another G. So $\phi_{\mathcal{AS}}$ has low modal depth.

We also note that the size of the formula $\phi_{\mathcal{AS}}$ is quadratic in the size of $\mathcal{AS}$ if we think of the latter as the sum of the cardinalities for sets $\leq$, A, C, and $U \times U$ – which seems a reasonable measure. So the conversion from $\mathcal{AS}$ to $\phi_{\mathcal{AS}}$ does not represent an exponential blow-up.

3.3 Correctness of encoding

We now explain why this encoding correctly decides the realizability of authorized workflow schemas. For a constrained workflow authorization schema $\mathcal{AS} = (T, A, C)$ and $\phi_{\mathcal{AS}}$ its encoding in $\mathsf{LTL}(\mathsf{F})$, we have that

$\phi_{\mathcal{AS}}$ is satisfiable if and only if workflow $\mathcal{AS}$ is satisfiable.

We sketch the proof of that claim here and do this in part since one direction of that proof also explains how a model of $\phi_{\mathcal{AS}}$ is translated into a witness for workflow satisfiability. Understanding this technique will be beneficial for appreciating other reductions discussed in this paper.

It is easy to show that our encoding is complete (that is, that $\phi_{\mathcal{AS}}$ is satisfiable if $\mathcal{AS}$ is): given an authorized plan (L, α) for $\mathcal{AS}$ with $L = [t_1, \ldots, t_k]$, we construct a model $\pi = v^\omega$ of $\psi_{\mathcal{AS}}$ where

$$v \stackrel{\text{def}}{=} \{t_1, \alpha(t_1)\}\{t_2, \alpha(t_2)\} \ldots \{t_k, \alpha(t_k)\}.$$

The crucial result is the soundness of our encoding. Let $\phi_{\mathcal{AS}}$ be satisfiable. Then there is some model π with $\pi \models \phi_{\mathcal{AS}}$ and so π satisfies all conjuncts that make up $\phi_{\mathcal{AS}}$.

For $\phi_{\mathsf{F}T}$ this means that for each task t in T there is a first state s_{i_t} of π at which t is true. From ϕ_{seU} and $\phi_{\mathsf{G}U}$ we know that there is a unique user u in U that holds at state s_{i_t}. We define $\alpha(t)$ to be that u. Using ϕ_A together with ϕ_{seU} we then know that the pair $(t, \alpha(t))$ is in A.

Now let $(D, t \to t', \rho)$ be a constraint in set C. From $\phi_{(D,t\to t',\rho)}$ we then know that if $\alpha(t)$ is in D, then the consequent of the implication for u being $\alpha(t)$ is also true. Using ϕ_{seU} and $\phi_{\mathsf{G}U}$, we then infer that $(\alpha(t), \alpha(t'))$ has to be in ρ as desired.

Finally, we can construct a linearization L for $(T, \leq)$ from the order in which tasks t occur in the sequence that is listing the states from set $\{s_{i_t} \mid t \in T\}$ in the same order as in model π. That this is indeed a linearization follows from $\pi \models \phi_{\leq}$ using the well-foundedness of $\leq$.

4 Abstraction of temporal order

Our experimental results for model checking $\phi_{\mathcal{AS}}$, reported in Section 5, suggest that this approach may already become infeasible for workflows with 30 tasks and high constraint densities.[1] This is clearly unsatisfactory.

We therefore now investigate whether we can use a simpler encoding in $\mathsf{LTL}(\mathsf{F})$, referred to as $\psi_{\mathcal{AS}}$ subsequently, that allows us to

– increase the number of tasks that $\mathsf{LTL}(\mathsf{F})$ model checking can handle,
– increase the number of authorized plans that this process may compute,

[1] Constraint density is the term we will use for the ratio of the number of constraints in the workflow to the total number of possible constraints.

$$\psi_{\mathsf{F}T} \stackrel{\text{def}}{=} \phi_{\mathsf{F}T} = \bigwedge_{t \in T} \mathsf{F}\, t$$

$$\psi_{\mathsf{G}U} \stackrel{\text{def}}{=} \phi_{\mathsf{G}U} = \mathsf{G}(\bigvee_{u \in U} u)$$

$$\psi_A \stackrel{\text{def}}{=} \bigwedge_{t \in T} \mathsf{G}\left(t \rightarrow \neg(\bigvee_{(t,u) \notin A} u)\right)$$

$$\psi_C \stackrel{\text{def}}{=} \bigwedge_{(D,t \rightarrow t',\rho) \in C} \psi_{(D,t \rightarrow t',\rho)}$$

$$\psi_{(D,t \rightarrow t',\rho)} \stackrel{\text{def}}{=} \bigwedge_{u \in D} \left(\mathsf{F}\,(t \wedge u)\right) \rightarrow \mathsf{G}\left(t' \rightarrow \neg(\bigvee_{(u,u') \notin \rho} u')\right)$$

$$\psi_{\mathcal{AS}} \stackrel{\text{def}}{=} \phi_{\mathsf{F}T} \wedge \phi_{\mathsf{G}U} \wedge \psi_A \wedge \psi_C \qquad (12)$$

Fig. 4. Defining $\psi_{\mathcal{AS}}$ in $\mathsf{LTL}(\mathsf{F})$ so that several tasks and users may hold in a state and where temporal order of task execution is ignored.

- compute a minimal user base for realizing the workflow, and
- compute a safe bound for the resiliency of the workflow.

All of these capabilities are quite useful. The first one means that more workflows of realistic size can be subjected to formal analysis. The second one gives a system more flexibility in allocating users to tasks since more solutions are available. One application of this flexibility is that we can gain better resiliency of workflow satisfiability, as we can guarantee that the workflow remains satisfiable even after a certain number of users are no longer available to the workflow [20]. The third capability above provides an informative lower bound on the number of users needed for realizing a workflow. And the fourth capability is useful as it gives us a reliable measure of the actual resiliency of the workflow for the current user bases.

4.1 The ψ encoding

The key observation is that if (L, α) is an authorized plan, then (L', α) is also an authorized plan for *any* linearization L' of T. In other words, the determination of the temporal order of task executions can be computed independently of the satisfaction of the other constraints. It is easy to see that this claim holds for the constraints related to the authorization relation A. For constraints of form $(D, t \rightarrow t', \rho)$, note that satisfaction of such a constraint is based solely on the relationship between the users that execute the tasks, not the order in which they are executed.

We can therefore derive a simpler encoding $\psi_{\mathcal{AS}}$, depicted in Figure 4. It implicitly assumes that we will run the LTL *bounded* model checker, and so the model checker will explore whether solutions exist of length 0, 1, 2, etc., until and if a solution of minimal length is found.

Therefore, we have no need for formula $\phi_{\mathsf{G}T}$ but we keep formulas $\phi_{\mathsf{F}T}$ and $\phi_{\mathsf{G}U}$ so that all tasks are realized, and all states identify at least one user. The latter is stipulated in order to avoid the generation of states that cannot contribute to the construction of α.

We also omit formulas ϕ_{seT} and ϕ_{seU} as we actually want more than one task and user to be true in a state, whenever possible. In fact, our use of $\mathsf{LTL}(\mathsf{F})$ and a bounded model checker lets us group multiple tasks and users in a single state, thereby enabling us to analyze additional properties of a workflow specification (beyond its satisfiability).

Formula $\phi_{\leq}$ is also omitted as we do not model the temporal order of tasks here. The formulas ϕ_A and ϕ_C are modified into formulas ψ_A and ψ_C, respectively. Formula ψ_A ensures that whenever a task t is true in a state, then no user that is not authorized for task t is true at that state. (That is, every user in state s is authorized for every task in that state.)

Formula ψ_C is again a conjunction over all constraints. The formulas $\psi_{(D,t \rightarrow t',\rho)}$ are similar in structure to the formulas $\phi_{(D,t \rightarrow t',\rho)}$ but they specify that whenever there is a state in which t is true and some user u from D, then it must be the case that at all states at which t' is true there is no user u' true such that (u, u') is not in ρ. We note that $\psi_{\mathcal{AS}}$, unlike $\phi_{\mathcal{AS}}$, does not nest any modal operators.

Let us consider this encoding on our example workflow with three users $Alice$, Bob, and $Carol$. A possible model π for $\psi_{\mathcal{AS}}$ may be $s_0 s_1 s_2 s_2 \ldots$ where

- t_2, t_6, and $Alice$ are true at s_0
- t_1, t_5, and Bob are true at s_1 and
- t_4, t_3, and $Carol$ are true at s_2.

In the number of relevant states, this solution is half as long as any solution that can be obtained from $\phi_{\mathcal{AS}}$. Moreover, we can directly read off the task/user mapping α from these three states.

Now suppose that we would also add a user $Fred$ who also has role FinClerk. Then we could make $Fred$ true at state s_1 above. In particular, each of t_1 and t_5 could be assigned to either Bob or $Fred$ without violating any entailment or other constraints.

4.2 From authorized plans to models, and back

From the above example, it is intuitively clear how one can take an authorized plan (L, α) and construct a model π for $\psi_{\mathcal{AS}}$ from it. One such construction proceeds in the same manner as we constructed a model from such a π for $\phi_{\mathcal{AS}}$.

The more interesting and practically relevant case is when we have a model $\pi = s_0 s_1 \ldots$ of $\psi_{\mathcal{AS}}$. How can we then construct an authorized plan (L, α) from it? As before, we focus on generating the function α from π.

One useful, non-deterministic method is to construct a total function $\Gamma : T \rightarrow 2^U$ that maps tasks to (non-empty) sets of users. The salient property of Γ is that *all* choice functions $\alpha : T \rightarrow U$ such that $\alpha(t)$ is in $\Gamma(t)$ for all t in T can be extended to an authorized plan (L, α) for the workflow $\mathcal{AS}$.

We now describe one way in which function Γ can be constructed. Let s_{i_0} be the first state in π at which some task is true. For each task t true in s_{i_0}, define

$$\Gamma(t) \stackrel{\text{def}}{=} s_{i_0} \cap U$$

as the set of users that are also true at s_{i_0}. By the properties of $\psi_{\mathcal{AS}}$ there has to be at least one such state s_{i_0} and then $\Gamma(t)$ is non-empty.

Again, by the properties of $\psi_{\mathcal{AS}}$, we either have defined Γ already for all tasks, or there must be some least state s_{i_1} in π at which a task t is true for which Γ is not yet defined. As before, we can define $\Gamma(t)$ to be the set of users true at s_{i_1}.

This process continues inductively, until Γ is defined on all tasks. Formulas $\psi_{\mathsf{F}T}$ and $\psi_{\mathsf{G}U}$ ensure that Γ will be defined for all tasks. The remaining conjuncts of $\psi_{\mathcal{AS}}$ ensure that Γ meets all constraints of the workflow:

- for all t in T and for all u in $\Gamma(t)$, we have (t, u) is in A, and
- for all $(D, t \to t', \rho)$ in C and all u in $D \cap \Gamma(t)$ we have that (u, u') is in ρ for all u' in $\Gamma(t')$.

From these two properties it follows that all choice functions α of Γ, which serve to resolve the non-determinism of Γ, meet all non-temporal constraints of the workflow. In fact, the bounded model checking of LTL allows us to say more: namely, that all states in the solution returned by the model checker will contribute to the computation of our solution. To see this, we note that the path $s_{i_0}s_{i_1}\ldots$ is a model of $\psi_{\mathcal{AS}}$ as well. But bounded model checking finds a shortest such model.

We now offer an alternative way of explaining the construction of possible solutions from a model π. This is useful as it highlights that models π capture *cartesian abstractions* of solutions. Such abstractions have successfully been used for the software verification of C programs in the past [1].

For the model $\pi = s_0 s_1 \ldots$ of $\psi_{\mathcal{AS}}$, we may construct for each i a subset of users U_i and a subset T_i of tasks:

$$\begin{aligned} T_0 &= s_0 \cap T \\ T_{i+1} &= s_{i+1} \cap \Big(T \setminus \bigcup_{j=0}^{i} T_j\Big) \\ U_i &= s_i \cap U \end{aligned} \tag{13}$$

Intuitively, the possibly empty set T_i contains those tasks that are true in state s_i and that have been false in all previous states of path π. In particular, each task t in T has a unique i_t such that t is in T_{i_t}. We can then construct task/user mappings α by assigning to t any user u in user set U_{i_t}.

4.3 The δ encoding

The more abstract encoding $\psi_{\mathcal{AS}}$ is certainly a big improvement over the encoding $\phi_{\mathcal{AS}}$; it improves scalability and provides us with other useful measures connected to workflow realizability, as we will show below.

But our experiments showed that models of $\psi_{\mathcal{AS}}$ often compute functions Γ that have relatively few choice functions α. In one synthetic workflow model with 150 tasks, for example, Γ generates only one α since $\Gamma(t)$ comprises a single user for each t.

We therefore suggest a variant of this encoding that may increase the number of choice functions without increasing too much the running time of LTL(F) model checking.

This encoding $\delta_{\mathcal{AS}}$ simply adds another conjunct that forces all users to be true eventually:

$$\delta_{\mathcal{AS}} \stackrel{\text{def}}{=} \psi_{\mathcal{AS}} \wedge \bigwedge_{u \in U} \mathsf{F}\, u \tag{14}$$

The intuition behind this is that we use a bounded model checker that will find the shortest possible prefix whose loop will give a model of the formula. So the model checker will try to pack as many users into states as possible to capture a solution, and will put all those users that were not needed for the solution into another "junk" state, and only into one such junk state.

4.4 Models as partitions of the task set and assigned users

Given a model π of either $\psi_{\mathcal{AS}}$ or $\delta_{\mathcal{AS}}$, let us consider the pairs (T_i, U_i) of non-empty sets T_i and sets U_i defined in (13). It is clear that these sets T_i partition set T of all tasks: these sets are mutually disjoint by definition, non-empty by choice, and their union contains T by construction. Moreover, the sets U_i (with the exception of the set of users associated with the junk state) are mutually disjoint. Assume, by way of contradiction, that there is a user u in $U_i \cap U_j$ where i is different from j. Then α could assign u to all tasks in $T_i \cup T_j$ and this would result in a shorter solution, which was not found and hence does not exist. This shorter solution would replace the two states s_i and s_j with the sole state

$$\{u\} \cup T_i \cup T_j$$

We experimentally verified this property on all models π computed for $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$ on our randomly generated $\mathcal{AS}$.

This property formalizes that our models express cartesian abstractions as task/user mappings, are invariant under both equivalence relations, and do not care about users that are not in some set U_i.

4.5 Minimal user base

We can use a similar argument to the one just employed to see why this encoding and construction of Γ, when run in bounded model-checking mode, can be used to return a solution with a minimal user base. By a minimal user base, we mean a set of users of size k such that the workflow can be realized with just these k users, and that there is no user set of size smaller than k that can realize that workflow.

Formally, we set

$$\mathsf{mub}(\Gamma) \stackrel{\text{def}}{=} |\{T_i \mid T_i \neq \emptyset\}|$$

to be the number of states s_i needed in order to enumerate all tasks in set T. From our definition of Γ it is clear that we can construct an authorized plan that allocates only $\mathsf{mub}(\Gamma)$ users, we just assign all tasks in T_i to some fixed user in U_i.

But we argue that this is also the minimal set of users one could assign for this workflow.

Assume, by way of contradiction, that there exists an authorized plan (L, α) that allocated fewer than $\mathsf{mub}(\Gamma)$ users, say users u_1 up to u_k. Then bounded model checking would be able to find a "counterexample" $(s'_0 s'_1 \dots s'_k)^\omega$ that is shorter than the one the bounded model checker reported (a contradiction): for each i with $1 \leq i \leq k$, we can define state s'_i to be $\{u_i\} \cup \{t \in T \mid \alpha(t) = u_i\}$. Then $(s'_0 s'_1 \dots s'_k)^\omega$ is a model of whatever encoding was used for computing Γ: formula $\psi_{\mathcal{AS}}$ or $\delta_{\mathcal{AS}}$.

These encodings can therefore be used to compute the *diameter* of an authorized workflow schema in terms of the minimal number of users required for realizing it.

4.6 Analyzing resiliency of realizability

Suppose that an authorized workflow schema is realizable. Then there is a mapping from tasks to users that meets all constraints. But there then is the question of how brittle such realizability is. For example, do we lose the realizability of that schema if we remove one user, two users, etc.?

This question has been studied in the literature under the name of *resiliency*, and here we focus on *static, level-1* resiliency [20]. An algorithm to decide whether a workflow has level-1 resiliency takes a natural number $k > 0$ and an authorized workflow schema $\mathcal{AS}$ as arguments and decides whether each workflow obtained from $\mathcal{AS}$ by removing an arbitrary set of users of size k is still realizable.

The worst-case complexity for this problem is typically NP-hard and in $\mathrm{coNP}^{\mathrm{NP}}$ [20]. But our abstract encoding can be used to compute a number k such that the workflow is resilient to the removal of at most k users. This bound then under-approximates the degree of resiliency of the workflow.

So let π be a model of $\psi_{\mathcal{AS}}$ and let Γ be the function constructed as described above. We define the safe resiliency level $\mathsf{res}(\Gamma)$ of function Γ as follows:

$$\mathsf{res}(\Gamma) \stackrel{\text{def}}{=} \min_{t \in T} |\,\Gamma(t)\,| - 1$$

So $\mathsf{res}(\Gamma)$ is the smallest size of all such sets $\Gamma(t)$ where t is in T, minus 1.

We then know that we can remove $\mathsf{res}(\Gamma)$ or fewer users from that system without compromising the realizability of that workflow. The reason for this is simple: such a removal guarantees that each set $\Gamma(t)$ where t is from T is still non-empty when such users have been removed from it. And so we can still construct a choice function α from that modified version of Γ as part of an authorized plan.

Let us illustrate this using our example where $Alice$ has role FinAdm, and Bob and $Carol$ have role FinClerk. Suppose that we introduce new users $Pamela$ and $George$ with role FinClerk, and also $David$ with role FinAdm.

One possible Γ computed from our abstract encoding is

$$\begin{aligned} \Gamma(t_2) = \Gamma(t_6) &= \{Alice, David\} \\ \Gamma(t_1) = \Gamma(t_5) &= \{Bob, George\} \\ \Gamma(t_3) = \Gamma(t_4) &= \{Carol, Pamela\} \end{aligned} \tag{15}$$

Since the minimal size of all these sets is 2, we get from this Γ a safe resiliency measure $\mathsf{res}(\Gamma) = 1$. That is, we may remove one user from this system without compromising the realizability of this workflow.

An interesting problem in this context is how can we achieve a certain level of resiliency while adding as few users as possible to the user population: here, for example, we added one manager and two clerks; we would like to be able to assert that no smaller changes to the user base can give us a resiliency bound $\mathsf{res}(\Gamma) = 1$. This will be the subject of future work.

4.7 Number of solutions

We can now express the number of solutions captured by function Γ computed in the same manner as for $\psi_{\mathcal{AS}}$ and for $\delta_{\mathcal{AS}}$. That number is

$$\mathsf{nos}(\Gamma) = \prod_{t \in T} |\,\Gamma(t)\,| \tag{16}$$

as for each t there are as many choices as there are elements in set $\Gamma(t)$, and as these choices are independent in t. For Γ defined in (15), we have that $\mathsf{nos}(\Gamma)$ equals 64.

5 Experimental evaluation for our LTL(F) encodings

In [10], a detailed study was conducted that investigated the effectiveness of various tools as a means of deciding the satisfiability of LTL formulas that were randomly generated or drawn from established benchmarks.

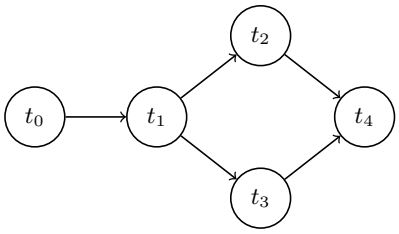
One crucial insight of that work was that the symbolic approach in model checkers such as NuSMV is manifestly superior to approaches that use explicit state spaces. We therefore decided to use the symbolic model checker NuSMV (version 2.5.3) for our experiments, and to use it in its bounded model-checking mode (which makes use of the MiniSat SAT solver).

5.1 Random model generator

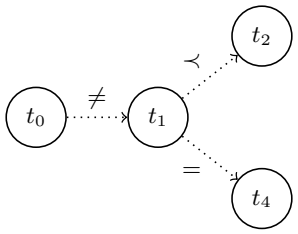
At present we do not have direct access to sufficiently many workflow authorization schemas as they occur in practice. This appears to be a known problem [22]. Therefore, we decided to randomly generate such workflow schemas for the experimental evaluation of our three reductions $\phi_{\mathcal{AS}}$, $\psi_{\mathcal{AS}}$, and $\delta_{\mathcal{AS}}$.

In our synthetic schemas, all constraints have the form $(U, t \rightarrow t', \rho)$, where ρ is in $\{=, \neq, \leq\}$. In particular, the constraint domain is always the set of users (so a constraint is always applied). A test configuration is defined by four parameters:

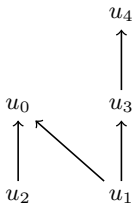
- number of tasks n_t, determining the size of set T
- number of users n_u, determining the size of set U
- authorization density p_a (that is the cardinality of the authorization relation $A \subseteq T \times U$ relative to that of $T \times U$ expressed as a percentage)



(a) Task ordering



(b) Constraints



(c) Seniority relation

$$A = \{(t_0, u_4), (t_1, u_0), (t_1, u_4), (t_2, u_3), (t_2, u_4), (t_3, u_0), (t_3, u_1), (t_4, u_1), (t_4, u_2), (t_4, u_3)\}$$

(d) Authorization relation

Fig. 5. Constrained workflow created by our random model generator

– constraint density p_c (that is the cardinality of the constraint set $C \subseteq T \times T$ relative to that of T expressed as a percentage)

The configuration $(100, 100, 20\%, 10\%)$, for example, denotes 100 tasks and users, 20% authorization density (meaning the cardinality of A is 2000), and 10% constraint density (which yields 10 entailment constraints in this case). Given the configuration $(5, 5, 40\%, 60\%)$, for example, our random model generator might construct the model depicted in Figure 5.

5.2 Experimental setup

Our reduction of $\mathsf{LTL}(\mathsf{F})$ satisfiability checking to model checking generates transition systems M_{AP} depicted in Figure 6 and computes $M_{\mathsf{AP}} \models \neg\eta$ with the NuSMV model checker, where η is one of the formulas $\phi_{\mathcal{AS}}$, $\psi_{\mathcal{AS}}$ or $\delta_{\mathcal{AS}}$. All tasks and users are declared as Boolean variables. The two `INIT` statements ensure that all states are initial, and that all states have transitions to all states. Finally, the keyword `LTLSPEC` specifies the formula to be model checked, in this case $\neg\eta$.

If a model check fails, the model checker will report a "counterexample", a finite description vw in S^* of an infinite path $\pi = vw^\omega$ where $\pi \not\models \neg\eta$. But this then means that $\pi \models \eta$. When $\eta = \phi_{\mathcal{AS}}$, for example, each state in vw contains exactly one task and one user, from which we can construct the α component of an authorized plan. On the other hand, if a model check succeeds, then we know that η (and thus $\mathcal{AS}$) is unsatisfiable.

```
MODULE main
VAR
t0 : boolean;
...
tn : boolean;
u0 : boolean;
...
um : boolean;

INIT
TRUE

TRANS
TRUE

LTLSPEC !(η)
```

Fig. 6. NuSMV code for satisfiability checking of η in $\{\phi_{\mathcal{AS}}, \psi_{\mathcal{AS}}, \delta_{\mathcal{AS}}\}$, where T equals $\{t_0, \ldots, t_n\}$ and $U = \{u_0, \ldots, u_m\}$.

We now report and discuss our results for several experiments with model checking

$$M_{\mathsf{AP}} \models \neg\eta \quad \text{where} \quad \eta \in \{\phi_{\mathcal{AS}}, \psi_{\mathcal{AS}}, \delta_{\mathcal{AS}}\} \tag{17}$$

on randomly generated instances of $\mathcal{AS}$. We do not report the running times needed for the linear-time topological sort of temporal task executions for the encodings $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$, whereas the running times for $\phi_{\mathcal{AS}}$ subsume such sorting as it is enforced in the encoding.

We studied the configuration space

$$\begin{aligned}\{(n_t, n_u, p_a, p_c) \mid\ & n_t = n_u, \\ & n_t \in \{10, 20, \ldots, 150\}, \\ & p_a \in \{100\%, 50\%, 10\%\}, \\ & p_c \in \{5\%, 10\%, 20\%\}\}\end{aligned} \tag{18}$$

where we incremented the number of tasks and users by 10 from the initial 10 up to 150.

For each such configuration, we ran and generated 10 random models and averaged results in order to smooth out random effects. We then wrote and used a prototype tool that converts such models into NuSMV models as depicted in Figure 6. We chose these configurations based on the intuition that the computational complexity of the models should increase, as p_a reduces, and it should increase again, as p_c increases. Also, as the models increase in size (number of tasks and users), the expected complexity should increase.

Our experiments were conducted on a server that has an Intel Core 2 Duo 2.8GHz (dual core) CPU with 4G of memory. The experiment consists of multiple runs, one for each configuration and each encoding $\psi_{\mathcal{AS}}, \delta_{\mathcal{AS}}$, and $\phi_{\mathcal{AS}}$. These runs are conducted with the NuSMV model checker in version 2.5.3 against every model, for each combination of configuration and encoding, in ascending order of configuration size.

We use the bounded model-checking mode (which relies on the MiniSat SAT solver) because the BDD-based approach for fully connected models performed poorly, and so we do

not discuss it here. In our experiment, this bound is set to n_t, as η is either unsatisfiable or has a model of length no greater than n_t.

If NuSMV takes more than 20 minutes to verify a model, a background job will terminate the NuSMV process automatically, so the run can move on to the next model check. If more than three automatic terminations occur in succession for a given configuration, we deem that an unacceptable running time has been reached for that and larger configurations. We then manually terminate the run and record the number of tasks when manual termination is required. We refer to this number informally as the *magic boundary*.

We ran additional experiments on a more fine-grained configuration space, as "sanity checks" on our random model generator. These experiments (not reported here) confirmed our intuition that formulas become easier to solve and are more likely to be satisfiable when p_a increases, and that they become harder to solve and less likely to be satisfiable as p_c increases.

Moreover, we used unit testing code on "counterexamples" extracted for all three encodings to independently verify the correctness as realizability witnesses of their models $\mathcal{AS}$. For $\phi_{\mathcal{AS}}$, unit tests independently verified all non-temporal constraints of $\mathcal{AS}$; temporal constraints were verified manually on small test cases. For $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$, unit tests verified non-temporal constraints in a manner consistent with the structure of Γ. For example, if t_1 and t_2 are constrained to have the same user allocated, then we checked whether the states in which t_1 and t_2 occur have a shared user true at these states.

5.3 Scalability and running times

We investigate how well model checking of these encodings scales in the number of tasks and constraint densities. Figure 7 shows a summary of our scalability experiments for all three encodings.

For each encoding, the figure shows the average running times for all satisfiable models of a configuration type. These data points are interpolated as a piece-wise linear plane where one dimension is the number of tasks (which is equal to the number of users) and the other dimension is the (increasing) density of constraints, denoted by xx-yy, where xx is the percentage of p_a, and yy is the percentage of p_c.

If the model checker ran out of memory or if a time out of 20 minutes was reached for one of the models of a configuration type, the figure does not show a data point. For example, in Figure 7, the runtime data for $\phi_{\mathcal{AS}}$ is absent for those configurations containing more than 30 tasks.

The plot in Figure 7 clearly shows that the encoding $\phi_{\mathcal{AS}}$ is not feasible beyond 30 tasks, and cannot even solve some models with 30 tasks if the density of constraints is sufficiently high. These results also demonstrate that the more abstract encodings $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$ fare much better in that they can solve models with 150 tasks and users if the constraint density is not too high. For example, the encodings $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$ are able to handle nearly all of the models we generated

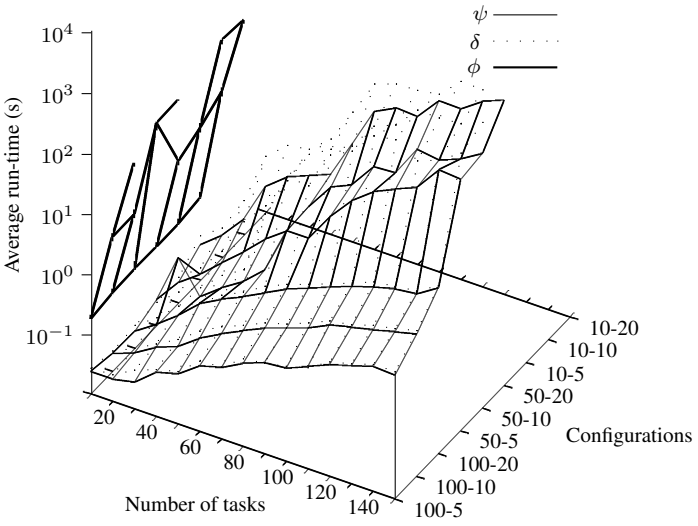


Fig. 7. Average running times of satisfiable models $\mathcal{AS}$ for all three encodings $\phi_{\mathcal{AS}}$, $\psi_{\mathcal{AS}}$, and $\delta_{\mathcal{AS}}$ for increasingly dense constraints.

when p_a is 100% and 50%. Even when p_a is 10%, model checks for these abstract encodings are able to reach a magic boundary of 70.

We then explored how far we can push the magic boundary for the abstract encodings. Generally, we found that $\delta_{\mathcal{AS}}$ takes longer to model check than $\psi_{\mathcal{AS}}$ (this is more apparent when configurations increase in complexity). For configuration types of form $(\cdot, \cdot, 50\%, 10\%)$, we seem to reach the limits of NuSMV on our test machine when model checking $\delta_{\mathcal{AS}}$ for 170 tasks and users, and 220 tasks and users for the $\psi_{\mathcal{AS}}$ encoding.

In Figure 8, we report average running times for those randomly generated models $\mathcal{AS}$ that are unsatisfiable. Note that the running time data for unsatisfiable models are sparse, because for configurations with high p_a, most of the models are satisfiable (e.g. when p_a = 100%, there are only two unsatisfiable models generated in the experiment, and only 12 in total when p_a = 50%).

Also, unsatisfiable models in general take much longer to verify, therefore, they are more likely to be automatically terminated due to the timeout strategy adopted in the experiment. Despite these issues, the available running time data for unsatisfiable models in Figure 8 clearly show that the average running time for NuSMV to complete the checks for unsatisfiable models increases, as the complexity of the configuration, as well as the model size increase.

5.4 Three measures for abstract encodings

We now report our experimental findings for the three measures $\mathsf{mub}(\Gamma)$, $\mathsf{res}(\Gamma)$, and $\mathsf{nos}(\Gamma)$ for the functions Γ extracted from models of the encodings $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$.

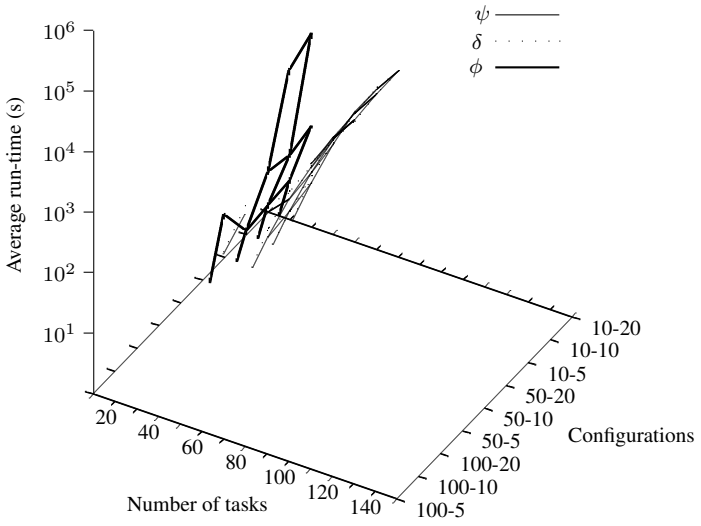


Fig. 8. Average running times of unsatisfiable models $\mathcal{AS}$ for all three encodings $\phi_{\mathcal{AS}}$, $\psi_{\mathcal{AS}}$, and $\delta_{\mathcal{AS}}$ for increasingly dense constraints.

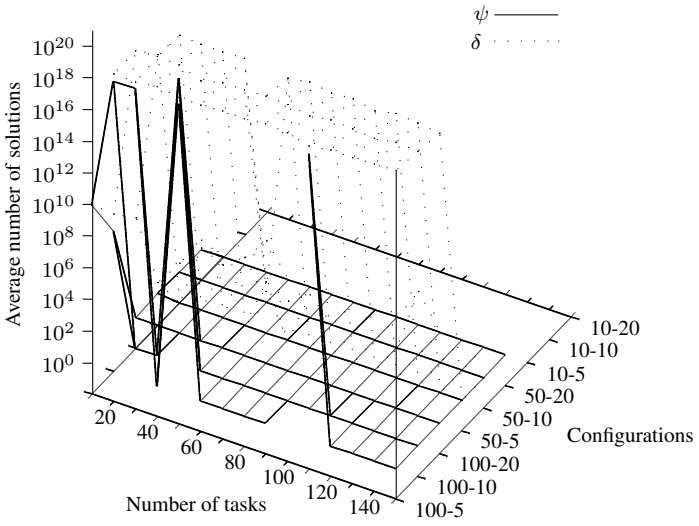


Fig. 9. Average number of solutions $\mathsf{nos}(\Gamma)$ for satisfiable models $\mathcal{AS}$ for the encodings $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$ for increasingly dense constraints.

We already saw that these encodings increase the size of realizability problems we can handle. As already stated, we use the same configuration space and indeed the same set of randomly generated models as for model checking with the encoding $\phi_{\mathcal{AS}}$.

Figure 9 shows our experimental data for the average number of solutions proposed by the NuSMV counter examples. This figure uses the same two dimensions for the piecewise linear planes (the number of tasks/users, and the density of constraints), and plots the average value of $\mathsf{nos}(\Gamma)$ as third dimension. From the figure, we make the following observations:

- The $\delta_{\mathcal{AS}}$ encoding does indeed give rise to more solutions, and may therefore give planners more flexibility in scheduling users for tasks. In many cases, $\psi_{\mathcal{AS}}$-encoded models return only one solution, whereas models of $\delta_{\mathcal{AS}}$ may have a number of solutions of order 10^{18} or more.
- As the configuration increases in complexity, the difference in $\mathsf{nos}(\Gamma)$ of $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$ encodings decreases to eventually zero. This suggests that as the constraints of the models become more restrictive, the "solution space" shrinks. This in turn, restricts the possible solutions proposed in the counter examples returned by $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$ encodings.

We now turn to the measure $\mathsf{res}(\Gamma)$ that computes a safe lower bound on the resiliency of satisfiable models $\mathcal{AS}$ under the removal of users.

Figure 10 shows our experimental data for the average resiliency $\mathsf{res}(\Gamma)$ of these workflows, using the same first two dimensions as used in previous figures. Apart from a small number of models, the resiliencies of the generated models are mostly 0. This suggests that the solutions extracted from

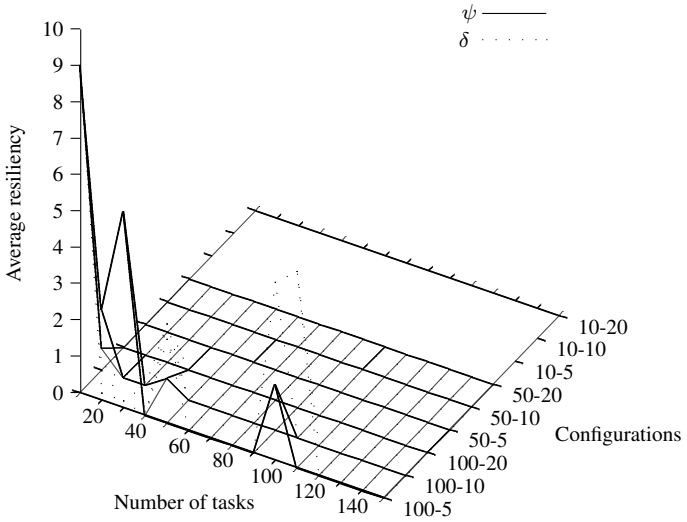


Fig. 10. Average resiliency of user bases under removal of users for satisfiable models $\mathcal{AS}$ for $\psi_{\mathcal{AS}}$ and $\delta_{\mathcal{AS}}$ for increasingly dense constraints.

the counter examples are already quite compact – meaning most of the proposed solutions do not have "unnecessary" users that can be removed and still remain valid. The highest average resiliency we observed in the experiment for all configurations is still below 10.

Figure 11 shows our experimental data for the average number of minimal users required to satisfy these workflows using the $\psi_{\mathcal{AS}}$ encoding. This figure uses the same first two dimensions as previous data charts, and plots the average $\mathsf{mub}(\Gamma)$ (number of minimal users) as third dimension. The

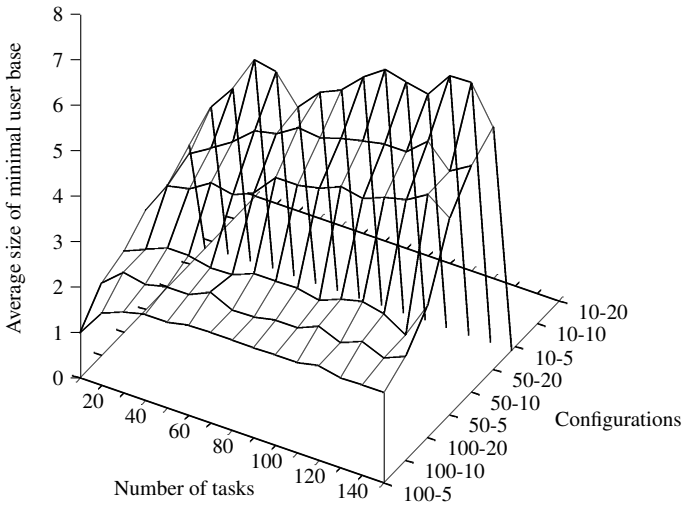


Fig. 11. Average size of minimal user bases for satisfiable models $\mathcal{AS}$ for the encoding $\psi_{\mathcal{AS}}$ for increasingly dense constraints.

minimal user base is an invariant of a workflow, so it is unnecessary to duplicate these data for the encoding $\delta_{\mathcal{AS}}$.

In Figure 11, we see that as configurations increase in complexity, the average $\mathsf{mub}(\Gamma)$ (minimum user base) also increases. This behavior is in line with the intuition that a greater number of users are required in order to fulfil a more demanding and restrictive task-user policy.

6 Workflow satisfiability in practice

We have demonstrated that we can express satisfiability of the workflow schema from [5] in $\mathsf{LTL}(\mathsf{F})$, and our experimental results suggest that these encodings have practical value.

Now we want to explore the shortcomings of the workflow model we studied as well as the limitations of $\mathsf{LTL}(\mathsf{F})$ as a tool for reasoning about workflow satisfiability. Specifically, we identify several situations of practical interest that cannot be represented using the workflow models for which satisfiability results are known.

6.1 Workflow execution patterns

The study of authorization constraints as a mechanism to enforce business rules such as separation of duty in workflow systems has assumed rather simplistic workflow specifications: Bertino *et al.* assume that the set of tasks is a list [2]; this has been extended to the analysis of workflows in which the set of tasks is partially ordered and a task may be executed several times [5].

However, these workflow models are not able to encode the richer workflow patterns that both occur in practice and have been studied in the workflow modeling community, notably in the work of van der Aalst and ter Hofstede [18]. In other words, research on workflow satisfiability needs to be extended to account for these richer workflow patterns.

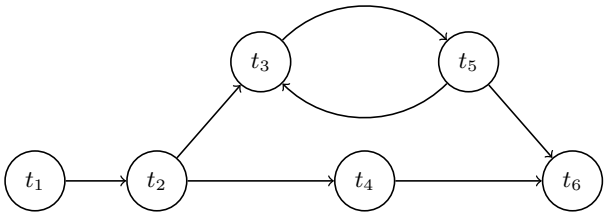


Fig. 12. A workflow specification with cycles

Suppose, for example, that items in a purchase order may be delivered separately. Then tasks t_3 and t_5 in our purchase order example may be executed multiple times in pairs. The resulting workflow specification is shown in Figure 12.

Moreover, existing approaches on workflow satisfiability in the presence of authorization constraints assume that the number of tasks is fixed and that all tasks are executed, although there may be some flexibility in the order in which tasks are performed. However, as we have seen in the example in Figure 12 the number of tasks that is executed in a workflow instance may vary.

Other common workflow patterns where the number of tasks is not pre-determined include OR-forks and OR-joins: in the former the execution of a task causes more than one task to enter the ready state, but only one of those tasks is required to be executed; in contrast, an OR-join only requires one of several preceding tasks to be executed for the next task to enter the ready state. To model OR-forks and OR-joins, we may generalize the form of formula $\phi_{\mathsf{F}T}$ such that it is in disjunctive normal form (DNF) where "atomic" propositions are of form $\mathsf{F}\,t$ with t from T. This allows us to model alternative workflows, where each term of the DNF represents a possible workflow and where only one of these workflows needs to be executed or synthesized.

For example, the formula

$$(\mathsf{F}\,t_1 \wedge \bigwedge_{i=2}^{10} \neg\mathsf{F}\,t_i) \vee (\neg\mathsf{F}\,t_1 \wedge \bigwedge_{j=2}^{10} \mathsf{F}\,t_j)$$

may express a non-deterministic choice between two possible workflows: a "normal" workflow in which tasks t_2 up to t_{10} occur (but t_1 does not), and an error handling workflow in which only task t_1 occurs.

All this suggests that foundations for workflow models need to be fairly expressive. In particular, $\mathsf{LTL}(\mathsf{F})$ cannot reason about the past and, more generally, temporal logics with support for past tense may still not suffice due to their inability to "count". The automata-theoretic extensions of temporal logics in [19] may give us sufficient counting abilities but the acceptance notions of automata cannot cope well with the presence of constraints such as those found within workflow specifications.

6.2 Workflow execution models

A workflow specification is instantiated by the workflow management system (WFMS) to create a *workflow instance*. Note that at any stage in the execution of a workflow instance, the (finite) set of tasks that have been completed is an order ideal in T.[2] The set of tasks that remain to be performed is, by definition, an order filter in T. Given an order ideal $I \subseteq T$, the set of *next* or *ready* tasks is defined to be the set of minimal elements in the order filter $T \setminus I$.

Two different modes of task selection and assignment are known for WFMSs:

- A *static* execution model assigns an (authorized) user to each task in a workflow when a workflow instance is created by the WFMS. (It is this execution model that has been assumed up to now in this paper.)
- A *dynamic* execution model assigns users to ready tasks during the execution of a workflow instance.

Any WFMS that employs a static execution model is in complete control of the allocation of tasks in a workflow instance to users, and performs the allocation when the workflow schema is instantiated. A WFMS that employs a dynamic execution model may also allocate tasks to users in an incremental way to those tasks that are ready to be executed. Such an execution model, with its support for late binding of users to tasks, is much more responsive to changes in the user base and in providing features such as load-balancing (in the sense of equitable user-task allocation). However, an alternative dynamic execution model is to maintain and advertise a list of ready tasks and allow users themselves to select (self-assign) tasks from that list.

In a static execution model, the satisfiability of the workflow specification is performed once and the task-user assignment list is a model for the workflow specification. In a dynamic execution model, the satisfiability of the workflow instance has to be checked when each task is assigned. A WFMS can perform this check by considering the satisfiability of a modified authorization schema.

More formally, let $\mathcal{AS} = (T, A, C)$ be a workflow authorization schema and let $I \subseteq T$ be an order ideal representing a set of completed tasks in a workflow instance W. Let $\alpha(t)$ represent the user that performed task t in I. Then $\mathcal{AS} \rhd I$, defined as (T, A', C) where

$$A' \stackrel{\text{def}}{=} A \cap \{(t, \alpha(t)) \mid t \in I\}$$

is also a workflow authorization schema (one in which there is a single authorized user for each task in I). Now suppose that tasks in I have been completed and u wishes to perform the ready task t. Then it suffices to compute the satisfiability of the workflow authorization schema

$$\mathcal{AS} \rhd (I \cup \{t\}).$$

[2] An order ideal $I \subseteq X$ in a partially ordered set $(X, \leqslant)$ has the property that if $x \in I$ and $y \leq x$, then $y \in I$. A set $F \subseteq X$ is an order filter if its complement $X \setminus F$ is an order ideal.

We can decide this by deciding the satisfiability of (say) $\phi_\Gamma \wedge \psi_{\mathcal{AS}}$ in $\mathsf{LTL}(\mathsf{F})$, where

$$\phi_\Gamma \stackrel{\text{def}}{=} \mathsf{G}\,(t \leftrightarrow u) \wedge \bigwedge_{t' \in I} \mathsf{G}\,(t' \leftrightarrow \alpha(t')).$$

From this we learn that this seemingly dynamic analysis has to query a static analysis before it can bind users to tasks at run-time.

Even when a static execution model is used, there may be situations in which we wish to assign particular tasks to particular users. One obvious reason is to ensure that user workloads are fair. Checking that a workflow remains satisfiable when a particular subset of tasks are assigned to particular users is no different from checking satisfiability at task selection time in a dynamic execution model. Formally, let Γ' be a context that assigns task t_i to user u_i for $i = 1, 2, \ldots, k$. In order to determine whether $\mathcal{AS}$ is satisfiable under these additional assumptions, we simply check for the satisfiability of $\phi_{\Gamma'} \wedge \psi_{\mathcal{AS}}$ in $\mathsf{LTL}(\mathsf{F})$, where we set

$$\phi_{\Gamma'} \stackrel{\text{def}}{=} \bigwedge_{i=1}^{k} \mathsf{G}\,(t_i \leftrightarrow u_i).$$

6.3 Richer workflow constraints

Existing approaches to workflow satisfiability assume that constraints are defined on pairs of tasks. In practice, we may wish to define threshold constraints. Such a constraint specifies a set of tasks $T' \subseteq T$ and an integer $k \leqslant |T'|$, which we denote by the pair (T', k). We suggest two possible and relevant definitions of satisfaction for (T', k):

- (T', k) is satisfied if the assignment of users to tasks T' involves at least k users.
- (T', k) is satisfied provided no user performs k or more of the tasks.

With both interpretations, separation of duty can be modeled by letting T' have two elements and by setting $k = 2$. Although such constraints are, in principle, expressible in $\mathsf{LTL}(\mathsf{F})$ in a manner similar to the encoding of constraints in ϕ_C above, such encodings will grow exponentially in that parameter k.

Most existing research on workflow satisfiability assumes that the scope of a constraint is a workflow instance. This is clearly limiting for general needs in security, audit, and control. For example, we may require that different users execute the same task in different instances of a workflow schema. To the best of our knowledge, the only research in this area is the work of Warner and Atluri [21]. However, this work seems to consider rather artificial constraints and does not account for control-flow constraints.

$\mathsf{LTL}(\mathsf{F})$ is able to reason about relationships between instances of workflows. For example, suppose that we want to compute two authorized plans, one for the first instance of the workflow and then another one for a second instance. The idea would be to copy each atomic proposition a in a primed

version a'. Then $\phi'_{\mathcal{AS}}$ (say) is the formula $\phi_{\mathcal{AS}}$ except that each atom is written in its primed version. Synthesis of authorized plans for both instances under additional constraints reduces to satisfiability of $\mathsf{LTL}(\mathsf{F})$ formulas: for example, models of

$$\phi_{\mathcal{AS}} \wedge \phi'_{\mathcal{AS}} \wedge \bigwedge_{u \in U} (\mathsf{F}\,(t \wedge u) \rightarrow \mathsf{G}(t' \rightarrow \neg u))$$

yield plans for the unprimed and the primed instance of the workflow for $\mathcal{AS}$, and also ensure that designated task t is assigned a different user in the latter instance.

7 Related work

This paper is an extended and adapted version of an informal workshop paper [8] that was subsequently published in amended form at a workshop with formal proceedings [7]. The work reported in this paper significantly extends that prior work by providing detailed experimental studies for model checking the LTL encodings of workflow satisfiability from [8, 7]. But these results then led us to the formulation of two new and more abstract encodings reported in our paper. For these novel encodings we provide much more favorable experimental results, and derive useful measures such as the resiliency of solutions and the sizes of minimal user bases required for solutions. And they allow us to compute a representation of an entire set of authorized plans through the use of bounded model checking.

The use of LTL for the verification of process-aware information systems has already been proposed in [17]. In that setting, the goal was to establish the consistency of the process-aware information system with a set of constraints on the control flow, rather than user-task constraints. We therefore cannot really make any informed comparison to that work. Having said that, initial attempts to check such consistency through LTL model checking apparently could handle only systems with five to ten tasks and had very high running times. In subsequent work [22], several algorithms were suggested in order to improve running times and increase the number of tasks and constraints one can handle. The experimental results in [22] suggest that these algorithms can now considerably speed up consistency checks and handle up to 70 tasks and 50 constraints. But these experiments were confined to randomly generated models that are *satisfiable*, the rationale being that real workflows are either satisfiable or where small changes to them will make them satisfiable.

Our approach reported in this paper is quite different. Firstly, we use a standard tool, the NuSMV bounded LTL model checker, for our consistency analysis of workflows. Secondly, we consider various encodings of realizability in the NP-complete front-end language $\mathsf{LTL}(\mathsf{F})$, whereas the encodings in [22] are in a PSPACE-complete fragment of LTL. Thirdly, our more abstract encoding not only may provide more than one witness for realizability but also implicitly computes a minimal user base and a sound measure of the resiliency of consistency. Fourthly, we are able to handle at least as many tasks and constraints as the above approach. Fifthly, we also considered unsatisfiable models to stress test our encodings. This is useful as workflows that are satisfiable once small changes have been made still need to be recognized as being unsatisfiable first.

Declarative models for workflows have also been studied in [9], where distributed dynamic condition response graphs are proposed as a declarative model for event-based workflows. These models generalize prime event structures [23]. Nodes in these graphs represent events, and edges in these graphs denote conditions, responses, inclusions or exclusions of events. This approach also supports analysis by transforming such graphs into Büchi automata.

8 Conclusions and future work

Our use of temporal logic $\mathsf{LTL}(\mathsf{F})$ for workflow realizability demonstrated the benefits of having a formalism in which many different analyses can be expressed as formulas of $\mathsf{LTL}(\mathsf{F})$, and where the analysis algorithm is generic and then instantiated with parameter instances (here a satisfiability solver for $\mathsf{LTL}(\mathsf{F})$).

We feel this is essential in order to make analysis capabilities less brittle under change of specifications. As corroborating evidence we cite parameterized approaches to shape analysis of programs [3], and ways of understanding program analyses as instances of model checking [14].

We proposed three encodings of workflow satisfiability in $\mathsf{LTL}(\mathsf{F})$ for a prominent workflow authorization schema from the literature [5]. These encodings are increasingly more abstract, and our experiments show that more abstract encodings yield better scalability of model checking workflow satisfiability.

We also saw the two more abstract encodings may be used to compute the minimal number of users needed for satisfying an authorized workflow schema (and to indeed compute such a set of users). Moreover, these encodings gave us safe bounds on how many users we may remove without compromising such satisfiability.

The "counterexamples" computed for both abstract encodings were shown to partition tasks and to partition a subset of users that could be allocated. This meant that "counterexamples" represent a Cartesian abstraction of an entire set of authorized plans for the encoded workflow schema.

The second abstract encoding just added a conjunct about realizing all users to the other abstract encoding. Our experiments also showed that this addition can yield models that represent many more solutions than the models of the other abstract encoding.

We now discuss directions for future work.

Richer workflows and $\mathsf{LTL}(\mathsf{F})$. We want to study whether $\mathsf{LTL}(\mathsf{F})$ is expressive enough to deal with workflows in which tasks are not given by a partial order but as an expression in a declarative language in which tasks are composed sequentially, where sets of tasks may be chosen non-

deterministically (OR/AND-joins and OR/AND-forks, for example), and where workflows may be repeated.

Pattern-based modeling and verification. The identification of *patterns* for workflow systems [18,12] needs to be extended to patterns for security, audit, and control. Declarative languages for workflows then ought to be able to express these additional patterns, and programs written in such languages have to be analyzable for pattern compliance and consistency. We already suggested that temporal logic and its automata-theoretic extensions alone will be inadequate for supporting these analysis capabilities.

Workflows and collaboration in the cloud. As software and platform alike increasingly become services hosted in the cloud, workflows and their management systems need to be realizable in clouds as well. Related to that, workflows from different organizations will have to be composed so that the composition meets the constraints of each organization. How to do such compositions is a challenging problem since organizations may not share all their constraints with each other, suggesting a workflow satisfiability problem under *imperfect information* and for *multiple agents*. Here, we think that methods from algorithmic game theory may be of use.

Need for empirical comparison of methods. We also think it is important to critically evaluate modeling and verification methods for workflow satisfiability in terms of their expressiveness and scalability. For example, while $\mathsf{LTL}(\mathsf{F})$ has NP-complete satisfiability checks, the problem's complexity becomes PSPACE-complete as soon as we add, say, a temporal operator for "Next State". But empirical data are needed in order to determine whether these worst-case complexities have any bearing on deciding the satisfiability of workflows as they arise in practice. We therefore want to conduct experiments on non-random workflow systems next.

Reference monitors using model checking. Our abstract encoding $\delta_{\mathcal{AS}}$ tends to produce many solutions to workflow satisfiability, and expresses that large solution space compactly as a finite word over $2^T \times 2^U$. We mean to explore how this representation and its computation can be used to facilitate the dynamic allocation of tasks without compromising workflow satisfiability. A first idea is to implement a reference monitor for workflows that are statically satisfiable. To then compute the abstract representation of solutions from $\delta_{\mathcal{AS}}$. And to then use that solution as a look-up table for access-control decisions of the reference monitor.

Optimized LTL *satisfiability checks.* The initial investigations into the utility and scalability of LTL satisfiability tools [10] led to a follow-up study [11]. That recent work presents new encodings of symbolic, transition-based Büchi automata, and novel BDD variable orderings informed by the parse tree of an LTL formula. Detailed experiments show that these new techniques can significantly improve the scalability of symbolic LTL satisfiability checking. Therefore, it would be of interest to see to what extent these improvements provide for a more scalable analysis of our workflow encodings.

References

1. Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. Boolean and Cartesian abstraction for model checking C programs. *STTT*, 5(1):49–58, 2003.
2. E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, 1999.
3. Igor Bogudlov, Tal Lev-Ami, Thomas Reps, and Mooly Sagiv. Revamping TVLA: Making parametric shape analysis competitive. In *Proceedings of the 19th International Conference on Computer Aided Verification*, pages 221–225, Berlin, Heidelberg, 2007. Springer-Verlag.
4. Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *CAV*, pages 359–364, 2002.
5. J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, pages 38–47, 2005.
6. J. Crampton and H. Khambhammettu. Delegation and satisfiability in workflow systems. In I. Ray and N. Li, editors, *Proceedings of 13th ACM Symposium on Access Control Models and Technologies*, pages 31–40, 2008.
7. Jason Crampton and Michael Huth. On the modeling and verification of security-aware and process-aware information systems. In Wil van der Aalst and Rafael Accorsi, editors, *Proc. BPM Workshop on Workflow Security Audit and Certification*, Lecture Notes in Business Information Processing, Clermont-Ferrand, France, August 2011. Springer. To appear.
8. Jason Crampton and Michael Huth. Synthesizing and verifying plans for constrained workflows: Transferring tools from formal methods. In Saddek Bensalem and Klaus Havelund, editors, *Proc. of Int'l Workshop on Verification and Validation of Planning and Scheduling Systems*, Freiburg, Germany, June 2011.
9. Thomas T. Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *PLACES*, page 59, 2010.
10. Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. *STTT*, 12(2):123–137, 2010.
11. Kristin Y. Rozier and Moshe Y. Vardi. A multi-encoding approach for LTL symbolic satisfiability checking. In *Formal Methods*, pages 417–431, 2011.
12. Nicholas Charles Russell. *Foundations of Process-Aware Information Systems*. PhD thesis, Faculty of Information Technology, Queensland University of Technology, December 2007.
13. R. Sandhu, E.J. Coyne, H. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
14. David A. Schmidt and Bernhard Steffen. Program analysis *is* model checking of abstract interpretations. In *Proceedings of 5th International Symposium on Static Analysis*, pages 351–380, 1998.
15. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32:733–749, July 1985.

16. K. Tan, J. Crampton, and C. Gunter. The consistency of task-based authorization constraints in workflow systems. In *Proceedings of 17th IEEE Computer Security Foundations Workshop*, pages 155–169, 2004.
17. Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.
18. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
19. Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, November 1994.
20. Q. Wang and N. Li. Satisfiability and resiliency in workflow systems. In *Proceedings of 12th European Symposium on Research in Computer Security*, pages 90–105, 2007.
21. J. Warner and V. Atluri. Inter-instance authorization constraints for secure workflow management. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 190–199, 2006.
22. Michael Westergaard. Better algorithms for analyzing and enacting declarative workflow languages using LTL. In *BPM*, pages 83–98, 2011.
23. Glynn Winskel. Event structures. In *Advances in Petri Nets*, pages 325–392, 1986.