Synthesizing and Verifying Plans for Constrained Workflows: Transferring Tools from Formal Methods

Jason Crampton
Information Security Group, Royal Holloway
University of London
jason.crampton@rhul.ac.uk

Michael Huth
Department of Computing
Imperial College London
m.huth@imperial.ac.uk

Abstract

Many business processes are modeled as workflows and workflow management systems are used to specify and coordinate the execution of those business processes. The execution of workflows is often constrained, e.g. by business rules, legal requirements or access control.

It is therefore important to know whether a workflow specification is consistent and so implementable. This question of workflow satisfiability has been studied by the computer security community in the past. But the solutions produced tend to be tailored to a particular workflow model and don't, therefore, adapt easily to other models or to richer forms of analysis, e.g. those between instances of the same workflow.

In this paper we demonstrate that tried and tested tools and techniques from formal methods, notably model checking based on linear-time temporal logic and its automata-theoretic extensions (Vardi and Wolper 1994), can be fruitfully transferred to this setting to provide more robust, uniform, and expressive foundations for creating and validating plans for authorized workflows. We also discuss the limitations of this and other formal approaches in trying to decide the satisfiability problem for richer workflow models and briefly explore how hybrid techniques might help solve the problem.

Introduction

Informally, a workflow is a set of logically related tasks that are performed in some sequence in order to achieve some business objective. A *workflow schema* is an abstract specification of the workflow tasks, indicating the order in which the tasks in every workflow instance of that schema should be performed. For each task in the schema, we identify a set of authorized users. 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 instance of the workflow. Such constraints may be imposed for varying 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. Note that a workflow specification need not be linear: the processing of the goods received note and of the invoice can occur in parallel.

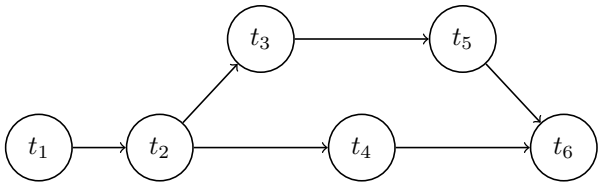
In addition to constraining the order in which tasks are to be performed, a number of business rules are specified to prevent fraudulent use of the purchase order processing system. These rules take the form of constraints on users that can perform pairs of tasks in the workflow: the same user may not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

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*. That is, there is no way of selecting an authorized user for each task in such a way that all the constraints are satisfied. Clearly, there is little practical value in an unsatisfiable workflow schema, so it is important that we can test whether a schema is satisfiable or not.

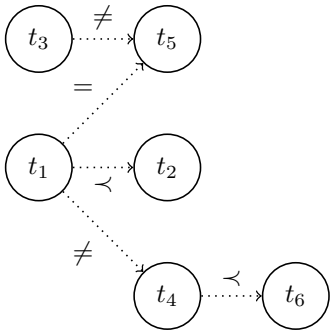
The problems of user-task assignment and workflow satisfiability have been studied by researchers in the security community using a variety of *bespoke* methods and algorithms (Bertino, Ferrari, and Atluri 1999; Crampton 2005; Wang and Li 2007). The bespoke nature of these methods makes it hard to assess whether they adapt easily to other, more expressive workflow models or to more sophisticated analyses, as we will illustrate in this paper.

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, for example, show that workflow satisfiability is NP-hard for many types of models, by exhibiting a simple reduction from graph-colorability to workflow satisfiability.[1] These complexity re-

[1] The reduction associates graph nodes with workflow steps and colors with authorized users in a rather natural way. As we shall see, workflow satisfiability is, in fact, a constraint satisfaction problem, and it is well known that graph-colorability can be reduced to a constraint satisfaction problem.



(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

Figure 1: A simple constrained workflow for purchase order processing

sults inform the search for more adaptable and general solution methods, since we can only choose methods with matching complexities.

In this paper we therefore explore whether tried and tested techniques and tools from the formal methods community can provide a more robust, more expressive, and more adaptable foundation for authorized workflow systems. We will particularly assess the approaches of model checking for linear-time temporal logic and its automata-theoretic extensions (Vardi and Wolper 1994), and of temporal logic satisfiability (Sistla and Clarke 1985).

We argue that the use of these techniques in that problem space has good to great potential. In particular, we demonstrate how an NP-complete fragment of linear-time temporal logic is a natural target language for several analyses of an important workflow model studied in the literature.

Outline of paper. We first provide technical background from temporal logic model checking and from workflow authorization schemes. Then we study workflow satisfiability for one such authorization scheme and show that it reduces to satisfiability in a well known fragment of propositional linear-time temporal logic. Next we look at workflow satisfiability in practice to assess the nature and scope of needs for solutions. In particular, we suggest that models based on temporal logics alone won't suffice in general. We then take a more detailed look at how formal methods may be fruitfully applied to model and solve constrained workflows as they may arise in practice. We conclude the paper with our overall assessment of the potential use of formal methods in this problem space.

Technical background

First we recall the definition of an important and representative model of workflow systems, and the definition of their satisfiability (Crampton 2005). Then we define the propositional linear-time temporal logic $\mathsf{LTL}(\mathsf{F})$ (Sistla and Clarke 1985) that allows us to reduce the workflow satisfiability problem to one of the satisfiability of a formula in $\mathsf{LTL}(\mathsf{F})$.

Constrained workflow authorization schemas

Definition 1 *A* constrained workflow authorization schema *is a triple* (T, 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'*;*[2]
- $A \subseteq T \times U$*, where* U *is a finite set of users and* $(t, u) \in A$ *is the* authorization relation*;*
- C *is a set of* entailment constraints*, tuples of form* $(D, t \to t', \rho)$ *where* $D \subseteq U$ *and* $\rho \subseteq U \times U$*.*[3]

The partial order $(T, \leq)$ expresses the constraints for linearizing tasks: any topological sort of T consistent with $\leq$ renders such a legal *linearization*. Relation $t \leq t'$ therefore specifies that task t must occur prior to task t'. Additionally, the partial order expresses which tasks (those incomparable with respect to $\leq$) may be executed independently from each other, e.g. concurrently.

Relation A specifies the authorization policy for the workflow. In particular, the informal interpretation of A is that u is authorized to perform task t if and only if $(t, u) \in A$.

A constraint $(D, t \to t', \rho)$ encodes a business rule by encoding restrictions on the users that can perform certain pairs of tasks. The constraint $(D, t \to t', \rho)$ requires that if $u \in D$ and $u' \in U$ are assigned to tasks t and t', respectively, then $(u, u') \in \rho$. Relation ρ and set D can vary with each element of C.

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* $(t, \alpha(t)) \in A$ *for all* $t \in T$*;*
   - L *is a linearization of* T*; and*

[2] Throughout, $<$ denotes the strict version of partial order $\leq$.

[3] We dropped the requirement that $t' \not\leq t$ since it does not seem to be required for the translation into temporal logic and we can imagine that it be useful to have entailments between tasks that are not merely temporal in nature or that may refer back to past tasks.

- *for all* $(D, t \to t', \rho) \in C$*, we have that if* $\alpha(t) \in D$ *then* $(\alpha(t), \alpha(t')) \in \rho$.

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

Authorized plans of constrained workflow authorization schemas are solutions of such schemas: they assign all tasks only to authorized users, provide a linearization of all tasks as a task scheduler, and ensure that the task assignment meets all entailment constraints.

Crampton introduced the notion of a *well-formed* workflow schema, and showed that if (L, α) is an authorized plan of $\mathcal{AS} = (T, A, C)$, 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 $\tilde{\rho}$ denotes the converse of ρ. The most natural type of relation to use in this situation would be one that is symmetric (whence $\rho = \tilde{\rho}$); both $\neq$ and $=$ are symmetric, for example.

We would expect that workflow schemata will be well-formed in many application domains. For such schemata, Crampton's result means 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 planning problem.

Nevertheless, there will be business processes that cannot be represented using a well-formed workflow schema. In the purchase order example illustrated in Figure 1, for example, it would be quite reasonable to impose constraints on t_3 and t_4 that would mean the resulting workflow schema was not well-formed: for example, $(U, t_3 \to t_4, \prec)$ and $(U, t_4 \to t_3, \neq)$ require that if the sign goods received note (GRN) task is performed before the create payment task, then the user that creates the payment must be more senior that the user that signs the GRN, whereas if the tasks are performed in the reverse order, we only require the users to be different (since the more commercially sensitive task has been performed first in this case).

Linear-time temporal logic $\mathsf{LTL}(\mathsf{F})$

We now show that the satisfiability problem for this workflow model can be expressed as a satisfiability problem in a fragment of propositional linear-time temporal logic that has NP-complete satisfiability checks.

Given a finite set AP of atomic propositions, the propositional temporal logic $\mathsf{LTL}(\mathsf{F})$ 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. The formal semantics of formulas is then given in Figure 2.

$\pi \models p$ iff $p \in s_0$
$\pi \models \neg\phi$ iff not $\pi \models \phi$
$\pi \models \phi_1 \wedge \phi_2$ iff ($\pi \models \phi_1$ and $\pi \models \phi_2$)
$\pi \models \mathsf{F}\,\phi$ iff there is some $i \geq 0$ such that $\pi^i \models \phi$, where π^i is the infinite suffix $s_i s_{i+1} \ldots$ of π

Figure 2: Formal semantics of temporal logic $\mathsf{LTL}(\mathsf{F})$ over infinite sequences of states. States are subsets of AP.

We use the usual abbreviations for disjunction ($\vee$), implication ($\rightarrow$), and the "Global" temporal connective $\mathsf{G}\,\phi$, which stands for $\neg\mathsf{F}\,\neg\phi$.

We say that a formula ϕ is *satisfiable* iff there is some infinite sequence of states π such that $\pi \models \phi$.

Expressing workflow satisfiability in $\mathsf{LTL}(\mathsf{F})$

In this section, 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 $\mathsf{LTL}(\mathsf{F})$) iff $\mathcal{AS}$ is satisfiable (in the sense given in Definition 2). This correspondence is constructive in that the respective models translate directly and in a meaningful way.

We first define the set of atomic propositions AP to be $\mathsf{AP} = U \cup T \cup \{\checkmark\}$, where we assume that sets U, T and $\{\checkmark\}$ are mutually disjoint. The atomic proposition $\checkmark$ models the fact that each task in T has been assigned a user.

Formula $\phi_{\mathcal{AS}}$ is a conjunction of formulas as depicted 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 prove this formally in Lemmas 1 and 2 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
- $\checkmark \in s_i$ means that state s_i does not schedule any tasks from T.

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

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 $\phi_{\mathsf{F}T}$ specifies that all tasks, including the $\checkmark$ "task" are eventually scheduled in π.

$$\phi_{\mathsf{F}T} = \bigwedge_{t \in T \cup \{\checkmark\}} \mathsf{F}\, t \tag{1}$$

$$\phi_{\mathsf{G}T} = \mathsf{G}\left(\checkmark \vee \bigvee_{t \in T} t\right) \tag{2}$$

$$\phi_{seT} = \bigwedge_{t \in T} \mathsf{G}\left(t \rightarrow \bigwedge_{t' \in T \cup \{\checkmark\} \setminus \{t\}} \neg t'\right) \tag{3}$$

$$\phi_{\mathsf{G}U} = \mathsf{G}\left(\checkmark \vee \bigvee_{u \in U} u\right) \tag{4}$$

$$\phi_{seU} = \bigwedge_{u \in U} \mathsf{G}\left(u \wedge \neg\checkmark \rightarrow \bigwedge_{u' \in U \setminus \{u\}} \neg u'\right) \tag{5}$$

$$\phi_{\leq} = \bigwedge_{t \in T} \mathsf{G}\left(t \rightarrow \mathsf{G}(\checkmark \vee \bigvee_{t' \not\leq t} t')\right) \tag{6}$$

$$\phi_{\checkmark} = \mathsf{G}\,(\checkmark \rightarrow \mathsf{G}\,\checkmark) \tag{7}$$

$$\phi_{A} = \bigwedge_{t \in T} \mathsf{G}\left(t \rightarrow \bigvee_{(t,u) \in A} u\right) \tag{8}$$

$$\phi_{C} = \bigwedge_{(D, t \rightarrow t', \rho) \in C} \phi_{(D, t \rightarrow t', \rho)} \tag{9}$$

$$\phi_{(D, t \rightarrow t', \rho)} = \bigvee_{u \in D} \left(\mathsf{F}\,(t \wedge u)\right) \rightarrow \mathsf{G}\left(t' \rightarrow \bigvee_{(u,u') \in \rho} u'\right) \tag{10}$$

$$\phi_{\mathcal{AS}} = \phi_{\mathsf{F}T} \wedge \phi_{\mathsf{G}T} \wedge \phi_{seT} \wedge \phi_{\mathsf{G}U} \wedge \phi_{seU} \wedge \phi_{\leq} \wedge \phi_{A} \wedge \phi_{C} \tag{11}$$

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

(2) Formula $\phi_{\mathsf{G}T}$ specifies that all states of π assign some task from T or assign the task $\checkmark$.

(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, not even the $\checkmark$ task.

(4) Formula $\phi_{\mathsf{G}U}$ specifies that all states of π either schedule the $\checkmark$ task or 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 and does not schedule task $\checkmark$, then that state does not assign any other users.

(6) Formula $\phi_{\leq}$ specifies, for each task t from T, that whenever task t is scheduled at a state in π, then all future states can only schedule task $\checkmark$ or a task t' such that $t' \not\leq t$.

(7) Formula $\phi_{\checkmark}$ specifies that $\checkmark$ models the completion of authorized plans: all states of π that schedule task $\checkmark$ are such that their infinite suffix of π contains only states that schedule $\checkmark$ as well.

(8) Formula ϕ_A encodes the authorization schema A. It specifies that for all tasks $t \in T$ and all states s_i of π, if s_i schedules task t, then state s_i assigns some user u with $(t, u) \in A$.

(9) Formula ϕ_C encodes the entailment constraints in C. It is a conjunction of formulas $\phi_{(D, t \rightarrow t', \rho)}$, one for each element of C.

(10) Formula $\phi_{(D, t \rightarrow t', \rho)}$ encodes any entailment constraint $(D, t \rightarrow t', \rho)$ of C. It is a disjunction with one disjunct for each user u from D. For each such user u, its disjunct 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.

It should be clear that most of these formulas over-approximate intended behavior of a constrained workflow authorization schema. For example, $\phi_{(D, t \rightarrow t', \rho)}$ ensures that there is some assignment of users that satisfies the entailment constraint. It does not prevent the assignment of users that violate these constraints. But it is the interaction of all conjuncts in $\phi_{\mathcal{AS}}$ that gives these approximations the desired precision.

Soundness of temporal logic encoding

We now show that the encoding of $\mathcal{AS}$ as $\phi_{\mathcal{AS}}$ is sound in that any authorized plan of $\mathcal{AS}$ naturally renders an infinite sequence that satisfies $\phi_{\mathcal{AS}}$.

Lemma 1 *Let $\mathcal{AS} = (T, A, C)$ be a constrained workflow authorization schema that is satisfiable. Then $\phi_{\mathcal{AS}}$ is satisfiable as well.*

Proof: Since $\mathcal{AS}$ is satisfiable, it has an authorized plan (L, α). Let the linearization L be $t_0 t_1 \dots t_n$. We construct an infinite sequence π of states $s_0 s_1 \dots$ from (L, α) as follows: for $0 \leq i \leq n$, we set $s_i = \{t_i, \alpha(t_i)\}$; and for all $i > n$, we set $s_i = \{\checkmark\}$. Intuitively, this infinite sequence schedules task t_i to user $\alpha(t_i)$ at state s_i and schedules task $\checkmark$ without assigning users at all other states of π.

We need to show that $\pi \models \phi_{\mathcal{AS}}$, i.e. that $\pi \models \psi$ for all conjuncts ψ listed in Figure 3.

(1) We have $\pi \models \phi_{\mathsf{F}T}$ since the first $n+2$ states of π witness the respective truth of every element in $T \cup \{\checkmark\}$.

(2) We have $\pi \models \phi_{\mathsf{G}T}$ since all states of π either satisfy $\checkmark$ or some t in T by definition of π.

(3) We have $\pi \models \phi_{seT}$ since each state of π schedules at most one task.

(4) We have $\pi \models \phi_{\mathsf{G}U}$ since each state either assigns a user $\alpha(t_i)$ or schedules task $\checkmark$.

(5) We have $\pi \models \phi_{seU}$ since each state that schedules a user does not schedule any other user.

(6) We have $\pi \models \phi_{\leq}$ since the linearization L is a topological sort with respect to $\leq$, and since π schedules these tasks in that order, and since $\pi \models \phi_{seT}$ holds as well.

(7) We have $\pi \models \phi_{\checkmark}$ by construction of π: all and only states s_i with $i > n$ schedule $\checkmark$.

(8) We have $\pi \models \phi_A$ since the authorized plan (L, α) respects the authorization relation A: for all $t \in T$, we have $(t, \alpha(t)) \in A$. Therefore, for each t from T we can choose $\alpha(t)$ as the u that makes the disjunction true in a state that schedules task t.

(9) We have $\pi \models \phi_C$ iff we have $\pi \models \phi_{(D, t \to t', \rho)}$ for each $(D, t \to t', \rho)$ from C.

(10) To conclude the proof, we show $\pi \models \phi_{(D, t \to t', \rho)}$, as follows. For each t from T, let $u \in D$ be given such that $\pi \models \mathsf{F}(t \wedge u)$. (There is nothing to show if there is no such u.) Then we know from the encoding of π that u must equal $\alpha(t)$. Since (L, α) is an authorized plan of $\mathcal{AS}$, this implies that $(\alpha(t), \alpha(t')) \in \rho$. Therefore, we can take $\alpha(t')$ as witness for the truth of the disjunct $\bigvee_{u' | (u, u') \in \rho} u'$. And, by construction of π, this disjunct is true at all states (of which there is only one) in which $\alpha(t')$ is scheduled. **QED.**

Completeness of temporal logic encoding

Now that we have shown that our encoding is sound, we also show that it is complete: a satisfiable temporal logic formula $\phi_{\mathcal{AS}}$ renders an authorized plan of the constrained workflow authorization schema $\mathcal{AS}$.

Lemma 2 *Let $\mathcal{AS} = (T, A, C)$ be a constrained workflow authorization schema and $\phi_{\mathcal{AS}}$ its corresponding encoding in* $\mathsf{LTL}(\mathsf{F})$. *If $\phi_{\mathcal{AS}}$ is satisfiable, then $\mathcal{AS}$ is satisfiable as well.*

Proof: Let $\phi_{\mathcal{AS}}$ be satisfiable. Then there is an infinite sequence of states π such that $\pi \models \phi_{\mathcal{AS}}$. We claim that we can construct from π an authorized plan (L, α) of $\mathcal{AS}$.

Let t be a task from T. Since $\pi \models \phi_{\mathsf{F}T}$, there is some state s_i of π such that $\pi^i \models t$. Let i_t be minimal with that property. Since $\pi \models \phi_{seT}$ we infer that $\pi^{i_t} \models \bigwedge\{\neg t' \mid t' \in T \cup \{\checkmark\} \setminus \{t\}\}$. From this we learn that state $\pi^{i_t} \not\models \checkmark$, i.e. that $\pi^{i_t} \models \neg\checkmark$. Combining this with the fact that $\pi \models \phi_{\mathsf{G}U}$ we obtain $\pi^{i_t} \models u$ for some $u \in U$. But $\pi \models \phi_{seU}$ together with the already established $\pi^{i_t} \models \neg\checkmark$ then ensures that there is exactly one $u \in U$ such that $\pi^{i_t} \models u$. Therefore, we set $\alpha(t) = u$ where u is that unique element of U.

Since $\pi \models \phi_{seT}$, it follows that $t \neq t^*$ implies $i_t \neq i_{t^*}$. Define L to be the list of tasks arranged in increasing order of their respective minimal indices. Then we claim that L is a linearization of $(T, \leq)$. Let $t, t^* \in T$ such that t occurs before t^* in L. Since t occurs before t^* in L, we have $\pi^{i_t} \models t$ with $i_t < i_{t^*}$. Moreover, since $\pi \models \phi_{\leq}$, we have $\pi^{i_t} \models \mathsf{G}(\checkmark \vee \bigvee_{t' \not\leq t} t')$. Hence, since $i_t < i_{t^*}$, we conclude that $\pi^{i_{t^*}} \models \checkmark \vee \bigvee_{t' \not\leq t} t'$. Now, by definition, $\pi^{i_{t^*}} \models t^*$, so we may conclude that $\pi^{i_{t^*}} \models \neg\checkmark$ and hence that $\pi^{i_{t^*}} \models \bigvee_{t' \not\leq t} t'$. Finally, since $\pi \models \pi_{seT}$ we conclude that the (unique) witness for the satisfiability of $\bigvee_{t' \not\leq t} t'$ is t^*. Specifically, we may conclude that $t^* \not\leq t$. Hence, if t precedes t^* in L, then we may conclude that either $t < t^*$ or that t and t^* are incomparable with respect to $\leq$, as required.

We now show that (L, α) is an authorized plan of $\mathcal{AS}$. For that, it remains to show that

1. for all t from T, we have $(t, \alpha(t)) \in A$ and
2. for all $(D, t \to t', \rho)$ from C, we have $(\alpha(t), \alpha(t')) \in \rho$.

For the first item, let t be in T. By definition, we have $\pi^{i_t} \models t$. Since $\pi \models \phi_A$ we therefore infer that $\pi^{i_t} \models \bigvee_{(t,u) \in A} u$. Let u be the witness from U for that truth. In particular, (t, u) is in A. Since $\pi \models \phi_{seU} \wedge \phi_{seT}$, this implies that this u must be $\alpha(t)$. And so $(t, \alpha(t))$ is in A as required.

For the second item, let $(D, t \to t', \rho)$ be from C. If $\alpha(t)$ is not in D, there is nothing to show. Otherwise, let $\alpha(t)$ be in D. Since $\pi^{i_t} \models t \wedge \alpha(t)$, where $\alpha(t)$ is in D, we obtain that $\pi \models \mathsf{F}(t \wedge u)$ for $u = \alpha(t)$ in D. But we also have that $\pi \models \phi_{(D, t \to t', \rho)}$. From both we then deduce that $\pi \models \mathsf{G}(t' \to \bigvee_{u' | (u, u') \in \rho} u')$. In particular, we get that $\pi^{i_{t'}} \models t' \to \bigvee_{u' | (u, u') \in \rho} u'$. By definition of $i_{t'}$, we also have $\pi^{i_{t'}} \models t'$. Therefore, we get that $\pi^{i_{t'}} \models \bigvee_{u' | (u, u') \in \rho} u'$. Thus, we get that $\pi^{i_{t'}} \models u'$ for some u' with $(\alpha(t), u') \in \rho$. Again, using that $\pi \models \phi_{seT} \wedge \phi_{seU}$ we infer that u' must equal $\alpha(t')$ and so $(\alpha(t), \alpha(t'))$ is in ρ as required. **QED.**

Discussion

The first lemma is easy to show, the encoding $\phi_{\mathcal{AS}}$ just has to specify things that are true of an authorized plan of $\mathcal{AS}$. The crucial result is the second lemma. In its proof we used all formulas in Figure 3, except for $\phi_{\checkmark}$. That formula (and indeed the use of the atomic proposition $\checkmark$) is therefore not needed to secure both lemmas. However, it should be useful in the satisfiability checks of $\phi_{\mathcal{AS}}$ as it ensures that all tasks are scheduled "early" in an infinite sequence π.

Related to that, the witnesses π that a satisfiability checker may produce may contain more than one state s_i that schedules task t. As the proof of the second lemma reveals, this is not a problem for correctness. It is also not a problem for the complexity of the approach since models of formulas in $\mathsf{LTL}(\mathsf{F})$ can be found that have size linear in the size of the formula. Therefore, this gives us only a linear blow-up over the non-redundant models (L, α).

In fact, an attempt to model the tasks that are scheduled in a unique state forces us to leave the fragment $\mathsf{LTL}(\mathsf{F})$. For example, one could add to $\phi_{\mathcal{AS}}$ the conjunct

$$\bigwedge_{t \in T} \mathsf{G}(t \rightarrow \mathsf{X}\,\mathsf{G}\,\neg t)$$

which states that whenever t is true in a state, it will be false in all subsequent states. Together with the requirement that t will be true eventually, this gives the desired effect that t be true at exactly one state. However, the fragment $\mathsf{LTL}(\mathsf{F}, \mathsf{X})$ that adds clause X for "neXt state" with meaning "$\pi \models \mathsf{X}\,\phi$ iff $\pi^1 \models \phi$" has PSPACE-complete satisfiability checks (Sistla and Clarke 1985).

The translation from $\mathcal{AS}$ to $\phi_{\mathcal{AS}}$ is also linear in the size of $\mathcal{AS}$. Therefore we get a generic way of solving satisfiability of such $\mathcal{AS}$ by means of temporal logic satisfiability.

Workflow satisfiability in practice

Having demonstrated that we can express satisfiability of Crampton's workflow schema in $\mathsf{LTL}(\mathsf{F})$, we now explore the deficiencies of Crampton's workflow model. Specifically, we identify several situations of practical interest that cannot be represented using existing workflow models for which satisfiability results are known.

Workflow 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 (Bertino, Ferrari, and Atluri 1999); Crampton extends the analysis to workflows in which the set of tasks is partially ordered (Crampton 2005). The work of van der Aalst and ter Hofstede on control flow and resource provisioning provides a good introduction to the modeling of real workflows (van der Aalst et al. 2003). Workflow satisfiability addresses concerns related to the control and resource perspectives of workflow modeling. However, there are several workflow patterns[4] for which the existing approaches on workflow satisfiability are ill-suited.

In other words, research on workflow systems suggests that we need to be able to support richer workflow control patterns, as described by van der Aalst and ter Hofstede and widely used in business processes. Suppose, for example, that different items in a purchase order may be delivered separately. Then the 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 4.

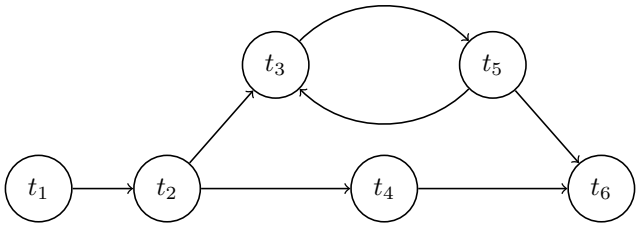


Figure 4: A workflow specification with cycles

Also, existing approaches on workflow satisfiability 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 4 the number of tasks that is executed in an 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.

YAWL ("yet another workflow language") was developed by van der Aalst and ter Hofstede as a language for defining workflow patterns (van der Aalst and ter Hofstede 2005). YAWL is based on Petri nets, a common modeling tool in the workflow research community, which makes it suitable for modeling the control perspective of workflows. However, it is unclear whether YAWL will be easy to integrate with authorization rules.

These works suggest that foundations for workflow models need to be fairly expressive. In particular, they suggest that temporal logics alone won't suffice due to their inability to "count".

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.[5] 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 have been identified 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.)

[4] van der Aalst and ter Hofstede maintain a website with numerous resources related to workflow patterns `http://www.workflowpatterns.com/`

[5] 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 $X \setminus F$ is an order ideal.

- A *dynamic* execution model assigns users to ready tasks in 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 but in an incremental way and based on dynamic needs. However, an alternative dynamic execution model is to maintain and advertise a list of ready tasks and allow users 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. We 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 $A' = A \cap \{(t, \alpha(t)) : t \in I\}$, 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 $\mathcal{AS} \rhd (I \cup \{t\})$.

In our LTL formalization, we can decide this by deciding the satisfiability of $\phi_\Gamma \wedge \phi_{\mathcal{AS}}$, where

$$\phi_\Gamma = \mathsf{G}\,(t \leftrightarrow u) \wedge \bigwedge_{t' \in I} \mathsf{G}\,(t' \leftrightarrow \alpha(t')) \tag{12}$$

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 $\phi_{\Gamma'} \wedge \phi_{\mathcal{AS}}$ where we set

$$\phi_{\Gamma'} = \bigwedge_{i=1}^{k} \mathsf{G}\,(t_i \leftrightarrow u_i) \tag{13}$$

Workflow constraints

Existing approaches on 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). There might be different notions of satisfaction for (T', k): below we suggest two possibilities.

- (T', k) is satisfied provided the assignment of users to tasks T' involves at least k users. With this interpretation, the separation of duty constraints considered thus far in this paper are a special case of threshold constraints, in which the cardinality of T' is 2 and $k = 2$.
- (T', k) is satisfied provided no user performs k or more of the tasks. The separation of duty constraints in this paper are also a special case of threshold constraints interpreted in this way, in which the cardinality of T' is 2 and $k = 2$.

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 definitions are not paramaterized in k and will grow exponentially in that parameter.

Most existing research on workflow satisfiability assumes that the scope of a constraint is a workflow instance. However, there are practical situations in which it might be necessary to insist 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 (Warner and Atluri 2006). However, this work considers rather artificial constraints and does not account for control flow constraints. Below, we demonstrate how temporal logic specifications can do this.

Modeling constrained workflows

In light of the observations in the preceding section, we now consider some of the tools that could be applied to problems of satisfiability in more complex workflow patterns. We start with the temporal $\mathsf{LTL}(\mathsf{F})$ already discussed above.

Expressiveness of $\mathsf{LTL}(\mathsf{F})$

We now demonstrate that this fragment of LTL can already give us a generic engine for modeling and solving a variety of problems in authorized workflow schemas. We begin by showing that $\mathsf{LTL}(\mathsf{F})$ can express OR-forks.

Expressing OR-forks For example, one may generalize the form of formula $\phi_{\mathsf{F}T}$ such that it is in disjunctive normal form 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. Since $\mathsf{LTL}(\mathsf{F})$ has NP-complete satisfiability checks, this more general problem stays within NP. It is straightforward to extend this modeling of *initial* OR-forks to those occurring at any point of the work flow, by making use of the G modality.

In contrast, OR-joins seem to require reasoning about the past. It is therefore not immediately clear whether this can be accommodated in $\mathsf{LTL}(\mathsf{F})$ as the encoding of past tense operators in LTL uses Untils.

Synthesis of residual plans However, if we want to check whether $\mathcal{AS}$ is satisfiable under the assumption that tasks were scheduled first, second, etc. (with or without assigned users), we need to consider another fragment $\mathsf{LTL}(\mathsf{X})$ of linear-time temporal logic generated by

$$\phi ::= p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathsf{X}\,\phi \tag{14}$$

where p is from AP. For example, if task t_1 is scheduled first, task t_2 scheduled second, and task t_3 scheduled third we can check the satisfiability of $\mathcal{AS}$ under that assumption by checking the satisfiability of $(t_1 \wedge \mathsf{X}\, t_2 \wedge \mathsf{XX}\, t_3) \wedge \phi_{\mathcal{AS}}$. This is a conjunction where the first conjunct is from $\mathsf{LTL}(\mathsf{X})$ and the second one is from $\mathsf{LTL}(\mathsf{F})$ and so the satisfiability problem is still in NP. This would be of use for workflow models that are not well formed in the sense we mentioned, where one cannot solve for L and α independently.

Plans with inter-instance constraints The approach of using temporal logics also supports the reasoning about relationships between instances of workflows. For example, suppose that we want to compute two 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 $\phi'_{\mathcal{AS}}$ is the formula $\phi_{\mathcal{AS}}$ except that each atom is written in its primed version. Now, we can synthesize plans for both instances that meet additional constraints. For example, an infinite sequence π which satisfies

$$\phi_{\mathcal{AS}} \wedge \phi'_{\mathcal{AS}} \wedge \bigwedge_{u \in U} (\mathsf{F}\,(t \wedge u) \rightarrow \mathsf{G}(t' \rightarrow \neg u)) \qquad (15)$$

gives us plans for the unprimed and the primed instance of the workflow for $\mathcal{AS}$, and it also ensures that designated task t is assigned a different user in the second instance.

Parameterized analyses The examples above suggest that the approach we propose is capable of supporting many forms of analysis through a single specification formalism (here $\mathsf{LTL}(\mathsf{F})$) and a single algorithmic engine (e.g. a model checker). In this sense, formulas of $\mathsf{LTL}(\mathsf{F})$ specify a particular analysis and the algorithm conducting this analysis comes for free from any model checker for that logic.

This advantage of parameterized analyses can also be seen in parameterized approaches to shape analysis of programs (Bogudlov et al. 2007), and in ways of understanding program analyses as instances of model checking (Schmidt and Steffen 1998).

We note that one has to abandon the NP-complete fragment $\mathsf{LTL}(\mathsf{F})$ if one needs to express Strong Untils, Weak Untils, and non-trivial uses of the Next operator. For example, would could use past-tense operators for an encoding of OR-joins, but the encodings of past-tense operators themselves use Untils.

Workflow automata

In formal methods, many modeling and solution methods reduce to fundamental decision problems for certain types of automata. It is therefore relevant to investigate whether automata-theoretic techniques can be applied to authorized workflow schemas.

Let $\mathcal{AS} = (T, A, \emptyset)$ be a workflow authorization schema. (In other words, there are no constraints on the execution of tasks.) We define the *workflow automaton* $\mathcal{M}_{\mathcal{AS}} = (Q, q_s, q_f, \Sigma, \delta)$ in the following way:

- the set of states $Q = \mathcal{I}(T)$, where $\mathcal{I}(T)$ is the set of order ideals in T;
- the start state $q_s = \emptyset \in \mathcal{I}(T)$;
- the final state $q_f = T \in \mathcal{I}(T)$;
- the alphabet $\Sigma = A$;
- the transition function $\delta : Q \times \Sigma \rightarrow Q$, where

$$\delta(I, (t, u)) = \begin{cases} I \cup \{t\} & \text{if } t \text{ is a next task} \\ I & \text{otherwise.} \end{cases}$$

Hence, given $\mathcal{AS} = (T, A, C)$, we may write $\mathcal{M}_{\mathcal{AS}} = (\mathcal{I}(T), \emptyset, T, A, \delta)$, or simply $\mathcal{M}_{\mathcal{AS}} = (\mathcal{I}(T), A, \delta)$.

Informally, the execution of a task t induces a state change. Note that the transition function is only defined for transitions from an order ideal to another order ideal, so an attempt to perform a task that is not a next task will not result in a state change.

The language accepted by a workflow automaton $\mathcal{M}_{\mathcal{AS}}$ is a string of $|T|$ pairs from A. Each string accepted by $\mathcal{M}_{\mathcal{AS}}$ is an authorized plan for $\mathcal{AS}$. Hence, a workflow is satisfiable iff the language accepted by $\mathcal{M}_{\mathcal{AS}}$ is non-empty.

This approach would seem to be well-suited to modeling complex workflow patterns; it can easily represent the workflow specification in Figure 4, for example. However, we have assumed that the set of constraints is empty.

Hence, we need to consider how we can extend the definition of a workflow automaton to account for a workflow schema $\mathcal{AS} = (T, A, C)$ such that $C \neq \emptyset$. The most obvious approach is to modify the definition of δ. However, it turns out that to modify δ one needs to specify conditions that relate to the assignment of users to earlier tasks. Hence, δ would no longer have the required signature for a finite state automaton. Another obvious approach is to use "brute force" and simply introduce many new states which are dependent on C, A and the tasks that have already been assigned a user. In effect, such an automaton will directly encode all possible models.

An alternative is to specify only the workflow using a finite state machine. In other words, the alphabet is just the set of tasks and the language accepted by the machine is simply the set of "legal" task executions. With this approach, we can specify workflow behavior that is considerably more complex than is possible when we model the workflow as a partially ordered set of tasks. We can, for example, model tasks that may be repeated one or more times or jump to an "abort" state. However, it is no longer possible to reason about authorization information and constraints. In other words, we cannot answer questions about workflow satisfiability using only a finite state machine of this form.

Workflow composition

A genuinely important thing to consider is the composition of workflows. We now give an example of such a composition language in Figure 5. This language has tasks t as atomic workflows, and uses three composition operators:

- W^+ takes a workflow and sequentially repeats it an unbounded but finite number of times,

- $V; W$ is sequential composition $V; W$ of workflows V and W, and
- choose k from $\mathbf{W}$ denotes any workflow that schedules exactly $k \geq 1$ many workflows from the non-empty set $\mathbf{W}$ of workflows.

Note that these compositions allows for nestings as in XML documents: the elements of $\mathbf{W}$ themselves may be complex workflows, not merely atomic tasks.

V, W ::=	*Workflows*
t	(Atomic Workflow)
W^+	(Unbounded Iteration)
$V; W$	(Sequential Composition)
choose k from $\mathbf{W}$	(Threshold Choice)

Figure 5: Grammar of a work-flow language: t ranges over a set of tasks T, threshold k is an integer greater than 0, and $\mathbf{W}$ refers to any non-empty, finite set of workflows generated by this grammar.

Example 1 *We can express the purchase order workflow from Figure 4 in this composition language as*

$$(t_1; t_2); \text{choose } 2 \text{ from } \{(t_3; t_5)^+, t_4\}; t_6 \qquad (16)$$

Figure 6 illustrates how program expression in (16) may be written in a more graphical form. It also suggests that graphical languages (with navigation capabilities across specified layers) will provide better interfaces for users than the concrete language defined above. We hope to learn valuable lessons here from hierarchical state machines.

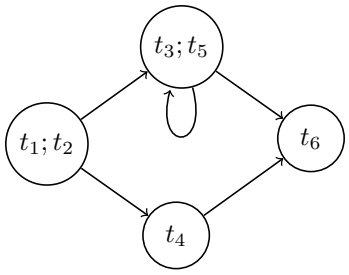


Figure 6: Composing workflow specifications

This example also demonstrates that choose k from $\mathbf{W}$ can express AND-forks and AND-joins when k equals $|\mathbf{W}|$, through appropriate composition contexts. This operator can also express an OR-fork by choosing $k = 1$ and $|\mathbf{W}| = 2$.

A variant of this operator would be when k refers to "at least k" completions of workflows from the choice set, modeling a quorum or a majority vote for example.

Our composition language clearly needs additional mechanisms for enforcing constraints that go beyond the static composition operators of the language, for example, separation of duty. Such constraints act across the composition boundaries of a composed workflow. Some of the basic questions here are:

- How to define constraints when tasks may be repeated an unbounded but finite number of times?
- Are there approaches to synthesizing plans for such workflows that exploit their compositional structure?
- Would a compositional technique yield better computational complexity than a non-compositional method?

The answer to the latter point is not always "yes". For example, in LTL model checking it is better to negate a formula prior to synthesis than it is to first synthesize the unnegated formula and to then complement the automaton (Vardi and Wolper 1994).

The automata-theoretic approach seems promising here, but the presence of constraints may prevent its full use. We turn next to another formal method that is custom-made for handling constraints.

Constraint satisfaction problems

Constraint satisfaction problems (CSPs) are well known in many areas, including combinatorics, operations research, data bases, and artificial intelligence. Such a problem is typically expressed via a set of constraints between formal variables and where solutions instantiate variables with elements from homogeneous value spaces such that all constraints are met. This suggests that the problems for authorized workflow systems may be expressed as constraint satisfaction problems, and that one may then rely on known solution techniques for constraint satisfaction problems.

More formally, a *constraint satisfaction problem* is defined by a set of variables X, a domain D and a set of constraints C (Tsang 1993). Each constraint is a pair (L, ρ), where $L = [x_1, \ldots, x_\ell]$ is a list of (distinct) variables taken from X and $\rho \subseteq D^\ell$. We refer to L as the *(constraint) scope* and to ρ as the *(constraint) relation*.

Given a function $\phi\colon X \to D$, we write $\phi([x_1, \ldots, x_\ell])$ for the tuple $(\phi(x_1), \ldots, \phi(x_\ell))$ in D^ℓ. We say that ϕ solves the constraint satisfaction problem (X, D, C) if for each constraint $(L, \rho) \in C$, $\phi(L) \in \rho$.

Example 2 *The triple* $(\{x_1, x_2, x_3\}, \{A, B, C\}, \{c_1, c_2\})$ *has* $c_i = ([x_i, x_{i+1}], \neq)$ *for* $i = 1, 2$ *(where the symbol* $\neq$ *denotes the relation* $\{(x, y) \in D \times D : x \neq y\}$*). Then the function* ϕ_1*, where* $\phi_1(x_1) = A$*,* $\phi_1(x_2) = B$ *and* $\phi_1(x_3) = C$ *is one solution to this constraint satisfaction problem. The function* ϕ_2*, where* $\phi_2(x_1) = A$*,* $\phi_2(x_2) = B$ *and* $\phi_2(x_3) = A$ *is also a solution, but* ϕ_3*), where* $\phi_3(x_1) = A$*,* $\phi_3(x_2) = B$ *and* $\phi_3(x_3) = B$ *is not.*

Clearly some constraint satisfaction problems do not have a solution. Equally clearly, the number of potential solutions grows exponentially with $|X|$. It can be shown that determining whether a constraint satisfaction problem has a solution is NP-complete (the problem is clearly in NP and many NP-complete problems, such as three-colorability, can be expressed as constraint satisfaction problems).

The CSP and workflow satisfiability

Let $\mathcal{AS} = (T, A, C)$ be a constrained workflow authorization schema. Let $c = ((t, t'), \rho)$ be a constraint in C. Then

define c^* to be the constraint

$$((t, t'), \{(u, u') \in U \times U : (t, u), (t', u') \in A, (u, u') \in \rho\})$$

Finding a model for a (well-formed) constrained workflow authorization schema $\mathcal{AS} = (T, A, C)$ is easily seen to be equivalent to solving the constraint satisfaction problem $\mathsf{CSP} = (T, U, C^*)$, where $C^* = \{c^* : c \in C\}$. However, this approach limits us to the consideration of workflow schema in which the set of tasks is fixed in advance. So results for CSPs can be applied directly to the constrained workflow authorization schema defined by Crampton, but are unsuitable for many of the workflow patterns (such as OR-fork and workflow repetition) identified in this section.

Many different varieties of constraint satisfaction problem have been identified. Of particular interest to us is *dynamic* constraint satisfaction problems (Mittal and Falkenhainer 1990), in which the set of constraints changes as more information becomes available. There is a substantial literature on the efficient solution of a dynamic constraint problem given a solution to some initial constraint satisfaction problem. Such methods should be of particular use when we have a dynamic workflow execution model, as the execution of a particular task instance by a particular user corresponds to a "tightening" of an existing constraint.

Satisfiability for complex workflow patterns

Once we have found a suitable model for complex workflow patterns and a way of specifying constraints on task execution we will need to find an appropriate way of determining whether a workflow schema is satisfiable. It is unlikely that complex workflow schemata, in which the set of tasks that will be executed in an instance of that workflow is not fixed, will be able to use the planning tools described *as is*.

It seems likely, for example, that a richer temporal logic or more complex automata and constraint satisfaction problems are needed for the production of partial plans for workflow instances or to test for workflow satisfiability.

In particular, we will not be able, in general, to produce a full user-task assignment plan when the number of tasks and the order in which they occur is more fluid. Therefore, we will need to be able to construct a component that can either produce user-task assignments for the ready tasks or determine whether an attempt by a user to perform a self-assigned task should be permitted.

An important area for future work, therefore, will be the investigation of appropriate models for such a component. One interesting area of work in this area is the application of discrete control theory to synthesize a controller that can guide the execution of workflows in such a way that unsafe states are avoided (Wang, Kelly, and Lafortune 2007).

Conclusions

We studied current approaches to modeling and synthesizing workflows in the presence of authorization specifications. We observed that extant work develops algorithmic techniques that are customized to particular models and target specific analysis or synthesis needs. These approaches are therefore bespoke, brittle under change of model or analysis, and lack expressive features needed for workflow specifications in real-world application domains.

We then pointed out that the tool set of formal methods offers expressive specification formalisms with generic analysis capabilities. Model checking of a fragment of linear-time temporal logic, for example, covers a range of analysis needs for an important model of workflow authorization schemes. We saw that automata, constraint satisfaction solvers, and formal compositional workflow languages are able to provide robust but incomplete foundations for the specification and synthesis of authorized work plans. So more research is needed to determine how these partial solutions can be integrated into a powerful platform for modeling and reasoning about a wide range of workflow models and their security.

References

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

Bogudlov, I.; Lev-Ami, T.; Reps, T.; and Sagiv, M. 2007. Revamping TVLA: Making parametric shape analysis competitive. In *Proceedings of the 19th International Conference on Computer Aided Verification*, 221–225. Berlin, Heidelberg: Springer-Verlag.

Crampton, J. 2005. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, 38–47.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *AAAI*, 25–32.

Schmidt, D. A., and Steffen, B. 1998. Program analysis *is* model checking of abstract interpretations. In *Proceedings of 5th International Symposium on Static Analysis*, 351–380.

Sistla, A. P., and Clarke, E. M. 1985. The complexity of propositional linear temporal logics. *Journal of the ACM* 32:733–749.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press Ltd.

van der Aalst, W., and ter Hofstede, A. 2005. YAWL: yet another workflow language. *Inf. Systems* 30(4):245–275.

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

Vardi, M. Y., and Wolper, P. 1994. Reasoning about infinite computations. *Information and Computation* 115:1–37.

Wang, Q., and Li, N. 2007. Satisfiability and resiliency in workflow systems. In *Proceedings of 12th European Symposium on Research in Computer Security*, 90–105.

Wang, Y.; Kelly, T.; and Lafortune, S. 2007. Discrete control for safe execution of IT automation workflows. In *Proceedings of the 2007 EuroSys Conference*, 305–314.

Warner, J., and Atluri, V. 2006. Inter-instance authorization constraints for secure workflow management. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, 190–199.